1.02 LinuX Enterprise präsentiert:

Deutschland 9,80 € Österreich 10,80 € / Schweiz 19,20 SFR Luxemburg 11,25 €

S&S

PHP Magazin

PHP Magazin

mit CD

Web Services

SOAP Implementierungen für PHP im Einsatz

Start-up
Objektorientierung in PHP

Do it yourself!
CMS im Eigenbau mit MySQL und PHP

PHP und Flash
XML-basierter Datenaustausch

Auf der Überholspur
MySQL mit InnoDB

Magie oder Handwerk?
Reguläre Ausdrücke – Teil I

Die Magazin-CD

Testversionen:
Zend Studio 2.0 Client/Server
ASTARTE webEdition 1.3.0.2

- PHP-Releases
- CMS-Systeme
- Datenbanken
- PHP-Tools
- Application- und Webserver
- Browser

4 195649 609809 01

Anzeige

Anzeige

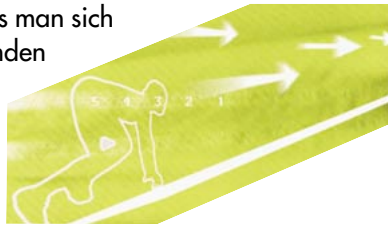
PHP Magazin Inhalt März 2002

10 „PHP ist ideal als Web Frontend“

Im Umfeld der Open Source Skriptsprache PHP haben sich bereits verschiedene Unternehmen etabliert, die Tools und Entwicklungsumgebungen für den PHP-Einsatz im High End Bereich entwickeln. Wir sprachen mit Tobias Ratschiller, CEO der Maguma AG, über die PHP4 Enterprise Edition, die von seinem Unternehmen entwickelt wird, und über den Einsatz von PHP in Unternehmen.

21 Hoch damit! Aber sicher ...

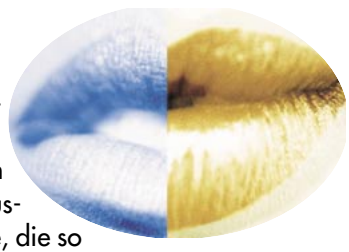
HTML-Formulare zum Datei-Upload lassen sich mit PHP in nur wenigen Zeilen einfach implementieren, ohne dass man sich dabei mit den Einzelheiten der zugrunde liegenden Mechanismen auseinander setzen muss. Dieser Komfort täuscht allerdings darüber hinweg, dass für robuste und sichere Anwendungen trotz allem bei Uploads noch ein paar wichtige Dinge zu beachten sind.



27 Planungssicherheit – Software-Engineering mit PHP

PHP wird besonders von Programmier-Einsteigern geschätzt, weil sich damit schnell eine sonst statische Website um programmierte Funktionalität ergänzen lässt. Auf den zweiten Blick erkennt man, dass PHP auch zur Entwicklung ausgewachsener Web-Applikationen geeignet ist. Wer sich vom Skript-Programmierer zum Entwickler komplexer Web-Applikationen aufschwingen will, sieht sich plötzlich mit ganz neuen Herausforderungen konfrontiert. Damit das erste größere Projekt keine Bauchlandung wird, ist fundiertes Hintergrundwissen hilfreich.

40 Schwarze Magie oder Handwerk?



Man sagt, ein PHP-Entwickler sollte neben der Programmiersprache PHP auch HTML beherrschen. Auch SQL- und JavaScript-Kenntnisse machen sich in einem Bewerbungsbogen gut. Aber reguläre Ausdrücke? Das sind doch diese kryptische Ausdrücke, die so aussehen, als hätte man sich beim Telefonieren auf die Tastatur gelehnt? Unsere zweiteilige Serie macht Sie mit den Geheimnissen Regulärer Ausdrücke vertraut.

Enterprise

Startup

Tools & Tipps

Development

<XML...> Magazin

Datenbanken

Solutions

Rubriken

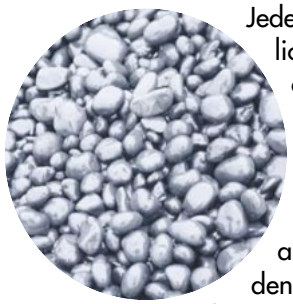
55 Brücke an Maschinenraum

PHP Skripte werden vom Betriebssystem genau wie alle anderen Prozesse ausgeführt und laufen somit in einem gekapselten Speicherbereich. Dadurch wird ein hohes Maß an Sicherheit gewährleistet. Andererseits wird damit die Kommunikation zwischen den parallel laufenden Prozessen unterbunden, da jeder Prozess in der Illusion lebt, den Server für sich alleine zu haben. Was jedoch, wenn die Applikation explizit verlangt, dass mehrere gleichzeitig laufende Prozesse Daten austauschen?

65 SOAP Web Services mit PHP

Web Services und speziell das Simple Object Access Protocol SOAP sind momentan in aller Munde. Schaut man hinter den Hype der Web Services, so findet man einen portablen, in vielen Sprachen und Plattformen einsetzbaren Standard zur Interoperabilität. Auch für PHP gibt es eine Reihe von SOAP-Implementierungen, von denen hier eine vorgestellt und in einem praktischen Beispiel eingesetzt werden soll.

77 MySQL auf der Überholspur



Jeder professionelle PHP Entwickler stellt sich mehrfach täglich die Frage: „Wie kann ich Daten am besten speichern?“. Die häufigste Antwort auf diese Frage ist MySQL. Nur, was machen, wenn es sich nicht mehr um einfache Webapplikationen sondern um komplexe Anwendungen handelt und Funktionen wie Transaktionen, „Referenzielle Integrität“ oder erhöhte Performance unabdingbar sind? Mit InnoDB muss sich MySQL auch vor den ganz Großen nicht mehr verstecken und kann durchaus überzeugen.

CD

Quellcodes zu den Artikeln im Heft

PHP-Releases:
PHP 4.1.1 und ältere Versionen

PHP-Tools:
APBoard 2.02a
DBG 2.10
Komodo 1.2.5
Pepe 0.5.1 pre
Phorecast 0.42
Phorum 3.3.2a
PHPDoc 1.0 beta
phpEdit 0.6
phpGroupWare 0.9.12
PHProjekt 3.1a

CMS-Systeme:
CMF 1.2
eZ Publish 2.2.2
Midgard 1.4.2-1
open-medium CMS 0.11 beta
phpCMS 1.1.6 pl1
PHP-Nuke 5.4
phpWebSite 0.8.1.1
Typo3 3.3 beta1.3

Datenbanken:
FreeTDS 0.53
FireBird Kit 1.0 rc2
GigaBASE 2.57
HypersonicSQL 1.61
InterBase 6.5
MySQL 3.23.47
phpMyAdmin 2.2.3
PostgreSQL 7.2

Application- und Webserver:
Apache Tomcat 4.0.2 beta2
Apache Webserver 1.3.23
JOnAS 2.4
MyWebServer 1.0.1
Zope 2.5.0

Browser:
Mozilla 0.9.8

Liebe Leserin, lieber Leser!

Ich freue mich ganz besonders, Ihnen die erste Ausgabe des PHP Magazins präsentieren zu dürfen. Der Vorgänger, das PHP Sonderheft von Linux Enterprise, erfreute sich so großer Beliebtheit, dass der Software & Support Verlag sich dazu entschlossen hat, ein regelmäßiges PHP Magazin heraus zu geben. Das Magazin ist in sechs verschiedene Rubriken eingeteilt, die Ihre unterschiedlichen Interessen bedienen sollen: StartUp, Enterprise, Tools & Tipps, Development, Datenbanken, Solutions.

Die beiden Hauptrubriken bilden Development und Datenbanken. Hier finden Sie Artikel zur (Web-) Entwicklung mit PHP und den spezifischen Aspekten dynamischer Websites in Hinblick auf die Datenbank-Anbindungen. In einigen Fällen dürfen Sie auch Artikel erwarten, die auf den ersten Blick gar nichts mit PHP zu tun haben, sondern eher theoretisches Hintergrundwissen vermitteln, das Sie bei größeren PHP-Projekten durchaus gebrauchen können.
Der Tatsache, dass PHP immer wichtiger in eBusiness Projekten und kommerziellen Applikationen wird, wollen wir mit den Rubriken Enterprise und Solutions Rechnung tragen. Gerade in den letzten Monaten tauchen vermehrt Anbieter auf, die kommerzielle Services rund um PHP bieten. Dies wird sich auch in Zukunft fortsetzen und wir wollen dies in der Rubrik Enterprise beobachten. Mit den Solutions erwarten Sie Case Studies von PHP-Projekten.

Tools & Tipps vereint Softwaretests und (Buch-) Rezensionen. Bei den Büchern möchten wir auch hier das Augenmerk nicht nur auf PHP-Bücher setzen, sondern auf alles, was in diesem Zusammenhang für den PHP-Entwickler wichtig ist.
Ab Ausgabe 2/2002 wird Sebastian Bergmann, der in der PHP-Szene kein Unbekannter ist, in seiner Entwickler-Kolumne über die Neuigkeiten in der PHP-Welt berichten. Schreiben Sie uns, was Sie von dieser Ausgabe halten! Ihre Meinung interessiert uns: redaktion@phpmag.de.
Und sollten Sie als Autor mitarbeiten wollen, interessante PHP-Projekte erstellt haben oder PHP-Services anbieten, dann wenden Sie sich an mich: bjoern@phpmag.de.

Viel Spaß beim Lesen!

Björn Schotte, Chefredakteur

Apache: Covalent promotet Enterprise Ready Server

Covalent Technologies hat auf der LinuxWorld Expo in New York den Enterprise Ready Server, der auf Apache 2.0 basiert, vorgestellt. Covalent bietet seit längerer Zeit eigene Apache-Lösungen an, die mit kommerzieller, proprietärer Software gebündelt werden. Der Covalent Enterprise Ready Server basiert auf dem Kern von Apache 2.0 und unterstützt Multithreaded Processing. Zur Verschlüsselung dient Covalent SSL. Außerdem bietet Covalent ein Enterprise Management Portal, das zur Senkung der Total Cost of Ownership beitragen soll. Der Covalent Logging Service dient der Log-Konsolidierung. Außerdem werden Statistiken zur Serverperformance angezeigt.
www.covalent.com/

Rekall 1.0 für Linux erschienen

theKompany.com hat das Final Release des RAD DBMS Tools Rekall freigegeben. Mit Rekall 1.0 lassen sich auf der Basis von Rekall Forms und Reports Datenbank-Applikationen entwickeln. Die Software unterstützt bisher die Datenbanken xBase, MySQL und PostgreSQL. Bisher ist Rekall nur für den KDE Desktop unter Linux erhältlich, doch ab dem nächsten Major Release soll die Software auch für Windows zur Verfügung stehen. Auch die Unterstützung von MacOS ist geplant.
www.thekompany.com/

phpOpenTracker in Version 0.9.9 freigegeben

PHP-Entwickler Sebastian Bergmann hat Version 0.9.9 seines Tools phpOpenTracker freigegeben. Die Software dient der Messung des Traffics auf Websites und der Analyse des Benutzerverhaltens im Hinblick auf Webapplikationen. Den Kern von phpOpenTracker bildet die so genannte Logging Engine, die Daten über Besucher und Zugriffe in einer Datenbank ablegt. Mit dem neuen Plugin System ist es möglich, die Software zu erweitern und den eigenen Bedürfnissen anzupassen. Mit dem aktuellen Release, das das letzte vor der Final Version 1.0 ist, sind die APIs als stabil zu bezeichnen, so Sebastian Bergmann. Anwender brauchen PHP in der Version 4.1.0 oder neuer sowie eine MySQL, PostgreSQL oder Oracle Datenbank.
www.phpopentracker.de/

localizer-0.2.0 freigegeben

Die neue Version des localizer vermag es, für mehr als 9.000.000 IP-Adressen die entsprechenden Orte zu bestimmen. Je nach Provider lässt sich dabei ein User auf einen Radius von bis zu 25 Kilometer lokalisieren. Unterstützt werden alle großen deutschen Internet-Provider und deren Backbone-Anbieter. Die PHP-Extension, die die Funktionalität des localizers innerhalb von PHP bereitstellt, vermag es, Provider, Ort, Region und Land anhand der IP innerhalb von wenigen Millisekunden zu ermitteln. Neben deutschen Providern sind auch die ersten niederländischen Provider aufgenommen worden. Weitere europäische Internet-Anbieter sind in Vorbereitung.
jan.kneschke.de/projects/localizer/

Kommerzieller PHP-Support und Performance-Messungen

ThinkPHP erweitert seine Produktpalette um kommerziellen PHP-Support und Performance-Messungen von PHP-Applikationen. Das Supportangebot umfasst Einzeleinsätze (Single Incidents), längerfristige Supportverträge und telefonische Bereitschaft mit vertraglich zugesicherten Reaktionszeiten. Ergänzt wird dieses Angebot durch den Support von Fremdsoftware und von bekannten PHP-basierten Open Source Projekten.
Die Performancemessungen erlauben aufgrund einer neuen Technologie realitätsnähere Szenarien als die bekannten Benchmark-Tools ab (ApacheBench) und HttPerf. Es werden unterschiedliche konkurrierende aktive Nutzungsvorgänge simuliert, sodass Bottlenecks, Lock- und Race-Conditions erkannt werden können. Eingebettet wird dieses Angebot in eine detaillierte Analyse und das Aufzeigen von Lösungsstrategien.
www.thinkphp.de/
team@thinkphp.de

XPath Bibliothek Pathan freigegeben

Decisionsoft hat die Bibliothek Pathan unter eine Open Source Lizenz gestellt. Die Bibliothek soll XPath Parsing und Evaluierung ermöglichen und ist für den Einsatz mit dem Xerces-C Parser des Apache Projekts bestimmt. Die Bibliothek steht als Quelltext oder in Form vorkompilierter Pakete für Linux, Solaris und Windows zum Download bereit.
software.decisionsoft.com/

Developer Tools

PHP 4 Enterprise Edition in den Startlöchern

Maguma befindet sich mit seiner PHP 4 Enterprise Edition auf der Zielgeraden. Auf der International PHP Conference, die im November 2001 in Frankfurt stattfand, stellte das Unternehmen aus Bozen bereits einen Prototypen vor. Das Final Release soll bis zur CeBIT im März das Licht der Webwelt erblicken.
PHP4EE ist von Maguma als Rundum-sorglos-Paket für Webentwickler konzipiert. Die Lösung beinhaltet eine integrierte Entwicklungsumgebung zur Unterstützung der schnellen Anwendungsentwicklung sowie eine Runtime Library und eine Laufzeitumgebung zur Publikation und Überwachung der Applikationen auf dem Server.
Das System basiert auf LAMP-Technologien. Der Linux-Server kommt mit dem Apache Webserver, als Datenbank wird MySQL eingesetzt. Zur Versionskontrolle kommt CVS zum Einsatz. Das System ist so konzipiert, dass man mit Mehr-Server-Architekturen arbeiten und getrennte Entwicklungs-, Live- und Staging-Server betreiben kann. Die Client-Software wird zunächst nur für Windows-Systeme ausgeliefert.
Mit der Entwicklungsumgebung bietet Maguma einige unterstützende Tools, beispielsweise einen Class Browser, der eine strukturierte Übersicht über Klassen, Funktionen und Variablen von Skripten ermöglicht. Class Insight dient zur Auflistung aller Eigenschaften und Methoden einer Klasseninstanz.
Maguma richtet sich mit seinem Produkt nach eigenen Angaben vor allem an professionelle Programmierer wie PHP-Entwickler, Systemadministratoren, ISPs sowie Web- und Systemintegratoren und hat damit einen ähnlichen Markt im Visier wie Zend Technologies. Als Features für zukünftige Versionen werden denn auch Load Balancing, Failover und Clustering angekündigt. Außerdem soll die PHP4EE um einen SQL Editor erweitert werden und in Nachfolgeversionen weitere Architekturen unterstützen. *(nar)*
www.maguma.com/

News & Trends

Solomon Webinterface für MySQL

Mit Solomon steht ein Perl-basiertes Webinterface für MySQL-Datenbanken zur Verfügung. Das Skript zeigt die Inhalte von Tabellen in einer HTML-Tabelle an. Die Software beinhaltet ein Modul für den einfachen Aufbau von SQL Connections und unterstützt Datenmodifikationen, Suchfunktionen und das Einfügen und Löschen von Daten. Entwickler können auf das Development Release 2.1 zurückgreifen, außerdem gibt es ein Stable Release.
www.arches.uga.edu/~cygnusx5/

Nusphere unterstützt PHP-basierte Web Services

Die Nusphere Corporation hat die integrierte Unterstützung für die Entwicklung Open Source-basierter Web Services angekündigt. Das Paket für Entwickler soll aus Nuspheres IDE PHPEd und Open Source Technologien wie Apache, Perl und MySQL geschnürt werden. Das Unternehmen will mit dieser Lösung eine Alternative zu Ansätzen wie Microsoft .NET und J2EE bieten, die zum großen Teil auf proprietäre Tools setzten und Entwickler in Herstellerabhängigkeiten zu bringen drohten. Auf der Nusphere Website werden u.a. White Papers und Anwendungsbeispiele zur Verfügung gestellt, mit denen gezeigt werden soll, dass es sich bei PHP um „das schnellste und kostengünstigste" Tool zur Entwicklung von Web Services handelt.
www.nusphere.com/

Apache 1.3.23 freigegeben

Die Apache Software Foundation hat Version 1.3.23 des Webservers freigegeben. Dabei handelt es sich vor allem um ein Bugfix-Release. Außerdem wurden Verbesserungen am mod_proxy vorgenommen. „Unserer Ansicht nach ist der Apache 1.3.23 die beste verfügbare Version, sodass wir Anwendern raten, von älteren Versionen umzusteigen", äußerte die Foundation.
www.apache.org/dist/httpd/Announcement.html

CVS Erweiterung TkCVS 7.0.3 verfügbar

Mit TkCVS steht eine Tcl/Tk-basierte grafische Oberfläche für das CVS Konfigurationsmanagement zur Verfügung. Die Software erlaubt die Darstellung des Status von Dateien im Arbeitsverzeichnis und stellt Menüs und Buttons zur Ausführung von CVS Befehlen zur Verfügung. TkCVS läuft unter Unix und Windows.
www.twobarleycorns.net/tkcvs.html

PHP Accelerator 1.2 verfügbar

Der PHP Accelerator für Linux steht in Version 1.2 zum Download bereit. Das Caching Tool dient zur Beschleunigung der Ausführung von PHP-Skripten. Die neue Version umfasst neben verschiedenen Bugfixes neue Features wie parallelen Read/Write Cache Access, Crash/Instability Detection und ein Cache Display Tool.
www.php-accelerator.co.uk/

Opera 6 für Linux mit 3. Technology Preview

Opera Software hat von der Linux-Version des Browsers in der Version die 3. Technology Preview herausgebracht. Es wurden gegenüber der TP2 verschiedene Bugfixes im Hinblick auf das User Interface vorgenommen.
Außerdem lassen sich laut Anbieter nun die Plugins leichter handeln. Weiter wurden neue Funktionalitäten zur Anpassung der Oberfläche vorgenommen.
www.opera.com/linux/

Content Management

Astarte wirbt mit webEdition um den Mittelstand

Mit ihrem neuen Content Management System webEdition will die Astarte New Media AG den Markt für CMS aufrollen. Das Unternehmen, das Mac-Fans vor allem durch den DVDirector, der Anfang 2000 an Apple verkauft wurde, und die CD-Recording Software Toast ein Begriff sein dürfte, bietet mit seiner „Astarte webEdition" eine Content Management Lösung, die analoge Technologien wie High End Systeme nutzt, aber potenzielle Kunden durch einen wesentlich niedrigeren Preis überzeugen soll.
Das von Astarte geschnürte Paket basiert auf PHP4 und nutzt als Datenbank MySQL, was den günstigen Einstiegspreis von 159 Euro für eine Single-Domain-Lizenz ermöglicht. Die Software richtet sich vor allem an kleinere Unternehmen, die mit geringem Personalaufwand ihre Websites realisieren wollen. Deshalb verfügt das Softwarepaket auch über einen Installer, der die Einrichtung des Systems erleichtert, indem er beispielsweise die Konfiguration des entsprechenden Servers erkennen kann.
Das CMS unterstützt das Arbeiten mit Vorlagen, wobei für die Bearbeitung der Seiten keine HTML-Kenntnisse notwendig sind. Das System ist Browser-gesteuert. Verwendet werden können alle gängigen Browser, also auch Netscape und Opera, wobei im Internet Explorer zusätzlich eine WYSIWYG-Ansicht zur Verfügung steht.
Die Architektur basiert auf einer Sammlung von Tags, die Astarte in Anlehnung an XML entwickelt hat. Diese so genannten we:tags lagern Funktionen in eine Funktionsbibliothek aus und integrieren sie beim Seitenaufruf.
Astarte bietet neben der Single-Domain-Lizenz auch Lizenzen für fünf oder 20 Domains an, sodass die Software auch für Agenturen interessant ist, die auf der Basis der webEdition Kundenlösungen entwickeln wollen.
Das bereits bestehende System wird von Astarte durch weitere Module ergänzt. So stehen seit Dezember 2001 drei neue Module zur Verfügung. Diese stellen Benutzerverwaltung, ein Editor-Plugin und Scheduler-Funktionalität bereit. Für das Basismodul steht außerdem ein Update zur Verfügung.
Mit der neuen Benutzerverwaltung lassen sich die Zugriffsrechte, beispielsweise zwischen Administrator und Redakteur, differenzieren. Mit dem Editor-Plugin können Editoren wie Dreamweaver, GoLive oder FrontPage in das System integriert werden. Auch die Einbindung anderer Editoren ist laut Hersteller möglich. Der Scheduler wiederum ermöglicht zeitversetzte Publikationen, sodass erstellte Dokumente automatisch zu einem definierten Termin freigeschaltet werden können.
Laut Herstellerangaben lassen sich bereits mit der Basisversion 80 Prozent aller im Content Management anfallenden Aufgaben bewältigen. Ein Großsystem, das ab Mitte 2002 fertiggestellt sein soll, werde mit seinen Modulen rund 7.500 Euro kosten, was immer noch deutlich unter den Preisen vieler High End Anbieter liegt. (nar)
www.webedition.de/

Adaptive Website Framework in Version 1.02 freigegeben

Das Adaptive Website Framework liegt in Version 1.02 vor. Mit der Software, die auf PHP 4 und MySQL basiert, lassen sich personalisierte Websites erstellen. Das Framework verfügt über verschiedene Features für das Content Management wie Suchmodul, Caching und verschiedene Schnittstellenfunktionen.

liquidbytes.net/awf/

Newsmanagement unter PHP mit MT News

MT News ist ein PHP Skript, das die Implementierung von Kommentar- und Newsarchivierungs-Funktionalitäten ermöglicht. Auch Posting-Funktionen wie Editieren und Löschen sind abgedeckt. Als Datenbank wird MySQL verwendet. Das Skript liegt in der Entwicklerversion 1.4b vor.

mtnews.mactavern.com/

Datenbank Reports mit DataVision 0.1.0

Das Datenbank Reporting Tool DataVision wurde in Version 0.1.0 freigegeben. Das in Java geschriebene Werkzeug unterstützt Datenbanken wie PostgreSQL, MySQL, Oracle und ODBC und kann zur Ausgabe verschiedene Frontends wie HTML, DocBook oder LaTeX verwenden. Die Reports werden in XML beschrieben und abgelegt.

www.io.com/~jimm/downloads/datavision/

Postscript und PDF mit PHP

Die Bibliothek AxisPHP stellt PHP-Objekte zur Erstellung von Postscript- und PDF-Dokumenten bereit. Außerdem verfügt sie über Objekte für zustandslose Sessions und das Handling von User Accounts. Die Session- und User-Objekte legen ihre Daten via Abstraktionslayer in einer MySQL-Datenbank ab, sodass die Daten leicht auch an andere SQL-Datenbanken angepasst werden können. Die Bibliothek steht in Version 1.2.4 zur Verfügung.

www.axisdata.com/AxisPHP/

Konkurrenz für Zend? – Komodo 1.2 von ActiveState

ActiveState, Anbieter von Tools und Entwicklungsumgebungen für verschiedene Skriptsprachen, hat seine Entwicklungsumgebung Komodo als Version 1.2 Beta freigegeben. Für PHP-Entwickler interessant ist vor allem ein Regular Expression Toolkit für PHP 4.1 sowie der Debugger ActiveDebug.

www.activestate.com/Products/Komodo/

Webdevelopment

PHP wird immer mehr zum Renner

PHP ist ungebrochen auf Erfolgskurs. Nachdem die Skriptsprache im Juli 2001 erstmals auf sieben Millionen Websites eingesetzt worden war, waren die Zahlen von Netcraft zunächst wieder ein wenig eingebrochen, was im Zweifel lediglich an Routing-Problemen im deutschen Raum gelegen haben dürfte, die dazu führten, dass zwischen fünf und zehn Prozent der deutschen Server nicht auf die Netcraft-Anfragen antworten konnten. Nachdem dieses Problem behoben ist, zeigt der Pfeil nun weiter nach oben. Für November 2001 meldete Netcraft allein den Einsatz von PHP auf knapp 7,1 Millionen Websites.

Auch der Apache Webserver ist weiter in einer Aufwärtsbewegung begriffen. Die Netcraft Server-Studie vom Januar 2002 hält für den Apache einen Marktanteil von 63,97 Prozent fest, was einem Zuwachs um 0,47 Prozentpunkte gleichkommt. Auf Platz zwei rangieren Microsoft-Server mit 26,08 Prozent, was einem Minus von 0,37 Prozentpunkten gleichkommt. Netscape folgt mit einigem Abstand mit 2,33 Prozent (Minus 0,04 Prozentpunkte). Der Zeus Webserver wird auf 1,08 Prozent der abgefragten Server eingesetzt, was ein Plus von 0,01 Prozentpunkten bedeutet. Die Anfragen von Netcraft haben insgesamt 4,43 Millionen Webserver erfasst.

Zu den am meisten eingesetzten Apache-Modulen gehören PHP, OpenSSL, mod_ssl, FrontPage und perl. PHP wurde im Dezember 2001 auf insgesamt 46,49 Prozent der mit Apache laufenden Webserver eingesetzt. Die Verwendung von OpenSSL erweiterte sich um 27,37 Prozentpunkte auf 24,64 Prozent der Apache Server. mod_ssl wird gegenwärtig auf 23,83 Prozent der Server eingesetzt (Plus 4,94 Prozentpunkte). FrontPage ist auf 21,52 Prozent der Server im Einsatz (Plus 17,51 Prozentpunkte) und Perl auf 18,87 Prozent . *(nar)*

www.netcraft.com/survey/

Content Management Systeme

Vom Content Management zum eCommerce

Ein Content Management System mit Erweiterungsmöglichkeit um umfassende eCommerce Funktionalität bietet die Omeco GmbH aus Kaiserslautern. Das CMS „omeco webcontent" besteht aus einem Systemkern mit Datenbankserver und verfügt über ein Internet-Frontend sowie Intranet/Extranet-Anwendungen auf der einen und eine Backoffice-Verwaltungsschnittstelle auf der anderen Seite. Eine Erweiterung um ein Online-Shop-System ist durch „omeco webshop" möglich.

Das CMS ist mit einer PHP4 Scripting Engine ausgestattet und verfügt laut Hersteller über eine Reihe Standard-Funktionen, die in Templates flexibel eingebettet werden können. Ein komplexes System zur Benutzersteuerung und Rechteverwaltung regelt die Zusammenarbeit von Redakteuren bei der Erstellung von Inhalten. Ein integriertes Messaging Verfahren sowie automatisch erstellte ToDo-Listen sollen laut Omeco für einen reibungslosen Workflow sorgen. Das Content Management System basiert vollständig auf dem LAMP-Ansatz. Es besteht aus einem Linux-System, einem erweiterten Apache-Webserver, dem MySQL-Datenbankserver und PHP4. Zur Verschlüsselung wird SSL 3.0 eingesetzt. Der Zugang zum Extranet lässt sich außerdem über gesicherte PC-Arbeitsplätze mit Smartcard-Lesegeräten einrichten. Neben den Grundfunktionalitäten bietet der Hersteller noch verschiedene Module zur Erweiterung des Systems an. Mit dem „advanced scripting" Modul sollen sich leistungsfähige Zusatzfunktionen in Webseiten integrieren lassen. Mit „extranet solution" lässt sich laut Hersteller die Zugangsrechteverwaltung erweitern. Der „pdf generator" erstellt automatisch PDF-Dateien aus Webseiten, wobei die PDF-Dateien bei Änderungen an der Webseite automatisch aktualisiert werden. Mit dem „cd-rom generator" lassen sich Internet-Auftritte vollständig auf CD extrahieren. Dabei werden notwendige Navigationselemente automatisch erstellt. Als Einstiegspreis für „webcontent" nennt Omeco einen Preis von 4.900 Euro. *(nar)*

www.omeco.de/

Tobias Ratschiller von Maguma

„PHP ist ideal als Web Frontend“

von Nadja Rosmann

Im Umfeld der Open Source Skriptsprache PHP haben sich bereits verschiedene Unternehmen etabliert, die Tools und Entwicklungsumgebungen für den PHP-Einsatz im High End Bereich entwickeln. Wir sprachen mit Tobias Ratschiller, CEO der Maguma AG, über die PHP4 Enterprise Edition, die von seinem Unternehmen entwickelt wird, und über den Einsatz von PHP in Unternehmen.

Linux Enterprise: Für welche Anwendergruppen konzipiert Maguma die PHP4 Enterprise Edition?

Tobias Ratschiller: Wir konzentrieren uns auf PHP- und Webentwickler. PHP4EE ist ein Tool, das professionellen Entwicklern hilft, Zeit zu sparen und sicherer zu entwickeln, also die Anwendung sicherer auf die Laufzeitumgebung zu bringen. Außerdem adressieren wir auch weniger versierte Programmierer, die mit einem visuellen Editor komplexe Anwendungen erstellen können. Damit gehören auch Einsteiger zu unserer Zielgruppe. Wir gehen gegenwärtig auch Kooperationen mit Trainingspartnern ein, die die PHP4EE in ihren Schulungen einsetzen werden. Unser Fokus liegt auf dem kommerziellen Bereich, weil hier die Notwendigkeit gegeben ist, schneller zu entwickeln, in kürzeren Abständen Updates herauszubringen und diese dann auch so zu verwalten, dass sie sicher laufen.

LE: Welche Features umfasst die Entwicklungsumgebung?

Ratschiller: PHP4EE besteht aus drei Hauptkomponenten. Das sind zwei Tools beim Client, die also beim Entwickler laufen, und zwar der Quellcode-Editor PHP4EE Studio, der auf PHPCoder basiert, und der Visual Builder, der eine Trennung zwischen Applikationslogik und Design ermöglicht. Komplexe Applikationen lassen sich damit sehr rasch entwickeln. Wir haben einige Tests mit Applikationen mittlerer Größe angestellt und sind zu dem Ergebnis gekommen, dass man damit die Entwicklungszeit auf 50 Prozent der ursprünglichen Zeit senken kann. Für die Verwaltung und das Deployment der Applikationen, die mit dem Quellcode-Editor oder mit dem Visual Builder entwickelt wurden, haben wir den PHP4EE Server entwickelt. Dabei handelt es sich um eine Web-basierte Management-Oberfläche für Apache, MySQL und CVS, die es ermöglicht, die Applikationen sicher und performant auf die Website zu bringen. Wir haben hier beispielsweise vordefinierte Netzwerkarchitekturen wie die Trennung zwischen MySQL, Apache und CVS. Der Entwickler muss damit nicht mehr direkt auf die Konfigurationsdateien zugreifen, sondern kann alles über die Web-Oberfläche verwalten. Und wir machen die Entwicklung sicherer, indem wir eine Versionskontrolle integriert haben. Damit können Teams effizienter zusammenarbeiten. Wir haben zudem eine Trennung zwischen Entwicklungs- und Live-Server, sodass sich Updates sicher bewerkstelligen lassen. Die Vorteile sind also größere Produktivität, bessere Wartbarkeit und schnellerer und sicherer Einsatz von PHP-Applikationen.

LE: Für Projekte welcher Größenordnung ist PHP denn Ihrer Meinung nach inzwischen reif?

Ratschiller: Prinzipiell wird PHP heute in sehr verschiedenen Bereichen eingesetzt. Das reicht von Hobbyapplikationen bis hin zu überraschend komplexen Anwendungen im Enterprise Bereich. Wir sehen PHP am besten geeignet für das Web Frontend. In einer Enterprise Umgebung hat man die Businesslogik in einer Middleware laufen, als Enterprise Java Beans oder als COM-Komponenten, und mit PHP kann man sehr rasch ein Frontend

entwickeln. Das wird durch die gute Konnektivität in die vorhandene Logik integriert. Wir glauben auf jeden Fall, dass sich PHP in diesem Bereich weiter etablieren wird.

LE: Und sehen Sie auch spezielle Ansatzpunkte, wo PHP weiterentwickelt werden muss?
Ratschiller: Die gibt es wahrscheinlich immer. Wobei PHP dank der Open Source Community sich so rasch am Weiterentwickeln ist, dass fast jede Woche ein neues Update kommt. Und die Entwickler um Zend, die Kerngruppe, arbeiten bereits an einer neuen Version des Sprachkerns, die noch besser für komplexe Anwendungen geeignet ist.

LE: Maguma arbeitet ja an einer PHP4 Integration Edition zur Erstellung von Unternehmensportalen, die auf IBM WebSphere basieren. Warum setzen Sie hier gerade auf IBM? Sind auch Kooperationen mit anderen Application Server Anbietern geplant?
Ratschiller: Rund um unser Produkt PHP4EE werden einige Editionen entstehen. Das Eine ist die PHP4EE Voice Edition, die die Entwicklung von Voice Applikationen einfacher macht und den IBM Voice Server benutzt. Das Andere ist, wie Sie bereits angesprochen haben, die PHP4EE Integration Edition, die als Middleware IBM WebSphere benutzt. WebSphere ist heute die führende Middleware. Zusätzlich haben wir eine gute Partnerschaft mit IBM und werden daher in der ersten Version IBM WebSphere unterstützen.

LE: Planen Sie hier eine Ausweitung auf andere Application Server?
Ratschiller: Mittelfristig auf jeden Fall.

LE: Im Markt der PHP Enterprise Lösungen spielt Zend Technologies mit seinen Produkten ja bisher eine führende Rolle. Unterscheidet sich das Maguma Angebot vom Zend Portfolio?
Ratschiller: Mit Zend verbindet uns, dass wir ein kommerzieller Anbieter von PHP-Produkten sind. Mit unseren Lösungen bewegen wir uns auf einem höheren Level als Zend, die hauptsächlich Tools und Add-Ons für die Sprache anbieten. Wir versuchen, einen Schritt nach oben zu gehen und eine vollständige Umgebung für die Sprache und die Entwicklung mit PHP anzubieten. Wir haben ein sehr gutes Verhältnis zu Zend. Wir haben eine Kooperation und werden sicher in Zukunft bei einigen Sachen zusammenarbeiten.

LE: Zend ist ja im Prinzip direkt aus der PHP Community hervorgegangen und verfügt über die Hauptentwickler der Skriptsprache. Welchen Zugang zum Kern-Know-how von PHP hat Maguma?

> „PHP ist bestens geeignet als Web Frontend im Enterprise."

Ratschiller: Unsere Entwickler sind PHP-Experten und sie sind auch in der Community tätig, wobei einige von ihnen aktiv als Autoren, als Verfasser von Open Source Programmen und in der PHP-Qualitätssicherung tätig sind. Auf der anderen Seite geben wir direkt etwas an die Community zurück, indem wir beispielsweise PHP4EE Studio in einer full-featured Version kostenlos zur Verfügung stellen. Der einzige Unterschied zwischen Light- und Vollversion ist, dass die Light-Version das Interface für PHP4EE Server nicht hat, das für Standalone-Entwickler nicht unbedingt ausschlaggebend ist, aber für kommerzielle Projekte in einer bestimmten Größenordnung wichtig ist.

LE: Sie bieten die Server Software unter Linux an, die Clients unter Windows. Wie begründet sich diese Auswahl? Planen Sie auf der Serverseite auch Versionen für Unix-Derivate? Und wie sieht es mit Linux Clients aus?
Ratschiller: Auf der Clientseite sind auf jeden Fall auch Linux-Versionen geplant, wahrscheinlich schon kurz nach dem ersten Release. Wir verwenden Delphi und Java auf der Clientseite und sehen deshalb keine Schwierigkeiten, das auf Linux zu portieren. Auf der Serverseite werden Windows und Unix-Derivate hinzukommen. Wir wollen alle PHP-Plattformen unterstützen.

LE: Wie sieht Ihre weitere Roadmap für PHP4EE aus?
Ratschiller: Mit unseren verschiedenen Editionen wollen wir verschiedene Gebiete abdecken. Auf der anderen Seite haben wir als Weiterentwicklung von PHP4EE bestimmte Netzwerkfeatures geplant. Damit wollen wir das Deployment von großen PHP-Anwendungen noch performanter machen. Wir werden Load Balancing und Clustering unterstützen und verschiedene Fault Tolerance Systeme anbieten, die auch sehr große PHP-Anwendungen bedienen können.

LE: Wie sieht es mit der Webserver-Unterstützung aus? Ist da nur der Apache eingeplant oder auch andere Webserser?
Ratschiller: In den nächsten sechs Monaten wird sich die Entwicklung auf den Apache beschränken. Weitere Unterstützungen sind natürlich vorstellbar, aber es besteht im Moment hier keine große Nachfrage.

LE: Herr Ratschiller, wir danken Ihnen für das Gespräch.

Unser Gesprächspartner

Tobias Ratschiller ist CEO der Maguma AG und zeichnet vorrangig für die strategische Entwicklung des Unternehmens und den Aufbau von Partnerschaften verantwortlich. Vor der Firmengründung im Mai 2000 war Ratschiller als Berater am Aufbau und der Implementierung großer PHP-basierter Anwendungen international beteiligt. Ratschiller veröffentlichte bereits mehrere Bücher und Artikel über PHP, leitete in ganz Europa PHP-Seminare und tritt seit mehreren Jahren als Speaker auf Kongressen auf.

Das Unternehmen

Die Maguma AG positioniert sich seit Mai 2000 als Player im PHP- und Webdevelopment-Markt. Mit der Entwicklungsumgebung PHP4 Enterprise Edition bietet das Software-Unternehmen eine Lösung zum vereinfachten und schnelleren Aufbau komplexer und dynamischer Websites. Damit unterstützt die Maguma AG den Einsatz von PHP im Unternehmensumfeld. Gegründet wurde die Maguma AG von Tobias Ratschiller, CEO, der weltweit als PHP-Experte gilt. Der Hauptsitz des Unternehmens befindet sich in Bozen, Italien.

Kluge Köpfe bauen vor!

von Thorsten Suckow-Homberg

Objektorientierte Programmierung mit PHP

Die in PHP vorimplementierten Funktionen haben zwei große Vorteile: a) sie stehen von Anfang an zur Verfügung und b) man kann sie immer wieder verwenden. Dieser Vorteil kennzeichnet auch die Objekt-Orientierte Programmierung (OOP): Ist das Fundament einmal erstellt, lässt sich der Aufbau einer Applikation durch den Einsatz vordefinierter Objekte erheblich beschleunigen. Denken Sie einmal darüber nach, wie oft Sie schon die Datensätze einer Datenbank-Tabelle auslesen und diese im Browser ausgeben mussten. Um das in Zukunft schnell bewerkstelligen zu können, werden wir eine Klasse programmieren, deren Objekte diese Aufgabe für uns in wenigen Zeilen Quellcode realisieren.

Was ist prozedural? Und was OOP?

Basic ist eine prozedurale Sprache. Wer schon einmal mit Basic gearbeitet hat und den Begriff „prozedural“ noch nie gehört hat, weiß jetzt wahrscheinlich auf Anhieb, was damit gemeint ist: Der Code wird zeilenweise interpretiert. Stellen Sie sich einen Trichter vor, in den Sie oben Ihre Anweisungen hineinwerfen und aus dem diese unten Befehl für Befehl nacheinander herauskommen. Die Objektorientierung geht einen anderen Weg. Java zum Beispiel ist eine Programmiersprache, die komplett auf diesem Konzept aufbaut. Eine prozedurale Programmierung ist hier nicht möglich, da schon das Hauptprogramm eine eigene Klasse, ein Objekt ist. In den meisten Sprachen erlaubt OOP die Spezifikation und Implementierung von abstrakten Datentypen und... Halt! Bevor wir mit der Theorie den Umfang dieses Magazins sprengen, befassen wir uns lieber mit der praktischen Umsetzung. Ohne Theorie geht es aber auch nicht und deshalb werden wir nach und nach auf einige wichtige Begriffe genauer eingehen. Aber zuerst: Wie basteln wir uns eigene Klassen in PHP? Wie erzeugen wir Objekte? Schauen wir uns erst einmal an, was ein Objekt überhaupt ist.

Danke!

Kehren wir für einen Moment in die Welt zurück, die nicht aus Bits und Bytes und Übertragungsprotokollen besteht – betrachten wir für einen Augenblick unsere nicht-virtuelle Umwelt.

Was wir hier als Lampe, Fahrrad oder Herd zu bezeichnen gelernt haben, ist in der OOP nichts anderes als ein Objekt. So ein Objekt hat allerhand Eigenschaften: Eine oder mehrere Farben, es besitzt eine gewisse Größe, hat ein bestimmtes Gewicht und noch vieles mehr.

Kennen Sie „Hello World!“? Es ist egal, welche Programmiersprache Sie neu lernen, die Chancen stehen gut, dass Sie am Anfang eines jeden Lehrbuchs mit dem Quelltext zur Ausgabe der beiden magischen Wörter konfrontiert werden. Und raten Sie mal...die Lehre der OOP kennt etwas Ähnliches: Das Auto! Viele Dozenten und Autoren benutzen ein Auto, um Ihnen die Begriffsbestimmung eines Objekts näher zu bringen. Wir reihen uns nahtlos in diese Tradition ein und abstrahieren ein Auto. Was dann übrig bleibt, ist nichts anderes als ein Objekt.

Dieses Auto-Objekt hat Eigenschaften. Es hat eine Farbe und es hat einen Modell-Namen. Außerdem besitzt es „Methoden“: Es kann die Geschwindigkeit erhöhen, und es kann die Geschwindigkeit verringern, indem es bremst. Zu guter Letzt können wir dieses Auto-Objekt einer „Klasse“ zuordnen: Der Klasse der Fortbewegungsmittel.

Ein Auto hat einen Nutzen: Es kann uns von einem Ort zum anderen transportieren. Außerdem sollte es schick aussehen und nicht zuviel Benzin verbrauchen.

Ein Objekt in der Programmierwelt wiederum ist nichts anderes: Ein Modell, das Eigenschaften und Methoden besitzt und außerdem – hoffentlich – einen Nutzen hat.

Die Klasse, von der wir im Folgenden ein Objekt erzeugen wollen, soll auch einen Nutzen haben: Objekte der Klasse sollen eine MySQL-Datenbankabfrage in wenigen Zeilen Quellcode realisieren und uns die Ausgabe in wunderschönstem HTML auf den Bildschirm zaubern – und zwar in Form einer Tabelle. Die Objekte sollen außerdem Eigenschaften wie Hintergrundfarbe der Tabelle, Text der Fehlermeldungen und frei definierbare Spaltenüberschriften besitzen. Diese Eigenschaften müssen wir noch über Methoden bestimmen können, bevor die Ausgabe auf dem Browser unserer Wahl erscheint. Wir taufen diese Klasse auf den

Anzeige

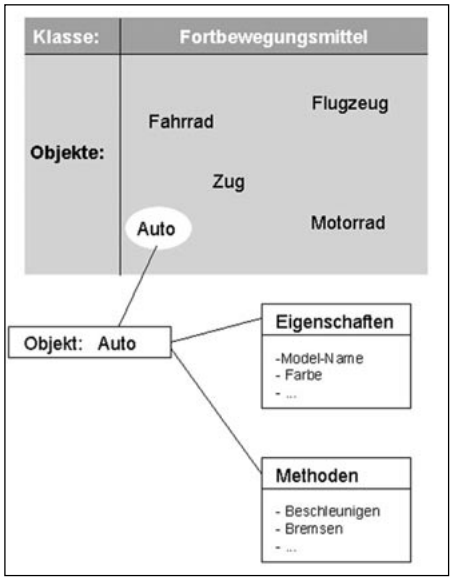


Abb. 1: Beispiel für das Objekt Auto

Namen *PTableGrid* – und während die Klasse Fortbewegungsmittel uns Objekte zur Verfügung stellt, die uns von A nach B bringen, soll die Klasse *PTableGrid* den Nutzen haben, Tabellen einer MySQL-Datenbank abzufragen und die gefundenen Datensätze in HTML auszugeben.

Ohne OOP

Wie kann man diese Aufgabe ohne Klasse bewerkstelligen?

Der Quellcode in Listing 1 erscheint nicht allzu umfangreich – aber auch wenn er schnell angepasst werden kann, gibt es doch schönere Mittel und Wege, eine Datenbankabfrage zu gestalten. Bevor Sie sich zurücklehnen und es sich gemütlich machen: Der Quellcode unserer Klasse wird um einiges umfangreicher.

Sicher ist der Einsatz von Klassen unkompliziert und geht schnell vonstatten. Aber so eine Klasse soll möglichst flexibel und universell einsetzbar sein. Diese Flexibilität zwingt dem Programmierer mehr Arbeit auf: Sollen unschöne Seiteneffekte vermieden werden, müssen schon in der Planungsphase mögliche Eventualitäten berücksichtigt werden: Es kann vorkommen, dass der Einsatz einer Methode der Klasse den Einsatz einer anderen Methode dieser Klasse sowie das Vorhandensein von bestimmten Variablen voraussetzt. Zugunsten der universellen Einsetzbarkeit wird der Quellcode einer solchen Klasse also schnell umfangreich.

Mit OOP

In Funktionen bzw. Methoden haben wir einen Methodenkopf und einen Methodenrumpf (oder auch Methodenkörper). Genau das Gleiche gilt für eine Klasse: Wir haben einen Kopf, der den Namen der Klasse enthält und dem anstelle von *function* das Schlüsselwort *class* voransteht. Dann haben wir einen Rumpf, in den wir die Methoden und Eigenschaften implementieren – gekennzeichnet durch eine sich öffnende und eine sich schließende geschweifte Klammer:

```
class PTableGrid   //Klassenkopf – diese Klasse
                                heißt PTableGrid
 {
          .
          .   //Klassenrumpf – Platz für
                  Methoden und Eigenschaften
        .
 }
```

Wenn wir von Eigenschaften und Methoden von Klassen sprechen, ist das nicht ganz richtig. Eine Klasse besitzt im eigentlichen Sinne weder Eigenschaften noch Methoden, vielmehr stellt sie Eigenschaften und Methoden den Objekten zur Verfügung, die wir durch diese Klasse erzeugen. Merke: Ein Objekt ist eine Instanz einer Klasse. Was das heißt? Nun, wenn wir am Ende dieses Artikels mit der Klasse fertig sind und diese einsetzen wollen, so müssen wir eine Instanz dieser Klasse bilden, ein Objekt. Wir binden die Klasse *PTableGrid* in die **.php*-Datei ein, über die eine Datenbankabfrage realisiert werden soll. Vorher haben wir den Quelltext der Klasse in der Datei *PTableGrid.php* abgespeichert:

```
include("pfad/zu/ihrer/Klasse/PTableGrid.php");
```

Wir können nun auf die Klasse zugreifen und ein Objekt, eine Instanz, dieser Klasse bilden:

```
$Tabelle=new PTableGrid;
```

Mit dem Schlüsselwort *new* haben wir die Klasse dazu veranlasst, der Variablen *$Tabelle* eine Instanz von *PTableGrid* zuzuweisen. Über diese Instanz können wir im Folgenden Eigenschaften und Methoden nutzen, die in der Klasse implementiert sind. Außerdem können wir bei einer solchen Instanziierung noch einen Konstruktor steuern.

Konstruktor?

Ein Konstruktor ist die Methode der Klasse, die, sobald ein Objekt der entsprechenden Klasse erzeugt wird, automatisch auf-

Listing 1

```
$gridQuery="SELECT Feld1,Feld2,Feld3,Feld4 FROM Tabelle";
$gridResult=mysql_query($gridQuery);
$gridCount=mysql_num_rows($gridResult);
$header_color="#ffffff";
$body_color="#ffffff";
$query_error="Es gab ein Datenbankproblem!";
$query_empty="Die Tabelle enthielt keine Datensätze!";




 if($gridResult)
  {
  echo "<table border='1'>";
  echo "<tr
bgcolor=".$header_color."><td>Feld1</td><td>Feld2</td><td>
                                Feld3</  td><td>Feld4</td></tr>";

if($grid_count=="0")
   {
   echo "<tr bgcolor=".$header_color."><td colspan='4'>
                                    ".$query_empty."</td></tr>";
   }
   else
    {
    while($arrayPosition=mysql_fetch_array($gridResult))
      {
      echo "<tr bgcolor=".$body_color.">";
      echo "<td>".$arrayPosition [0]."</td>";
      echo "<td>".$arrayPosition [1]."</td>";
      echo "<td>".$arrayPosition [2]."</td>";
      echo "<td>".$arrayPosition [3]."</td>";
      echo "</tr>";
      }
    }
  echo "</table>";
  }
  else
    {
    echo $query_error."<br>";
              echo mysql_error();
    }
```

gerufen bzw. ausgeführt wird. Es folgt ein Beispiel für den Einsatz eines Konstruktors:

```
class PTableGrid
  {

  function PTableGrid()  //Der Konstruktor der
                                        Klasse – muss
              den gleichen Namen wie die Klasse besitzen
    {
                    echo "Eine neue Instanz ist da!
                                        Hallo Welt!";
                }

  }
```

Sobald der Aufruf *$Tabelle=new PTableGrid* die Klasse *PTableGrid* dazu veranlasst, der Variablen *$Tabelle* eine Instanz von sich zur Verfügung zu stellen, erscheint die Ausgabe *„Eine neue Instanz ist da! Hallo Welt!"*. (Da dieser Konstruktor keine Parameter entgegennimmt, nennt man ihn auch *Standardkonstruktor*.)

Wir können eine Konstruktormethode auch so implementieren, dass sie Parameter entgegennimmt:

```
function PTableGrid($ausgabe)
    {
  echo $ausgabe;  //Der Konstruktor gibt den
                                übergebenen Parameter aus
                }
```

Damit diese Konstruktormethode dasselbe macht wie unser Standardkonstruktor, instanziieren wir ein neues Objekt mit

```
$Tabelle=new PTableGrid("Eine neue Instanz ist da!
                                        Hallo Welt!");
```

Woran merkt nun aber eine Klasse, dass sie einen Konstruktor besitzt? Das ist ganz einfach: Ein Konstruktor besitzt den gleichen Namen wie die Klasse – nur so wird eine Methode als „Objekt-Konstruktor" registriert. Implementiert der Programmierer in der Klasse keine Konstruktormethode, erstellt der Compiler zur Kompilierzeit automatisch einen Standardkonstruktor – mit leerem Rumpf. Warum aber der Einsatz von Konstruktoren sehr sinnvoll sein kann, werden wir später noch sehen.
Bis hierhin merken wir uns:

- Ein Konstruktor hat den gleichen Namen wie seine Klasse.
- An eine Konstruktormethode können Parameter übergeben werden.
- Ein vorhandener Konstruktor wird dann aufgerufen, wenn wir ein Objekt einer Klasse mit *new* instanziieren.

Variabel

Fangen wir langsam damit an, der Klasse Variablen zuzuweisen, die später die Eigenschaften eines Objekts der Klasse kennzeichnen. Welche Eigenschaften brauchen wir? Es wäre schön, wenn wir der Tabelle Farben zuweisen könnten: eine für den Hintergrund der Spaltenüberschriften und eine für den Rest der Tabelle. Des Weiteren wären freundliche Fehlermeldungen ganz nett, wenn etwas mit der Query oder den Datensätzen nicht klappen sollte.

Diese Eigenschaften werden wir in Variablen schreiben, und zwar in Instanzvariablen. Instanzvariablen sind Variablen, die nur in der Klasse bzw. dem initialisierten Objekt selbst sichtbar sind – sie können

Anzeige

nicht außerhalb der Klasse angesprochen werden, es sei denn, die Referenzierung findet über ein Objekt statt.

(OOP-Gurus werden hier laut protestieren: Mithilfe der Syntax *Klasse::Klassenmethode()* könnten wir ohne Initialisierung eines Objekts auf die Instanzvariablen zugreifen. In diesem Artikel wollen wir aber nur die Grundlagen besprechen und solche Sonderfälle außer Acht lassen.)
Die Variablen legen wir im Rumpf der Klasse ab:

```
class PTableGrid
        {

        var $header_color; //Farbe für den Hintergrund der Kopfzeile, in der die Spaltenüberschriften stehen werden
        var $body_color;  //Farbe für den Rest der Tabelle
        var $query_error; //Fehlermeldung, wenn z.B. das Query syntaktisch falsch war
        var $query_empty; //Fehlermeldung, wenn keine Datensätze vorhanden sind
    .
        .
        .
}
```

Beim Initialisieren eines Objekts der Klasse wird für diese Variablen Speicherplatz freigehalten. Außerdem werden sie „registriert“: Sobald eine dieser Instanzvariablen in einer Methode der Klasse (bzw. über das Objekt) verändert wird, wirkt sich das auf das gesamte Auftreten der jeweiligen Variablen in dem Objekt selbst aus.

Und was machen wir jetzt mit diesen Variablen? Die sind doch noch leer! Richtig. Deshalb basteln wir uns jetzt eigene Methoden, damit wir den Inhalt der Variablen setzen können. Methoden werden, wie der Konstruktor und wie die Instanzvariablen, im Rumpf der Klasse abgelegt. Listing 2 zeigt, wie die Methoden aussehen.

Was soll dieses *$this*? Und dieser Pfeil? Und überhaupt... Verzweifeln Sie nicht. Wenn wir mit Klassen arbeiten, ist die Schreibweise anders, als Sie es bislang von dem Einsatz normaler Funktionen oder globaler Variablen in PHP kennen.

Um einer Methode zu sagen, dass wir einen Wert einer Instanzvariable zuweisen wollen, müssen wir das Schlüsselwort *$this*, gefolgt von einem Pfeil -> dem Instanzvariablennamen voranstellen. Der Compiler weiß dann, dass wir uns auf eine Variable dieser Instanz beziehen. *$this* bezieht sich also immer auf die aktuelle Instanz der Klasse. Umgangssprachlich würden wir vielleicht statt *$this->x=$y;* schreiben:

Listing 2

```
class PTableGrid
        {
            .
            .
            .
            .

            function setHeaderColor($color) //Instanzmethode zum Setzen der Überschrift-Hintergrundfarben
                    { $this->header_color=$color; }

            function setBodyColor($color) //Instanzmethode zum Setzen der Datensatz-Hintergrundfarben
                    { $this->body_color=$color; }

            function setQueryError($msg) //Instanzmethode zum Setzen der Fehlermeldung, wenn z.B. das Query syntaktisch falsch war
                    { $this->query_error=$msg; }

            function setQueryEmpty($msg) //Instanzmethode zum Setzen der Fehlermeldung, wenn keine Datensätze gefunden wurden
                    { $this->query_empty=$msg; }

            }
```

Weise der Instanzvariablen x des Objektes den Wert aus $y zu

Das Gleiche gilt für Instanzmethoden. Sollen diese innerhalb einer Klasse bzw. eines Objekts aufgerufen werden (z.B. im Konstruktor), so müssen wir auch hier *$this* voranstellen.

Dass wir *$* nicht vor der Variablen *x* gesetzt haben, ist schon richtig. Wenn wir über *$this* Methoden oder Variablen referenzieren, ist das *$* vor dem Variablen- oder Methodennamen nicht erlaubt.

```
$this->$x=$y //Fehler!
$this->x=$y //Richtig!
```

Am Rande: Denken sie nicht um zu viele Ecken! Die Bedeutung von *$this* ist leichter zu verstehen, als es zunächst vielleicht erscheinen mag. Wenn sie eine Klasse basteln, dann werden sie im Rumpf Instanzvariablen und Instanzmethoden anlegen. Wenn sie jetzt in einer dieser Instanzmethoden eine Variable der Klasse ansprechen wollen, stellen sie dem Variablennamen einfach ein *$this* voran. Der Compiler und Sie werden dann wissen, dass sich die Anweisung auf eine Variable dieser Klasse bezieht.

Wir haben jetzt vier Instanzvariablen und vier Instanzmethoden in der Klasse definiert. Den oben genannten Konstruktor vergessen wir wieder, er wurde nur zu Anschauungszwecken verwendet und wird nachher neu implementiert. Was fangen wir jetzt damit an? O.k., kehren wir zurück zu der Stelle, an der wir ein Objekt der Klasse instanziiert haben:

```
$Tabelle=new PTableQuery;  //Objekt $Tabelle der Klasse PTableQuery instanziieren
```

Und wie wende ich eine Methode dieses Objektes an? Ganz leicht. Da *$Tabelle* nun ein Objekt der Klasse *PTableGrid* ist, können wir auch die Methoden dieses Objekts anwenden, die wir im Rumpf der Klasse definiert haben:

```
$Tabelle->setHeaderColor(´´#000000´´);
$Tabelle->setBodyColor(´´#000000´´);
$Tabelle->setQueryError(´´Es gab einen Fehler bei der Ausführung Ihres Querys.´´);
$Tabelle->setQueryEmpty(´´Die Tabelle enthielt keine Datensätze´´);
```

Anzeige

Was haben wir jetzt genau gemacht? Wir haben das Objekt *$Tabelle* dazu veranlasst, einige „seiner“ Methoden auszuführen bzw. die Methoden auszuführen, die die Klasse *PTableGrid* bereitstellt. Anschaulicher wird es, wenn wir noch einmal zu unserem Auto zurückkehren. Überlegen Sie sich, wie sie Quellcode verfassen würden, der ein Objekt *$Auto* der Klasse *Fortbewegungsmittel* erzeugt. Sie möchten das Objekt auf 50 km/h beschleunigen, um direkt danach wieder auf 0 km/h abzubremsen. Außerdem dürfen Sie dem Auto eine Farbe zuweisen (siehe Tabelle 1).

Listing 3

```
class PTableGrid
   {
                        .
                        .
                        .
                        .

    function PTableGrid()  //Der Konstruktor der Klasse – muss den gleichen Namen wie die Klasse selber besitzen
       {
                                    $this->header_color="#ffffff";
                          $this->body_color="#ffffff";
                         $this->query_error="Es ist ein Fehler aufgetreten!";
                           $this->query_empty="Die Tabelle war leer.";
       }

    }
```

Listing 4

```
class PTableGrid
   {
                        .
                        .
                        .
                  var $headers; //Array, in dem die Teil-Strings gespeichert werden
                        .
                        .
                        .


    function setHeaders($ueberschrift)  //Methode bekommt einen String, die Spaltenüberschriften, übergeben
       {
        $this->headers=explode(";",$ueberschrift); //Der Parameter wird nach ; durchsucht,
                                                   //gesplittet und in die Instanzvariable $headers geschrieben
       }

   }
```

```
$Auto=new Fortbewegungsmittel;                  // neues Objekt $Auto der Klasse Fortbewegungsmittel
                                                   instanziieren
$Auto->farbe("gruen_metallic");                 // der Eigenschaft 'Farbe' des Autos weisen wir über
                                                   eine Methode 'gruen_metallic' zu
$Auto->erhoehe_Geschwindigkeit("50km/h");       // die Geschwindigkeit wird auf 50 km/h erhoeht
$Auto->bremse("0km/h");                         // das Auto wird auf 0 km/h abgebremst (Stillstand)
```

Tabelle 1: Objekte und Klassen

Konstruktor!

Je umfangreicher eine Klasse wird, soll heißen: je mehr Methoden und Eigenschaften eine Klasse einem Objekt zur Verfügung stellt, desto größer ist die Gefahr, dass man den einen oder anderen Methodenaufruf vergisst. Der Programmierer will deswegen Sorge tragen, dass gewisse Eigenschaften bereits beim Erzeugen eines Objektes auf Standardwerte gesetzt werden. Hierzu bedienen wir uns einfach eines Konstruktors. Wir wissen jetzt, dass ein Konstruktor – wenn vorhanden – immer dann aufgerufen wird, wenn ein Objekt einer Klasse erzeugt wird. Und wenn es etwas gibt, was ein Objekt von vornherein wissen bzw. ausführen soll, dann schreiben wir es in den Konstruktor. Damit können wir unser Objekt in einen Anfangszustand versetzen (siehe Listing 3).

Beim Erzeugen des Objekts werden die genannten Klassenvariablen also automatisch mit diesen Werten gesetzt. Nichtsdestotrotz behalten nachfolgende Methodenaufrufe, die die Werte dieser Variablen ändern, immer ihre Gültigkeit.

Programmierarbeit

Sie haben nun einige der wichtigsten Werkzeuge der OOP-Architektur kennengelernt: Methoden, Instanzvariablen, Konstruktoren sowie die Schlüsselworte *$this* und *new*. Aber unsere Klasse ist noch nicht fertig. Wir können zwar schon einige Eigenschaften setzen, aber es gibt noch keine Möglichgkeit, eine Query an ein Objekt dieser Klasse zu übergeben; außerdem fehlt uns noch eine Methode, mit der wir den Spalten Überschriften zuweisen können.

Fangen wir mit den Spaltenüberschriften an. In dem anfangs erwähnten Quelltext haben wir vier Felder, die wir aus der Tabelle auslesen möchten. Also brauchen wir auch vier Spalten und dementsprechend vier Überschriften. Doch wie übergeben wir diese Überschriften der Methode, die wir uns basteln wollen? Sollen wir eine Methode schreiben, die vier Parameter aufnehmen kann? Nein! Das würde unserer Idealvorstellung einer flexiblen Klasse widersprechen.... denn was wäre, wenn wir demnächst diese Klasse in einem Projekt verwenden

Listing 5

```
class PTableGrid
   {
                        .
                        .
                        .
                        .
                        .

            function showGrid($query)
                        {

                              $result_query=mysql_query($query);   //
                        $result_count=mysql_num_rows($result_query);  // Methodenvariablen
                              $field_count=mysql_num_fields($result_query); //

                              if(!$result_query)                        //Überprüfung, ob ein Fehler vorliegt
                                { echo $this->query_error."<br>".mysql_error(); return;}

                               if($result_count==0)              //existieren schon Datensätze in der Tabelle?
                                  { echo $this->query_empty; return;}

                        if($result_query && $result_count>0)  //Tabelle wird ausgelesen
                               {
                                    echo "<table border='1'>";
                                    echo "<tr>";

                                if(isset($this->headers)) //wurden Werte an $this->headers übergeben?
                                  {              //wenn ja, lese die Werte aus und schreibe sie in die erste Tabellenzeile
                                  for($i=0;$i<count($this->headers);$i++) //Spaltenüberschriften werden
                                                                                  gesetzt (selbstdefiniert)
                                        { echo "<td bgcolor=".$this->header_color.">".$this->headers[$i]."</td>";}
                                  }
                                  else  //ist das nicht der Fall, lesen wir die Feldnamen direkt aus der DB-Tabelle und
                                                                                  schreiben
                                     { //sie in die erste Zeile
                                     for($i=0;$i<$field_count;$i++)  //Spaltenüberschriften werden gesetzt
                                                                                  (aus der DB-Tabelle)
                                     { echo "<td bgcolor=".$this->header_color.">".mysql_field_name($result_
                                                                                  query,$i)."</td>";}
                                      }

                                     echo "</tr>";

                                     while($result_array=mysql_fetch_array($result_query))
                                                {
                                                      echo "<tr bgcolor=".$this->body_color. ">";

                                     for($i=0;$i<$field_count;$i++)    //Datensätze werden gelesen und ausgegeben
                                                      { echo "<td>".$result_array[$i]."</td>"; }

                                                  echo "</tr>";
                                      }
                  echo "</table>";
                  }

                        }

}
```

Anzeige

möchten, bei dem wir sechs, sieben oder noch mehr Felder aus der Tabelle auslesen möchten? Wir müssten jedes Mal hingehen und eine zusätzliche Methode implementieren, an die die entsprechende Anzahl von Parametern übergeben werden kann. Der erfahrende Programmierer wird hier das Argument einbringen, dass man eine weitere Klasse implementieren kann, die dann über die „Eltern-Klasse“ mittels *extends* beliebig erweitert wird. Und obwohl das Konzept der OOP – bis jetzt – in PHP eher wenig umfangreich implementiert ist, existiert diese Möglichkeit. Diese Lösung ist in bestimmten Fällen sicherlich von Vorteil – für unsere Zwecke aber viel zu kompliziert. Und entgegen Ihrer Befürchtungen kommt nur wenig Programmierarbeit auf uns zu. Wir werden uns nämlich eine Methode basteln, der wir nur einen Parameter übergeben, und zwar einen String. Der Methodenkopf sieht folgendermaßen aus:

```
function setHeaders($ueberschrift)
```

Bevor wir uns dem Methodenrumpf widmen, kommt hier der Aufruf der Methode – vielleicht wird Ihnen jetzt schon klar, was wir vorhaben:

```
$Tabelle->setHeaders(“Feld1;Feld2;Feld3;Feld4“);
```

Wir werden *explode()* benutzen, um den String nach ; zu durchsuchen, zu splitten und die so erhaltenen Teil-Strings in ein Array zu schreiben.

Voilà! Wir haben unsere Spalten-Überschriften. Zunächst fügen wir unserem Klassenrumpf eine weitere Variable hinzu, die als Array die Teil-Strings speichern soll (siehe Listing 4).

Zu guter Letzt benötigen wir noch die Methode, an die wir unsere MySQL-Query übergeben werden. Diese Methode sieht wie in Listing 5 dargestellt aus.

Der Methodenkopf ist einleuchtend: Wir übergeben der Methode eine Query der Art *SELECT x,y,z FROM Tabelle*. Im Rumpf haben wir dann drei Variablen angelegt, die nur in der Methode sichtbar sind und außerhalb dieser keine Gültigkeit besitzen. *$result_query* ist eine boolesche Variable und speichert das Ergebnis von *mysql_query($query)*. *$result_count* speichert die Anzahl der Datensätze, die durch unsere Query zurückgegeben werden, und *$field_count* speichert die Anzahl der Felder, die wir abfragen.

Als Nächstes fangen wir eventuelle Fehler ab. Gibt *$result_query false* zurück, bricht die Methode ab, nachdem die vorher von uns definierte Fehlermeldung *$this->query_error* ausgespuckt wurde, samt ausführlicher Fehlerbeschreibung durch *mysql_error()*.

Ist die Query gültig, wird als Nächstes überprüft, welchen Wert *$result_count* hat. Ist die Anzahl der Ergebnis-Datensätze *0*, bricht die Methode hier ab und meldet uns durch *$this->query_empty*, dass durch die Suchabfrage keine Ergebnisse gefunden wurden.
Treffen beide Fälle nicht zu, baut die Methode unsere Tabelle auf. Dazu holt sie sich aus dem Array *$this->headers* alle Überschriften der Spalten, die wir dem Objekt vorher durch *setHeaders($ueberschrift)* übergeben haben. Wenn wir dafür zu faul waren, liest die Methode automatisch die entsprechenden Feldnamen aus der MySQL-Tabelle und setzt diese als Überschriften. Danach liest die Methode alle Datensätze aus dem Result-Set und gibt sie formatiert in der Tabelle aus.

Und jetzt vergleichen sie einmal die Beispielimplementierung am Anfang des Artikels mit dem folgenden Quelltext. Wir schaffen es in wenigen Zeilen, die Tabelle abzufragen und das Ergebnis HTML-formatiert auszugeben (siehe Listing 6).

Fazit

Der Umgang mit Klassen ist, ein wenig abstraktes Denkvermögen vorausgesetzt, relativ einfach. Wenn Sie einmal das Grundkonzept verstanden und schon die eine oder andere Klasse programmiert haben, fällt auch der Umgang mit den OOP-Konzepten anderer Sprachen nicht schwer.

Schauen sie sich ihre Projektliste einmal an und machen Sie sich eine Strichliste – es gibt immer wieder Aufgaben, die die Standard-Funktionsbibliotheken von PHP nicht ohne weiteres bewerkstelligen können. Und es gibt immer einen Weg, für diese häufig geforderten Aufgaben flexible Klassen zu erstellen. Vergessen Sie nicht: Es braucht Zeit, um eine Klasse zu planen und diese zu programmieren. Aber die Mühe zahlt sich spätestens dann aus, wenn sie demnächst schnell und effizient Ihre eigenen Komponenten in diverse Projekte einbringen können.

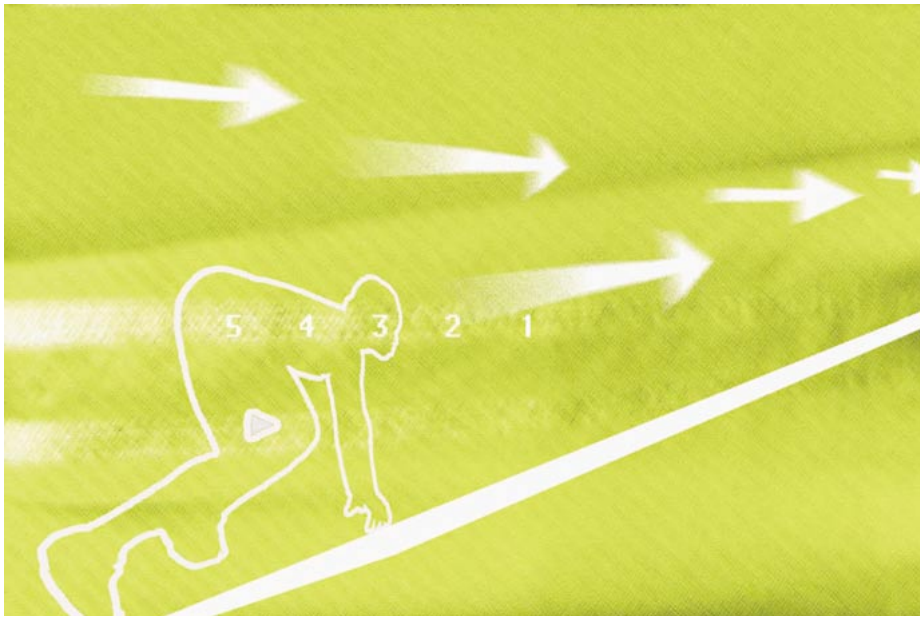
Da ein Magazin-Artikel nur die Grundlagen einer solch komplexen Materie abdecken kann, seien interessierte Leser auf das Buch *Objektorientierung in 7 Tagen* von Prof. Dr. Heide Balzert verwiesen, erschienen im Spektrum Verlag.

Den vollständigen Quelltext der Klasse und eine Beispielimplementierung können Sie sich auf der Homepage www.siteartwork.de/phpmagazin des Autors herunterladen.

Listing 6

```
include(“pfad/zu/der/Klasse/PTableGrid.php“)

$Tabelle=new PTableGrid; //$Tabelle wird ein Objekt der Klasse PTableGrid
$Tabelle->setHeaders(“Feld 1;Feld 2;Feld3;Feld4“); //Optional – wenn nicht angegeben, nimmt die Klasse die Feldnamen der DB-Tabelle
$Tabelle->setQueryError(“Beim Auslesen der Tabelle ist ein Fehler aufgetreten!“); //Optional – wird auch durch den Konstruktor initialisiert
$Tabelle->setQueryEmpty(“Die Tabelle enthielt keine Datensätze!“); //Optional – wird auch durch den Konstruktor initialisiert
$Tabelle->setHeaderColor(“red“); //Optional – wird auch durch den Konstruktor initialisiert
$Tabelle->setBodyColor(“white“); //Optional – wird auch durch den Konstruktor initialisiert
$Tabelle->showGrid(“SELECT feld1,feld2,feld3,feld4 FROM tabelle“); //Der Instanzmethode showGrid() wird das Query übergeben
```



Hoch damit! Aber sicher ...

von Hartmut Holzgraefe

Sichere Datei-Uploads mit PHP implementieren

HTML-Formulare zum Datei-Upload lassen sich mit PHP in nur wenigen Zeilen einfach implementieren, ohne dass man sich dabei mit den Einzelheiten der zugrunde liegenden Mechanismen auseinander setzen muss. Dieser Komfort täuscht allerdings darüber hinweg, dass für robuste und sichere Anwendungen trotz allem bei Uploads noch ein paar wichtige Dinge zu beachten sind.

Ein einfaches PHP-Skript, das den Upload einer Datei ermöglicht und diese sofort an den Client zurück überträgt kann z.B. wie in Listing 1 gezeigt aussehen.

Zugegeben, kein sehr sinnvolles Beispiel, aber es zeigt in übersichtlicher Form recht deutlich alle wesentlichen Komponenten eines Upload-Skripts. Im HTML-Teil sind dies zunächst einmal die *enctype*- und *method*-Attribute des <*form*>-Tags. Ein erfolgreicher Upload ist generell nur mit der *POST*-Methode in Kombination mit dem Encoding-Typ *multipart/form-data* möglich. Ein Eingabeelement für den Namen bzw. Pfad der zu übertragenden Datei wird mit Hilfe eines *input*-Elementes vom Typ *file* erstellt.

PHP wertet bei Empfang eines Uploads die übertragenen Daten selbstständig aus und erzeugt zunächst einmal auf dem Server eine temporäre Datei, in der der empfangene Dateiinhalt abgelegt wird. Weiterhin werden vier Variablen angelegt, die alle vorhandenen Informationen über die empfangene Datei enthalten. Die Namen dieser Variablen ergeben sich dabei aus dem *name*-Attribut des *file*-Input-Elements. Innerhalb des speziellen Arrays $HTTP_POST_FILES wird für jedes File-Input ein Element mit dem Namen des Inputs angelegt. Dieses ist selbst wieder ein Array mit den Elementen *‚name‘*, *‚size‘*, *‚type‘* und *‚tmp_name‘*, die den ursprünglichen Dateinamen, die Größe in Bytes, den Mime-Type und den Namen der Server-Datei, in der die übertragenen Daten vorübergehend abgelegt sind, enthalten. Je nach Server-Konfiguration sind diese Daten auch direkt in globalen Variablen abgelegt, von deren Benutzung aber aus Sicherheitsgründen abgeraten werden muss.

Serverseitige Risiken

File-Uploads unterscheiden sich von „normalen“ Formularen vor allem dadurch, dass ohne großen Aufwand zunächst einmal beinahe beliebig große Datenmengen übertragen werden können. Hierdurch können serverseitig die Ressourcen Bandbreite, Plattenplatz und Hauptspeicher erheblich belastet werden. Es ist daher im Allgemeinen sinnvoll, die übertragbare Datenmenge zu beschränken. Clientseitig lässt sich dies durch ein zusätzliches Formularfeld erreichen:

```
<input type='hidden' name='MAX_FILE_SIZE' value=
                                    'größe_in_bytes'>
```

Diese Angabe hat allerdings nur Vorschlagscharakter, es gibt keine Garantie darauf, dass der jeweilige Client diese auch wirklich beachtet. Bei modernen Browsern kann man zwar davon ausgehen, bei anderen Clients aber nicht unbedingt. Daher müssen auch auf Serverseite entsprechende Maßnahmen getroffen werden, die vor einer Überlastung schützen. Hierzu dient der Konfigurationspa-

rameter *upload_max_filesize*, der die maximale Größe eines Uploads in Bytes angibt. Hierdurch kann zwar nicht die Übertragung einer größeren Datenmenge durch den Client verhindert werden, zumindest jedoch ein übermäßiger Verbrauch vor Platten- und Hauptspeicherplatz. Im ungünstigsten Fall, d.h. wenn ein Server gezielt durch Upload-Requests attackiert wird und alle verfügbaren Server-Prozesse bzw. -Threads diese Requests verarbeiten, werden Dateisystem und Hauptspeicher mit *upload_max_filesize* multipliziert mit der maximalen Prozess- bzw. Thread-Anzahl Bytes belastet. Solange der verfügbare Plattenplatz für das temporäre Upload-Verzeichnis sowie die Summe aus Hauptspeicher und Swapspace über diesem Wert liegen, wird der Server durch solch eine Attacke zwar eventuell deutlich belastet, der ordnungsgemäße Betrieb bleibt aber sichergestellt.

Zu beachten ist dabei, dass alle bisherigen PHP-Releases die übertragenen Daten zur Auswertung zunächst einmal vollständig im Hauptspeicher ablegen, sodass die maximale Größe eines Uploads eventuell auch durch den Konfigurationsparameter *memory_limit* begrenzt wird. Beginnend mit PHP 4.2 werden dagegen die empfangenen Daten direkt beim Empfang in temporäre Dateien geschrieben, sodass die bisherige (wenn auch nur kurzfristige) Speicherbelastung durch Uploads entfällt.

Bezüglich der Bereitstellung von Plattenplatz gilt, dass die von PHP aus den heraufgeladenen Daten erstellten Dateien zunächst einmal nur temporär sind, d.h. sie werden am Skriptende wieder gelöscht. Dieser Mechanismus schützt vor der Verschwendung von Plattenplatz durch nicht ordentlich programmierte Skripte sowie weitgehend vor Denial-of-Service Attacken durch das Senden von Upload-Daten an Skripte, die diese gar nicht erwarten und dementsprechend auch nicht verarbeiten. Sollen übertragene Daten über die Laufzeit des Skriptes hinaus auf dem Server erhalten bleiben, so müssen diese an anderer Stelle abgelegt werden. Im einfachsten Fall reicht es dabei aus, die temporären Dateien an einen anderen Ort im Dateisystem zu verschieben oder zu kopieren.

Da die Informationen über die empfangenen Dateien in globalen Variablen abgelegt werden, können diese durch vor der Auswertung ausgeführten PHP-Code oder durch geschickte Manipulation der übertragenen Formulardaten verfälscht worden sein. Zum Beispiel könnte unser Beispielskript dazu missbraucht werden, beliebige Dateien vom Server an den Client zu übertragen. Diese Form des Missbrauchs lässt sich mit den ab PHP 4.0.2 verfügbaren Funktionen *is_uploaded_file()* und *move_uploaded_file()* verhindern, die jeweils prüfen, ob die angegebene Datei tatsächlich durch einen Upload erzeugt wurde.

Werden alle bisher beschriebenen Maßnahmen beachtet, so sind alle Gefahren, die für den Betrieb eines Webservers aus Uploads entstehen können, weitgehend abgeblockt. Unser entsprechend erweitertes Beispielscript sehen Sie in Listing 2.

Gefahr für die Clients

Potenzielle Gefährdungen ergeben sich allerdings nicht nur durch Betriebsstörungen des Servers. Auch bei korrekter Verarbeitung der empfangenen Daten und Beachtung aller bisherigen Sicherheitshinweise ist noch nicht sichergestellt, dass nicht von den übertragenen Daten selbst Gefahren ausgehen.

Sehr einfache, aber sicherheitsrelevante Manipulationen sind dadurch möglich, dass bei der Übertragung der Daten ein falscher Dateityp angegeben wird. So könnten beispielsweise in einer Bildergalerie statt Bildern auch ausführbare Programme angelegt werden, die so ungewollt an Clients übertragen und im ungünstigsten Fall auch ausgeführt werden.

Serverseitig kann der Dateityp einer Upload-Datei zum Beispiel auf Unix-Sys-

Listing 1

```
<?php
 if($HTTP_SERVER_VARS['REQUEST_METHOD']
                                    =='POST') {
  header("Content-type: $HTTP_POST_FILES
                                    [userfile][type];
filename=$HTTP_POST_FILES[userfile][name]");
  header("Content-length: $HTTP_POST_FILES
                                    [userfile][size]");
  readfile($HTTP_POST_FILES[userfile][tmp_name]);
  exit;
 }
?>
<html>
 <head><title>Upload-Beispiel</title></head>
 <body>
  <form method='POST enctype='multipart/form-data'>
   Datei:
   <input type='file' name='userfile'>
   <input type='submit'>
  </form>
 </body>
</html>
```

Listing 2

```
<?php
 $max_size=10000;

if($HTTP_SERVER_VARS['REQUEST_METHOD']=='POST')
{
  if($HTTP_POST_FILES['userfile']['size']==0) {
   $error_msg='Datei zu groß oder leer';
  } else
  if($HTTP_POST_FILES['userfile']['size']>$max_size) {
   $error_msg='Datei zu groß oder leer';
  } else
  if(!is_uploaded_file(
$HTTP_POST_FILES['userfile']['tmp_name'])) {
   $error_msg='Dateiname wurde manipuliert';
  } else {
  header("Content-type: $HTTP_POST_FILES
                                    ['userfile']['type'];
filename=$HTTP_POST_FILES['userfile']['name']");
  header("Content-length: $HTTP_POST_FILES
                                    ['userfile']['size']");
  readfile($HTTP_POST_FILES['userfile']
                                    ['tmp_name']);
  exit;
  }
 }
?>
<html>
 <head><title>Upload-Beispiel</title></head>
 <body>
  <?php echo $error_msg; ?>
  <form method='POST enctype='multipart/form-data'>
   <input type='hidden' name='MAX_FILE_SIZE'
                    value='<?php echo $max_size;?>'>
   Datei:
   <input type='file' name='userfile'>
   <input type='submit'>
  </form>
 </body>
</html>
```

temen mit Hilfe des *file*-Kommandos ermittelt werden. *file* nutzt dazu eine Datenbasis, die typische Byte-Signaturen aller gängigen Datenformate enthält. Die GNU Version des *file*-Kommandos bietet dabei mit der Aufrufoption *-bi* die direkte Ausgabe des ermittelten Mime-Types, andere Versionen geben dagegen eventuell nur Bezeichnungen ohne spezielle Namenskonventionen zurück.

Das folgende Beispiel akzeptiert nur GIF-, JPEG- und PNG-Grafiken. Zusätzlich ermittelt es bei JPEG-Dateien, ob diese im Progressive-Format kodiert sind. Der MS IE hat die unangenehme Eigenschaft, bei JPEG-Dateien grundsätzlich den Mime-Type *image/pjpeg* anzugeben, selbst wenn es sich tatsächlich um ein normales, nicht progressiv kodiertes Bild handelt:

```
<?php
...
$mimetype=exec('file –bi '.escapeshellcmd($filename));
switch(trim($mimetype)) {
 case 'image/gif':
 case 'image/jpeg':
 case 'image/pjpeg':
  /* ... ok ...*/
  break;
 default:
  /* ... fehler ...*/
  break;
}
...
?>
```

Eine Integration der Funktionalität des *file*-Kommandos in eine PHP Extension ist zurzeit in Arbeit, aber leider noch nicht öffentlich verfügbar.

Viren und anderes Ungeziefer

Da übertragene Dateien nicht nur gesammelt und gespeichert, sondern auch weiterverarbeitet oder zur Anzeige oder Weiterverarbeitung an andere Clients übertragen werden, besteht noch eine weitere Gefahr: die Übertragung von mit Viren verseuchten Dateien. Werden bei Uploads Dateien akzeptiert, die direkt ausführbar sind oder ausführbare Komponenten wie z.B. VBA-Makros enthalten, so sollten diese möglichst bereits beim Upload auf eventuell enthaltene Viren überprüft werden.

Unter Unix kann man auch hier wieder ein externes Kommando um Hilfe bitten. Viele gängige Virenscanner sind auch als Unix-Versionen verfügbar und können ähnlich wie das *file*-Kommando zur Analyse einzelner Dateien eingesetzt werden. So bietet z.B. McAfee das Unix-Kommando *uvscan*, das bei einer Virus-infizierten Datei den exit-Status 13 zurückgibt:

```
<?php
 ...
 exec('uvscan --secure —file '.escshellcmd($filename)
   , $result
   , $return_code
   );
 switch($return_code) {
  case 0: /* ... ok ... */; break;
  case 13: /* ... virus ... */; break;
  default: /* ... fehler ... */; break;
 }
 ...
?>
```

Demo-Versionen des McAfee Scanners sind unter www.nai.com/naicommon/buy-try/try/products-evals.asp verfügbar.

Fazit

File-Uploads sind somit mit PHP nicht nur einfach, sondern mit ein klein wenig Zusatzaufwand auch sicher und zuverlässig implementierbar. Dieser zusätzliche Aufwand ist in Anbetracht der möglichen Gefahren insbesondere bei öffentlichen Angeboten, aber durchaus auch in der relativ abgesicherten Umgebung eines Intranet durchaus gerechtfertigt, insbesondere, wenn man die potenziellen Schäden durch die Weiterverbreitung ungewollter Inhalte in Zusammenhang mit Viren und Trojanern bedenkt.

Anzeige

Workshop PHP

Tobias Hauser, Andreas Kordwig, Christian Wenz

Wer in Buchhandlungen vor den Regalen mit den EDV-Büchern steht, wird feststellen, dass der Markt für Werke, die zum Selbststudium verleiten wollen, offensichtlich boomt. Egal, ob Kenntnisse über Betriebssysteme, Konfigurationen von Server-Diensten, Anwendungsprogramme oder Programmier- und Scriptsprachen gewonnen werden wollen, der Markt deckt beinahe alles ab und das oftmals von mehreren Verlagen. Der Verbraucher hat die Qual der Wahl und der Verlag das Problem der Abgrenzung zu den Produkten der Mitbewerber. Der Verlag Addison-Wesley bietet daher u.a. zusammen mit den Büchern seiner „Workshop"-Reihe einen abschließenden Online-Test an, bei dem der Leser die gewonnenen Kenntnisse aus dem Buch unter Beweis stellen muss und bei Erfolg ein schriftliches Zertifikat erhält. Damit Missbrauch ausgeschlossen wird, findet der Leser auf der dem Buch beiliegenden CD-ROM eine persönliche Identifikationsnummer, mit der er sich für den Online-Test anmelden muss. Maximal drei Prüfungsanläufe sind erlaubt. Ein solcher Test kann auch für die Skriptsprache PHP abgelegt werden. Voraussetzung dazu ist die Anschaffung des Buches „Workshop PHP".

Um sich PHP aneignen zu können, bedarf es natürlich einer entsprechenden Entwicklungsumgebung. Wie diese für die verschiedenen Webserver eingerichtet wird, erfährt der Leser im ersten Kapitel. Leider wird die Installation der Webserver nicht erklärt, sondern nur die Bezugsquelle der Programme im Internet angegeben. Auch auf der Buch-CD-ROM findet sich keine entsprechende Software. Für den unerfahreneren Leser bietet sich hier sofort Raum für weitere Selbststudiumsaktivitäten. Weiter geht's im Buch mit Grundlagen der Skriptsprache. Hier werden beispielsweise Vergleichsoperatoren, Bedingungen und Arrays vorgestellt. Es folgt die Einbindung von PHP in Formulare, Session-Management, Dateien-Handling, Kommunikation mit anderen Servern (Mail, FTP) und Datenbankanbindung (MySQL, ODBC-Schnittstellen, MS SQL Server). Aber auch bei dem Thema „Datenbanken" bleibt das Buch unerfreulich konsequent, denn die Installation von z.B. MySQL wird nicht behandelt. Im neunten und letzten Kapitel werden Erweiterungen beschrieben. Darunter verstehen die Autoren unter anderem das Erstellen von Grafiken, das Generieren von PDF-Dateien und das Zusammenspiel von PHP und XML. Die Konfigurationsbeschreibung für die XML-Bibliothekeneinbindung beschränkt sich für Linux auf die Bemerkung, dass PHP mit der Option *– with –xml* konfiguriert werden muss. Alles klar, oder?

Die Kapitel schließen mit Übungsaufgaben ab. Im anschließenden Lösungsteil findet dann der Leser neben der Antwort bzw. dem Lösungsskript einige zusätzliche Erläuterungen. Einer sofortigen Erfolgskontrolle steht somit nichts mehr im Weg.

Das Buch richtet sich laut Einbandtext an Fortgeschrittene und Profis. Das würde das Fehlen der Installationsbeschreibungen für die benötigte Zusatzsoftware wie Webserver und Datenbank erklären, aber nicht unbedingt entschuldigen. Die Autoren verweisen zwar darauf, dass sie den Buchumfang nicht mit Dokumentationen unnötig aufblähen wollen und durch die Internet-Verweise kämpfen sie auch nicht mit veralteten Buchinhalten. Trotzdem wäre eine kurze Installationsbeschreibung sicherlich für viele Leser hilfreich. In anderen Büchern mit gleichem Thema ist das möglich. Es sind also fortgeschrittene Kenntnisse für die Umgebungseinrichtung erforderlich, aber nicht für die meisten Kapitelinhalte, die dafür zu einfach gestrickt sind und Themen wie beispielsweise Objektorientierung vermissen lassen.

Wer sich Hoffnungen gemacht hat, dass auf der Buch-CD-ROM die Zusatzsoftware und die Installationsdokumentation zu finden wären, wird enttäuscht. Die einzigen CD-Inhalte, die einen Bezug zum Buch haben, sind die PHP-Installationsprogramme inklusive Dokumentation, die Skripte aus den einzelnen Kapiteln und zwei Editoren. Dies umfasst insgesamt circa 33 MB. Bei den übrigen rund 250 MB(!) der CD-ROM hat es der Verlag sich nicht nehmen lassen, Werbung für die eigenen Produkte und über 130 MB Shareware-Programme aufzuspielen, die in keinem Zusammenhang mit dem Buchinhalt stehen. Was zum Beispiel Anrufbeantworter-Software und Audiodateien-Konverter mit PHP zu tun haben, weiß wohl nur der Verleger. Kurz gesagt, die CD-ROM enttäuscht auf ganzer Linie und passt sich damit dem Rest des Buches an.

Wer sich also in PHP professionell einarbeiten will, findet sicherlich bessere Bücher im Handel. Einsteiger stehen bei diesem Buch ohnehin schon im ersten Kapitel ohne Entwicklungsumgebung voll im Regen.

Markus Hasenbein

Tobias Hauser, Andreas Kordwig, Christian Wenz

Workshop PHP

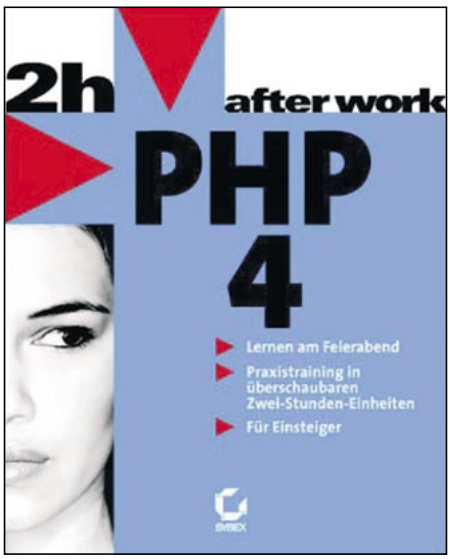
Addison-Wesley, 2001

360 Seiten, inkl. CD-ROM, € 35,74

ISBN 3-8273-1816-5

PHP – After Work

Mark Kronsbein, Thomas Weinert, Peter Petermann, Clemens Gutweiler



Wie zahlreiche Umfragen unter IT-Professionals und Hobby-ITlern immer wieder aufzeigen, nutzen viele ihre Freizeit, um sich über die neuesten Entwicklungen und Trends im IT-Bereich zu informieren oder sich in neuen Technologien und Programmiersprachen weiterzubilden. Öffentliche Bildungseinrichtungen wie beispielsweise Volkshochschulen folgen diesem Trend und bieten mittlerweile die unterschiedlichsten Kurse für alle möglichen Personengruppen an. Wem dies zu zeitaufwändig, zu unflexibel oder zu teuer ist oder wer vielleicht nach vielen einsamen Nächten am heimischen PC einfach die Nähe real existierender Mitmenschen scheut, kann sich auch mittels Fachliteratur im Selbststudium die gewünschten Fähigkeiten und Kenntnisse aneignen. Auf diese Personengruppe zielt eine neue Reihe von Büchern aus dem Düsseldorfer Sybex-Verlag. Der Namen der Reihe, „after work“, soll dies bereits andeuten. Das Niveau der Bücher orientiert sich an Einsteigern in das jeweilige Thema, setzt aber schon PC-Kenntnisse voraus. Ein Werk dieser Serie widmet sich der immer populärer werdenden Skriptsprache PHP. Die dem Buch beigefügte CD-ROM enthält die benötigte Software (Webserver, PHP-Modul, Datenbank MySQL, Editor) und alle Listings, die im Buch beschrieben werden, sodass dem Leser das Abtippen erspart bleibt. Die einzelnen Kapitel sollen in ihrem Umfang so beschaffen sein, dass sie in circa zwei Stunden durchzuarbeiten und am eigenen Rechner nachzuvollziehen sind.

Nach einer Einleitung, in der der Leser allgemeine Informationen über PHP und das Buch erhält, folgt im ersten Kapitel die Installation, die sowohl für den Apache Webserver unter Linux und Windows als auch für den MS Internet Information Server und den etwas exotisch anmutenden Xitami Webserver erklärt wird. Erfreulicherweise gehen die Autoren auch auf die Optionen für das Kompilieren von PHP unter Linux ein. Leider fehlt aber auf der CD-ROM die PHP4Win-Distribution, deren Installation im Buch beschrieben wird und die angeblich auch auf der CD-ROM zu finden sein soll. Die Kapitel zwei und drei führen den Leser dann durch die Grundlagen von PHP. Dies geschieht immer kurz anhand eines Beispiels. In Kapitel vier werden die Möglichkeiten einer sauberen Strukturierung des Codes unter anderem mittels der *include()*-Funktion erläutert. Kapitel fünf geht auf die Möglichkeiten der objektorientierten Programmierung von PHP ein und auch auf die bisher noch in diesem Zusammenhang vorhandenen Schwächen.

Die folgenden beiden Kapitel vertiefen dann anhand zweier praktischer Beispiele das bisher Gelernte und die Zusammenarbeit zwischen PHP und Formularen. Der Umgang mit der Datenbank MySQL ist Inhalt von Kapitel acht. Die folgenden Abschnitte behandeln dann praktische Beispiele wie beispielsweise personalisierte Webseiten, Adressbuch, Newsskript, Besucherzähler und Gästebuch. Letzteres sogar objektorientiert. Wieso allerdings dem Kapitel 19, in dem nur verschiedene Webseiten, die sich dem Thema PHP widmen, vorgestellt werden, wieder zwei Kapitel mit praktischen Beispielen folgen, wird wohl ein Geheimnis der Autoren bleiben. Aus inhaltlicher Sicht wäre Kapitel 19 eher ein Kandidat für den Anhang gewesen, in dem ein solcher Abschnitt nochmals vorhanden ist.

Alles in allem gefällt das Buch aufgrund der stellenweise auch anspruchsvollen Beispiele, in die das offensichtlich vorhandene praktische Wissen der vier Autoren, die allesamt im PHP- oder Webdesign-Bereich tätig sind, eingeflossen ist. Da die Beispiele auch auf der CD-ROM vorliegen, können Sie direkt für eigene Zwecke verwendet werden. Die CD-Rom liefert aber auch gleichzeitig einen Kritikpunkt, denn scheinbar erfolgte nicht immer eine Abstimmung zwischen Verlag und Autoren, da im Text oftmals auf Software verwiesen wird, die angeblich auch auf der Buch-CD-ROM zu finden sein soll, was aber nicht der Fall ist (neben PHP4Win zum Beispiel verschiedene Editoren). Dies ist aber nicht weiter tragisch, da die CD-ROM einen ansonsten vollständigen Eindruck hinterlässt und die benötigte Software mitgeliefert wird.

Einsteiger kommen mit diesem Buch sicherlich voll auf ihre Kosten und bekommen dank der Erklärungen über die Objektorientierung und die entsprechenden Beispiele auch fortgeschrittene Kenntnisse vermittelt. Ob diese immer in zwei Stunden nachvollziehbar sind, wie es der Einband verspricht, hängt von dem jeweiligen Leser ab.

Markus Hasenbein

Mark Kronsbein, Thomas Weinert, Peter Petermann, Clemens Gutweiler

PHP – After Work

Sybex, 2001

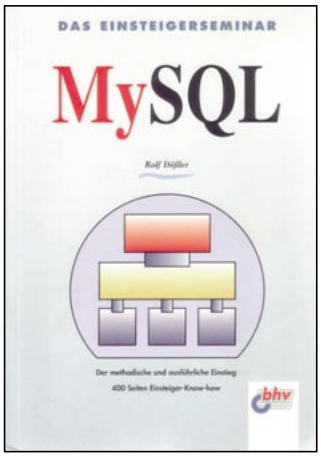
352 Seiten, inkl. CD-ROM, € 20,45

ISBN 3-8155-0517-8

Das Einsteigerseminar MySQL

Rolf Däßler

Aufgrund der einfachen Installation, Konfiguration und Bedienung sowie der günstigen Anschaffungskosten des Datenbankservers MySQL steigt dessen Wert auch auf der Beliebtheitsskala privater Anwender immer höher. Somit trauen sich auch Anfänger an das doch relativ komplexe Thema „Datenbank" heran. Die Möglichkeiten für selbstgeschriebene Applikationen, die sich ohne größeren technischen Aufwand mit MySQL bieten, sind dank einfach zu erlernender Scriptsprachen wie beispielsweise PHP fast unüberschaubar. Zudem merken mittlerweile auch viele private Anwender, dass Webpages mit dynamisch generierten Inhalten mehr Freude und Interesse beim Betrachter wecken als rein statische Auftritte. Einige Hostprovider bieten seit einiger Zeit für dynamische Web-Auftritte auch Datenbankservices an. Und MySQL ist dabei oftmals erste Wahl.



Lediglich die Einarbeitung in die stellenweise komplexe Thematik MySQL ist für Unerfahrene nicht ganz hürdenfrei. Auf diese Anwenderspezies hat sich der bhv-Verlag mit seiner Taschenbuchreihe „Das Einsteigerseminar" spezialisiert, die den Umgang mit populärer Software aller Art einsteigerfreundlich kurz und bündig inkl. zahlreicher Abbildungen vermittelt. In dieser Reihe liegt jetzt auch ein Buch für MySQL vor, das den gleichen Namen trägt.

Auf den ersten knapp 100 Seiten (erste zwei Kapitel) des beinahe 400 Seiten starken Buches erfährt der Leser grundlegende Dinge über eine relationale Datenbank. Von den Aufgaben, die an eine Datenbank gestellt werden, über den Aufbau bis hin zu Entity-Relationship- und relationalen Datenbankmodellen und der Klärung einiger Begriffe (z.B. Join, Kartesisches Produkt), die einem bei dem Umgang mit Datenbanken begegnen, reicht hier das Spektrum. Die folgenden 50 Seiten widmen sich allgemein der Sprache SQL, mit der auf die Datenbank zugegriffen wird und mit der Änderungen vorgenommen werden können. Das vierte Kapitel geht dann in die Praxis. Nach den zuvor vermittelten Grundlagen erfährt hier der Leser, wie MySQL installiert und in Betrieb genommen wird. Leider beziehen sich alle Angaben in dem Buch nur auf die Betriebssysteme einer Software-Firma aus Redmond. Informationen über die Installation unter Linux werden komplett ausgelassen. Kapitel 5 mit dem Titel „Arbeiten mit MySQL" befasst sich mit der Administration, dem Anlegen von Tabellen und der Benutzung. Als Administrationswerkzeug wird neben dem Befehl *mysqladmin* nur auf WinMySQLadmin eingegangen. Das populäre phpMyAdmin-Tool wird noch nicht einmal erwähnt. Da das Buch verstärkt auf Microsoft-Anwender setzt, wird in Kapitel 6 die Installation des ODBC-Treibers und die Verwendung von MS Access als MySQL-Clientsoftware erläutert. Die nächsten Kapitel des Buches befassen sich dann mit der Verwendung von MySQL als Web-Datenbank. Neben der kurzen Installationsbeschreibung eines Apache-Webservers (natürlich nur unter Windows) kommen dann Themen wie HTML und PHP und ein konkretes Beispiel zur Sprache. Diese Buchverwaltung behandelt die notwendigen Tätigkeiten und Scripts für eine kleine webbasierte Datenbankapplikation basierend auf PHP. Das abschließende Kapitel behandelt auf acht Seiten weitere Möglichkeiten zur Anbindung von Datenbanken im Internet. Aufgrund seines geringen Umfangs und der Thematik vermittelt dieses Kapitel den Eindruck von benötigtem Füllmaterial, um das Buch auf eine vorgegebene Seitenzahl zu hieven. Auf den letzten Seiten finden sich dann die Lösungen zu den Aufgaben, die neben einer Zusammenfassung des Kapitelinhaltes am Ende jedes Kapitels auf den Leser warten, ein Glossar, ein Anhang, der u.a. Referenzen zu SQL enthält, und ein Index.

Wer sich schnell in MySQL einarbeiten will, ohne dabei in die Tiefen eines Datenbanksystems abtauchen zu wollen, ist mit diesem Buch gut bedient. Offensichtlich geht der Autor auch davon aus, dass die meisten der Interessenten mit Windows-Betriebssystem arbeiten, da er sich nur darauf bezieht. Schade eigentlich, feiert doch MySQL gerade in Verwendung mit Linux-Servern große Erfolge. Negativ fällt auch das letzte Kapitel über die Webanbindung von Datenbanken auf. Scheinbar will er dem Leser gegenüber ein paar Begriffe wie Application Server usw. wenigstens einmal erwähnt haben. Anders dürfte dieses Kapitel oberflächlichen Inhalts nicht zu rechtfertigen sein, passt es doch so gar nicht in Konzept und Thema des Buches.

MySQL-Anfänger allerdings dürften ihre Freude an dem Buch haben, vorausgesetzt sie setzen Windows als Betriebssystem ein. Das dürfte aber wahrscheinlich die Mehrheit sein. Sie werden direkt zum Ziel geführt, ohne zu sehr von Details abgelenkt zu werden. Ambitioniertere MySQL-Benutzer sollten allerdings lieber andere Werke zur Hand nehmen, die mittlerweile in großer Zahl erschienen sind.

Markus Hasenbein

Rolf Däßler
Das Einsteigerseminar MySQL
Verlag bhv, 2001
400 Seiten, € 9,95
ISBN: 3-8266-7021-3

Planungssicherheit

von Stefan Priebsch

Software-Engineering mit PHP

Wer sich vom Skript-Programmierer zum Entwickler komplexer Web-Applikationen aufschwingen will, sieht sich plötzlich mit ganz neuen Herausforderungen konfrontiert. Applikationen sollen „überall" laufen, vollautomatisch installiert werden und keine weitere Systemkonfiguration erfordern. Damit das erste größere Projekt keine Bauchlandung wird, ist fundiertes Hintergrundwissen hilfreich.

Die ersten Gehversuche mit PHP unternehmen die meisten, indem sie ein vorhandenes System mit einigen PHP-Skripten um bestimmte Funktionalitäten erweitern. Der Kunde hat meist nur eine vage Vorstellung davon, was er eigentlich haben will und man freut sich, wenn das vorhandene System überhaupt dokumentiert ist oder jemand greifbar ist, der sich damit auskennt. Also bastelt man so lange an einer Lösung herum, bis der Kunde zufrieden ist. Spätere Änderungen und Erweiterungen werden üblicherweise noch nicht berücksichtigt. Das ist das typische Web-Scripting. Wer dagegen eine ausgewachsene Web-Applikation entwickeln will, trifft auf etwas andere Voraussetzungen: falls man bereits einen Kunden hat, hat dieser meist nur eine vage Vorstellung davon, was er eigentlich haben will. Man hat im Allgemeinen kein vorhandenes System, also auch keine Dokumentation dazu und logischerweise ist auch niemand greifbar, der sich damit auskennt. Spätere Änderungen jedoch erwartet der Kunde als Update oder Patch, am besten kostenlos. Während man im ersten Fall eine Lösung für eine definierte Umgebung liefern muss, steht man im zweiten Fall manchmal auf der sprichwörtlichen grünen Wiese.

Traue niemandem

Die Forderung, dass die Applikation auf „allen" PHP-Systemen mit „allen" Datenbanken arbeiten muss, liegt nahe. Das ist aber viel leichter gesagt als getan! Eine Anwendung wirklich plattformunabhängig zu halten, erfordert sehr viel Disziplin beim Programmieren. Selbstverständlich sind direkte System-Aufrufe beispielsweise mit *system, exec* oder *passthru* verpönt, denn etwa ein *wget* oder *gzip* existiert auf Windows-Plattformen normaler weise nicht. Shared-Memory-Befehle beispielsweise werden auf Windows-Systemen gar nicht erst unterstützt. Es empfiehlt sich, im Einzelfall unter Zuhilfenahme des Online-Handbuches auf www.php.net zu prüfen, ob verwendete Sprachkonstrukte „plattformsicher" sind. Heiße Problemkandidaten sind die Filesystem-Funktionen. Die für die PHP-Programmierung wichtigsten Unterschiede zwischen Windows und Unix sollte jeder Entwickler kennen:

- Unix unterscheidet zwischen Groß- und Kleinschreibung in Pfaden und Dateinamen, Windows nicht. Eine Datei *Foo.php* kann zwar unter Windows als *foo.php* angesprochen werden, unter Unix würde sie aber nicht gefunden.
- Unix benutzt den Schrägstrich / („forward slash") um Pfadangaben zu trennen, Windows benutzt den Rückstrich \ („backslash"). Glücklicherweise kann man in PHP-Skripten durchgängig den „forward slash" verwenden und hat damit auch in Windows-Systemen kein Problem.

- Unix hat ein anderes Rechtesystem als Windows. Wird unter Unix eine Datei mit PHP erzeugt, sollten die zu vergebenden Rechte dafür explizit gesetzt werden. Das ist unter Windows nicht möglich.

Auch wer nur eine Plattformfamilie, beispielsweise Windows, unterstützen will, sollte Vorsicht walten lassen. Das Windows-Verzeichnis beispielsweise ist unter Windows NT und Windows 2000 zwar standardmäßig *c:\winnt*, unter Windows XP dagegen *c:\windows*. Ganz abgesehen davon, dass der Benutzer bei der Installation jeweils ein anderes Verzeichnis wählen kann. Solche Systemspezifika sollten deshalb immer aus Umgebungsvariablen („Environment"-Variablen) gelesen werden; dies gilt auch für Unix. Übrigens ist es technisch ohne weiteres möglich, einem PHP-Skript „falsche" Umgebungsvariablen zuzuspielen. Hier wird nicht nur die Funktion der Applikation in Frage gestellt, sondern es ergibt sich auch ein erhebliches Sicherheitsloch. Im Zweifelsfall sollte man den Rückgriff auf Umgebungsvariablen des Betriebssystems ganz vermeiden.

Weitere Unterschiede zwischen Betriebssystemen ergeben sich aus der Sprache des installierten Systems. So ist Windows 2000 deutsch nicht gleich Windows 2000 englisch! Ein Ordner „Programme" heißt hier eben „Program Files" und zumindest Zahlen- und Datumsformate sind anders. Wirkliche Probleme entstehen aber erst, wenn exotische Betriebssystemversionen wie koreanisch oder chinesisch zum Einsatz kommen. Hier kann man nur hoffen, dass verwendete Zeichensätze korrekt unterstützt werden. Glücklicherweise sind diese Fälle aber eher selten.

Weitere Gefahren bergen die Unterschiede zwischen einzelnen PHP-Installationen. Eine CGI-Installation liefert andere Umgebungsvariablen als eine Installation als Apache-Modul; wieder andere Umgebungsvariablen liefert ein IIS oder ein ganz anderer Webserver. Meist ist man gezwungen, gewisse Umgebungsvariablen auszuwerten, um etwa herauszufinden, in welchem Pfad sich das gerade ausgeführte Skript befindet. Dies ist wichtig, wenn man dort andere Skripte per *include* oder *require* einbinden will. Die Probleme mit relativen Pfaden bei verschachtelten Verzeichnisbäumen haben schon manchen Entwickler zur Verzweiflung gebracht.

Wenn Funktionen aus PHP-Erweiterungen verwendet werden, beispielsweise die Kompressionsfunktionen aus *zlib* oder *bzip2*, wird damit vorausgesetzt, dass diese Erweiterungen auf dem Zielsystem vorhanden sind. Nicht jeder Anwender kann und darf seine PHP-Installation administrieren, gerade wer virtuelle Server von größeren Anbietern nutzt, muss mit dem leben, was installiert ist. Das bezieht sich selbstverständlich auch auf die verwendete PHP-Version. Man kann nicht immer davon ausgehen, dass die neueste PHP-Version für den Anwender verfügbar ist. Es empfiehlt sich, für ein Projekt eine bestimmte PHP-Version als Referenz und Mindestanforderung festzulegen und diese Version auch über die gesamte Projektlaufzeit zur Entwicklung einzusetzen. Will man sichergehen, dass eine Applikation auf jeder PHP-Standard-Installation lauffähig ist, muss man ganz auf die Verwendung von Extensions verzichten. Für viele PHP-eMail-Clients ist das ein Problem: ohne eingebundene *imap*-Extension verweigern sie ihren Dienst.

Ein weiteres Problem ist die Konfiguration einer PHP-Installation, die bekanntlich über die *php.ini*-Datei erfolgt. Abgesehen davon, dass nicht jeder Anwender die Möglichkeit hat, seine *php.ini* zu verändern: viele Applikationen fordern beispielsweise die Eintragung bestimmter *include*-Pfade. Werden PHP-Skripte mit *include* oder *require* eingebunden, durchsucht PHP alle im *include*-Pfad angegebenen Verzeichnisse nach dem angegebenen Dateinamen. Aber niemand garantiert, dass eine gefundene Datei *db.inc* auch tatsächlich zu der gerade laufenden Applikation gehört. Die Namensgleichheit von Dateien unterschiedlicher Applikationen kann zum Verhängnis werden. Wieder eine Sicherheitslücke im gesamten System. Sicher vermeiden lassen sich solche Probleme nur, wenn Applikationen von der *php.ini* vollständig unabhängig sind. Nur so kann sichergestellt werden, dass eine Applikation tatsächlich auf jeder PHP-Installation läuft.

Auf Nummer Sicher geht man mit einem Projekt, wenn man versucht, alle Systemabhängigkeiten zu vermeiden oder entsprechend zu kapseln. Da heute niemand weiß, welche Systemumgebung ein Kunde morgen einsetzen will, empfiehlt es sich, zwei oder mehrere Entwicklungssysteme abwechselnd zu verwenden, eines unter Windows, ein anderes unter Unix. Hier sieht man recht schnell, wo im Alltag Probleme auftreten und ob man mit der Programmierung auf der „sicheren", plattformunabhängigen Seite ist.

Es empfiehlt es sich auch, abwechselnd mit unterschiedlichen PHP-Installationen zu arbeiten, beispielsweise einmal PHP als Apache-CGI, einmal PHP als Apache-Modul und vielleicht einmal PHP als IIS-ISAPI-Filter. Will man nun noch verschiedene Datenbanken einsetzen, steht man schnell vor einer recht komplexen Systemlandschaft, die die Möglichkeiten einer Home-Office-Installation übersteigt.

Planung ist das halbe Leben

Ebenso, wie man sorgfältig planen sollte, welche Zielsysteme eine Applikation unterstützen soll, ist eine sorgfältige Planung der Software-Architektur notwendig. Auf welchem Wege man diese entwirft, ist eine Glaubensfrage. Keine der zahlreichen Methoden ist Selbstzweck (obwohl das gerade in größeren Firmen gerne vergessen wird), sondern dient dazu, mit möglichst wenig Aufwand von der Idee zum fertigen Produkt zu gelangen. Wie eine gute Software-Architektur aussieht, ist ebenfalls eine Glaubensfrage. Üblicherweise fasst man Datenzugriff, Geschäftslogik und Präsentation in jeweils einer Schicht zusammen. Alle Abhängigkeiten von Filesystem und Datenbank werden in der Datenzugriffsschicht behandelt; oberhalb dürfen sie keine Auswirkungen mehr haben. Eventuelle Abhängigkeiten vom Betriebssystem, vom verwendeten Webserver oder einer PHP-Installation sollten hier ebenfalls abgehandelt werden. Das Ergebnis ist eine systemunabhängige Applikation, die auch in Zukunft leicht auf geänderte Anforderungen in der Systemumgebung angepasst werden kann. Allerdings heißt das nicht, dass für jede PHP-Funktion eine Schalenfunktion in der Datenzugriffsschicht vor-

handen sein muss. Hier gilt es, das richtige Mittelmaß zu finden: als Faustregel gilt, dass jede Funktion oder Methode genau eine Aufgabe haben soll. Funktionen, die nur eine Umbenennung von vorhandenen PHP-Funktionen sind, sollten vermieden werden.

Die leichte Änderbarkeit von Programmcode ist eine der wesentlichen Anforderungen an eine größere Applikation. Viele Entwickler verbringen einen Großteil ihrer Zeit damit, zu entschlüsseln, was sie früher einmal selbst programmiert haben. Da sich regelmäßig die standardmäßig eingesetzten Betriebssysteme ändern, neue Datenbankversionen erscheinen, die Unterstützung eines anderen Datenbankherstellers gefordert oder auch PHP weiterentwickelt wird, wird der Entwickler immer wieder mit Änderungswünschen konfrontiert. Es ist daher ganz besonders wichtig, im Projekt frühzeitig die Änderbarkeit und Wartbarkeit zu berücksichtigen. Ein praktisches Beispiel mag dies verdeutlichen: Mit der Version 4.1.0 von PHP wird vereinfachter Zugriff auf *Get-*, *Post-* und *Cookie*-Variablen ermöglicht. Diese Änderung ist übrigens der Grund dafür, dass erstmalig die Minor-Versionsnummer von PHP erhöht wurde.

Dafür wird aus Sicherheitsaspekten nicht mehr empfohlen, *Get-*, *Post-* oder *Cookie*-Eingaben automatisch als globale Variablen zu registrieren, wie es in älteren Versionen Standard war (siehe hierzu die Variable *register_globals* in *php.ini*). Zwar wird die alte Methode weiterhin unterstützt, aber vorhandene Applikationen sollten schrittweise umgestellt werden. Überlegen wir, welche Auswirkungen diese Änderung auf unsere fiktive Applikation hat.

Wurden die *Get-*, *Post-* und *Cookie*-Variablenzugriffe gekapselt, beispielsweise in einer Eingabe-Klasse, die der Applikation alle Eingaben in einem globalen Array zur Verfügung stellt, muss nur an dieser einen Stelle der Quellcode geändert werden, um die Applikation an den neuen Zugriffsmechanismus anzupassen. Das ist leicht möglich.

Nutzt unsere fiktive Applikation aber die globale Registrierung von etwa *Get*-Eingaben, wird es schon schwieriger. Wenn mit einem *Get*-Request an ein PHP-Skript eine Variable *foo* übergeben wird, erzeugt PHP im Skript eine globale Variable *$foo*. Das ist zwar praktisch für den Programmieranfänger, da er sich um nichts kümmern muss, sondern intuitiv auf alle Eingaben zugreifen kann. Diese Vorgehensweise ist zwar ein Sicherheitsrisiko, da damit auch wichtige interne Variablen im Skript überschrieben werden könnten, aber wir hatten unsere fiktive Applikation leider vor dem Studium dieses Artikels entworfen. Wir müssen nun jede Programmstelle, an der Eingaben verarbeitet werden, ändern. Bedenkt man die Wahrscheinlichkeit, mit der man bei diesen Änderungen Fehler einbaut und den nötigen Testaufwand, wird man sich zweimal überlegen, ob man die Änderungen vornimmt. Wir bezahlen also bitter für die falsche Entscheidung im Entwurf.

Aber das Problem zieht noch größere Kreise: in älteren Versionen von PHP war diese automatische Registrierung von globalen Variablen standardmäßig aktiviert, unser Skript hätte also funktioniert. Mit der Version 4.1.0 ist die automatische Registrierung standardmäßig abgeschaltet, unsere Applikation würde also nicht funktionieren. Unsere fiktive Applikation ist nämlich von der eingesetzten PHP-Version und deren Konfiguration abhängig. Mancher Anwender würde vermutlich eine böse Überraschung erleben, wenn er seine PHP-Installation auf 4.1.0 aktualisiert, weil dann unter Umständen die bis dahin funktionierende Applikation ihren Dienst verweigert. Wenn der Anwender die Konfiguration seiner PHP-Installation nicht verändern kann, hat er mit dem Update auf 4.1.0 kaum eine Chance, unsere Applikation weiter zu betreiben.

Gutes Software-Design erkennt man daran, dass notwendige Änderungen sich auf jeweils eine Stelle im Quellcode beschränken. Das gleiche gilt für Erweiterungen: sind die Schnittstellen in der Applikation sauber definiert (mit anderen Worten: ist das Kapselungsprinzip eingehalten), können Erweiterungen der Applikation dadurch realisiert werden, dass zusätzliche Funktionen oder Klassen zum Quellcode hinzugefügt werden, ohne dass an anderen Stellen größere Änderungen notwendig sind.

Die wichtigste Kapselung in einer PHP-Applikation ist die Trennung von Logik und Design. Gerade für die Programmierer, die ihre Wurzeln im Web-Design haben, ist dies schwer zu meistern, da sie PHP als Sprache kennen gelernt haben, die in kurzen Stücken in HTML eingebettet wird. Aus der Sichtweise einer Web-Applikation wird man PHP-Programme schreiben, die HTML-Code generieren. Eine saubere Trennung von Design und Funktionalität bedeutet, dass sich Designer und Programmierer jeweils auf ihren Teil der Arbeit konzentrieren können, ohne dass sie tiefere Kenntnisse über die Arbeit des Anderen benötigen. Das Design einer Applikation kann verändert werden, ohne dass Programmzeilen angepasst werden müssen; umgekehrt kann die Programmlogik verändert werden, ohne dass man sich Gedanken um das Design machen muss.

Fazit

Um größere Projekte mit PHP erfolgreich zu meistern, braucht es mehr als bloße Grundkenntnisse in PHP. Ein Vorteil der Open-Source-Bewegung ist, dass man sich den Quellcode einer bestehenden Applikation ansehen und davon viel lernen kann. Das Copy-and-Paste-Programmieren, das am Anfang so schnellen Erfolg bringt, kann bei größeren Projekten fatal enden, denn ohne grundlegende Planung ist ein Projekt mit Sicherheit zum Scheitern verurteilt. Besonders wichtig ist allerdings, dass nicht für kleine Probleme große Lösungen gefunden werden, nur damit man alle Register des Software-Engineering ziehen kann. Das Internet ist auch deshalb so schnell groß geworden, weil viele Leute einfach schnell einmal eine kleine Lösung gebastelt haben. Diese Kreativität und Effizienz sollte man auf keinen Fall durch komplexe, schwerfällige Software-Engineering-Prozesse einschränken. Manchmal tut es eben auch ein kleines Skript.

Dipl. Inform. Stefan Priebsch ist Gründer und Geschäftsführer der e-novative GmbH in München und beschäftigt sich mit innovativen IT-Konzepten und Lösungen, beispielsweise einem Application Server in PHP. Er ist erreichbar unter stefan.priebsch@e-novative.de.

Content im Einsatz!

von Tobias Wassermann

Ein einfaches Content Management System im Eigenbau

In den Anfängen des Internets ging es nur darum, Wissen in Form von Text weltweit zu verbreiten – das ging „damals" auch recht einfach, es gab als Provider zum Großteil nur die Universitäten und größere Firmen – und Domains gab es damals (im Gegensatz zu heute) zum Nulltarif. Das Internet wuchs seitdem stetig, dies brachte eine Menge Vorteile. So können nun mehr Menschen von dem Wissen profitieren, das man im Internet vorfindet. Die Nachteile blieben leider auch nicht aus– mal ganz abgesehen davon, dass Domains nun etwas kosten – umso mehr Inhalt man auf seine Webseiten stellt, umso aufwändiger wird die Pflege.

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

Zurück zu den Anfängen – oder wie man neudeutsch sagt: „Back to the roots" – als die Seiten noch einfach waren, zum Großteil nur aus Text bestanden, die „Besucherzahlen" noch nicht allzu hoch waren und es sich bei den virtuellen Gästen meist nur um Studenten handelte, die gezielt nach Fachwissen suchten, waren alle Seiten stets statisch – es sollte sich einfach nichts auf den Seiten ändern. Damals gab es eigentlich nur eins im Web: Wissen.

Das Internet als Medium

Doch heute ist das Internet wohl das, was man Medium nennen darf. Es dient nicht mehr allein der Vermittlung von fachlichem Wissen, sondern auch als Informationsquelle, zur Unterhaltung, zum Zeitvertreib, mehr denn je zur Kommunikation – doch auch für etwas, das im Internet immer wichtiger wird: Business.

Geschäfte jedoch macht man im Netz nicht, indem man „einfach mal" eine Seite ins Netz stellt. Um wirklich erfolgreich zu sein, muss die Präsenz etwas „hermachen". Doch mit dem traditionellen Weg – jede Art von Inhalt ist statisch in eine Seite eingebunden – kommt man nicht weit.

Hier greift dann Content Management. Dabei geht es einfach nur darum, den Inhalt dynamisch in Seiten einzubinden; das macht die Webseiten mit ihren Inhalten flexibler und vor allem aktueller und einfacher in der Wartung. Gezeigt werden soll das durch ein kleines Exempel – das so innerhalb einer kleineren GbR verwendet wird. Doch nun bleibt eine Frage: Wie „managt" man Content? Nun, dies ist recht einfach, denn...

Am Anfang war die Datenbank

Wenn man erst einmal nur ein einfaches Content Management System entwickeln möchte – wir verwenden selbst eines für unsere Intranetseiten als Austausch von Informationen und News zwischen unseren Gesellschaftern –, ist selbst ein solch einfaches System ein ganz nützliches und leicht erweiterbares Werkzeug. Dieses Modell ist freilich nicht nur für Intranetseiten sondern auch für Internetseiten geeignet und mit etwas Ausbauarbeit des Ganzen lässt sich damit bestimmt ein – im Gegensatz zu den kommerziellen Lösungen – kostengünstiges Redaktionssystem realisieren.

Am besten geht man bei diesem vereinfachten Modell einfach einmal von drei Tabellen aus:

- eine Tabelle für den Content
- eine Tabelle für die Kategorien
- eine Tabelle für die Struktur der Kategorien untereinander

Doch was kommt nun alles in diese drei Tabellen? Die Tabelle *table_Content* enthält den Seiteninhalt – die Artikel, Beiträge, den Content; je nachdem, wie man es nun nennen möchte – mit einigen Metadaten. Nehmen wir einfach mal diese Felder in unsere Tabelle auf: *Content_ID*, *Content_Title*,

Feld	Typ	Attribute	Null	Standard
Content_ID	bigint(20)		Nein	
Content_Title	varchar(35)		Nein	
Content_Code	mediumtext		Nein	
Author_Name	varchar(40)		Nein	
Creation_Date	date		Nein	0000-00-00
Modify_Date	date		Nein	0000-00-00
Modify_Stamp	timestamp(14)		Ja	*NULL*
Category	smallint(6)		Nein	0

Abb. 1: Die Tabelle *table_Content*

Content_Code, *Author_ Name*, *Creation_Date*, *Modify_Date*, *Modify_Stamp*, *Category*. Ich denke, dass sich diese Felder von selbst erklären, den genauen Tabellenaufbau können Sie in Abbildung 1 sehen. Den Primary Key legen wir nur auf das Feld *Content_ID*, und einen Index benötigen wir lediglich auf *Category*. Damit haben wir die Struktur der Tabelle.

Nun zu unseren Kategorien. Eine Seite, die dynamischen Content anbietet und damit arbeitet, kann in viele Kategorien unterteilt werden Das macht auch die Verwaltung des Inhalts und die Aufteilung der Arbeit – der Inhaltsbeschaffung – recht einfach und vor allem flexibel. Eine Kategorisierung der Artikel macht auch die Navigation und das themenorientierte Suchen nach Informationen für den Benutzer einfacher. Diese Unterteilung sollte möglichst geschickt in der Datenbank vorgenommen werden, damit hier keine unnötige Rechenzeit verschwendet wird und sich die Seiten beim User möglichst schnell aufbauen. Eine Kategorie kann dabei auch Unterkategorien enthalten, dies macht eine präzisere Einteilung der Artikel möglich. Unterkategorien en masse sollten trotzdem erst eingesetzt werden, wenn es um viele Beiträge geht und es ohne Unterkategorien unübersichtlich wird.

Feld	Typ	Attribute	Null	Standard	Extra
Category_ID	int(11)		Nein		auto_increment
Category_Name	varchar(30)		Nein		

Abb. 2: *table_Category* – What's your name, category?

Die Tabelle *table_Category* ist recht einfach erklärt: Hier stehen einfach nur eine ID für die jeweilige Kategorie und ihr Name. Der Primary Key liegt einfach wieder auf der ID – fertig (siehe Abbildung 2).

Zu guter Letzt fehlt uns noch eine Tabelle, die ausschlaggebend für den Aufbau der Kategorien ist. Dieser Aufbau wird so realisiert: Jede Kategorie kann ein „Parent" oder ein „Child" sein. Ein Parent (Vater/Mutter) hat Children (Kinder), ist also eine „Überkategorie". Doch auch ein Parent kann natürlich Child sein, wenn es auch ein Parent(z. B. Großvater) hat. Diese Struktur soll nun in einer recht einfachen Tabelle abgebildet werden. Der hierbei gewählte Weg ist nicht der effizienteste, doch für unsere schlichte Absicht, ein einfaches CMS zu erstellen, sollte er reichen – Optimierung Ihrerseits nicht ausgeschlossen. In der Tabelle *table_CatStructure* legen wir fest, welche Kategorie nun Child von welcher ist. Am einfachsten ergibt sich dabei folgende Tabellen-Struktur: *Parent_ID*, *Child_ID* (siehe auch Abbildung 3).

Der Primary Key erstreckt sich dabei auf beide Felder der Tabelle, denn ein

Anzeige

Feld	Typ	Attribute	Null	Standard	Extra
Parent_ID	int(11)		Nein	0	
Child_ID	int(11)		Nein	0	

Abb. 3: Der letzte (Tabellen-)Schliff – die Struktur

Child darf in einer Parent-Kategorie nur einmal vorkommen (alles andere wäre auch schwacher bis grober Unsinn). Mit Hilfe dieser Tabelle ließen sich auch jeweils die Namen der Kategorien (egal ob Parent oder Child) ganz einfach mittels eines *Joins* ermitteln und in die jeweilige Abfrage integrieren. Jetzt ist unser Datenbank-Modell vollständig und wir können uns der nächsten Aufgabe widmen.

Templates für alle(s)

Nun ist er da – der Content; immer auf Abruf in der Datenbank abgelegt. Jetzt geht es aber um ein goldenes Prinzip: Die Entwickler verstehen zum Teil nicht so viel von Design wie ein Designer und ein Designer versteht im Normalfall nicht so viel von Programmierung wie ein Entwickler. Also sollte man auf jeden Fall verhindern, dass der Designer im Sourcecode – wenn auch unabsichtlich durch einen WYSIWYG-Editor – herumfuhrwerkt. Wir benötigen nun eine Methode, um Programmierung und Design voneinander zu trennen, damit diese unabhängig voneinander bearbeitet werden können; zum Schluss hingegen sollen diese beiden Teile wieder zusammengefügt werden – nahtlos und für den Benutzer unbemerkt, versteht sich. Die beste Methode dafür sind Templates. Hierfür gibt es verschiedene Libraries, Klassen und selbst geschriebenen Code. Da ein simples Content Management System realisiert werden soll, wird hier der Einfachheit halber auf die mehr oder weniger einschlägigen Klassen und Libraries verzichtet –begnügen wir uns mit ein paar Funktionen, die selbst geschrieben sind (sicher können sie nicht den Funktionsumfang anbieten, wie etwa *PHPLib* – doch sie sind funktionell ausreichend).

Der Vorteil eines Templates ist, dass man allem ein einheitliches Aussehen verleihen kann, was auf Webseiten doch eine große Rolle spielt, denn ein unterschiedliches Design der einzelnen Seiten innerhalb einer Präsenz wirkt doch sehr unentschlossen und abschreckend auf den Besucher.

Nun benötigt unser CMS einen Parser, der eingesetzte Templates mit dem richtigen, endgültigen Inhalt ausstattet. Zum Auslesen der einzelnen Template-Blöcke, die ersetzt werden sollen, ist die PHP-Funktion *eregi* sehr nützlich. Diese kann aus einer Variablen (hier: der Inhalt des Templates) mittels eines Suchmusters einen Block auslesen. Dieser Block wird später weiterverwendet und verarbeitet.

Doch nun zuerst einmal die *getRepeater*-Funktion:

```
function getRepeater($str)
{
 if(eregi("<!template>(.*)<!/template>", $str, $content))
                                        return $content[1];
}
```

In die Variable *$str* wird die ausgelesene Template-Datei übergeben, *$content[1]* wird dann den Block enthalten, in dem sich die einzelnen Template-Muster wiederfinden, die später zu ersetzen sind. Diese Template-Blöcke sind gut verwendbar, da wir so das Template auslesen können, die entsprechenden Stellen mit den Daten aus der Datenbank ersetzen können und dann die Ausgabe der Webseite einfach mit der *echo*-Methode produzieren. Wir lesen einen *header*, einen *footer* und unseren *repeater* aus – der Repeater ist der Teil der Datei, den wir verändern werden. Diese Gliederung in Blöcke gestaltet das Template selbst auch etwas übersichtlicher, denn so weiß man genau, dass die eigenen Template-Vorgaben nur innerhalb dieses oder jenes Blocks (hier: der *repeater*-Block) zu finden sind. Dies macht das Nachbearbeiten des Templates einfacher.

Jetzt benötigt das Management System noch eine Funktion, die die eigentliche Ersetzungsarbeit leistet. Diese Funktion *parseRepeater* ersetzt mit Hilfe eines Arrays das Template durch die eigentlichen Daten. Zum Ersetzen gibt es eine passende PHP-Funktion, *str_replace*:

```
function parseRepeater($str,$input) {
 global $default_url;

 $str = str_replace("<!Content_Title>", $input[0], $str);
 $str = str_replace("<!Author_Name>", $input[1], $str);
 $str = str_replace("<!Creation_Date>", $input[2], $str);
 $str = str_replace("<!Content_Code>", $input[3], $str);
 return $str;
}
```

Bereits im Sourcecode ist zu erkennen, in welchem Element des Arrays *input* welche Information aus der Datenbank enthalten sein soll. Selbstverständlich können noch mehr Informationen aus der Datenbank entnommen und mit *parseRepeater* in das Template integriert werden – bei dieser kleinen Auswahl an Daten handelt es sich lediglich um eine Möglichkeit der Umsetzung.

Schließlich geht es noch darum, das Template selbst in eine entsprechende Variable einzulesen – dies geschieht mit der Funktion *readTemplate*:

```
function readTemplate($filename)
{
 $templatedatei = fopen ($filename, "r");
 $template = fread ($templatedatei, filesize ($filename));
 fclose ($templatedatei);
 return $template;
}
```

Jetzt sind bereits sämtliche Funktionen für die Arbeit mit dem Template fertig. Nun bleibt noch das Auslesen und Verarbeiten der gespeicherten Daten aus der Datenbank.

Auf geht's – ran an die Daten

In unserem Beispiel verwenden wir eine MySQL-Datenbank, darauf wird auch der nachfolgende Code aufbauen. Den Sourcecode für ein anderes DBMS umzuschreiben (z. B. Postgres), ist mit PHP unter geringem Aufwand möglich.

Lassen wir uns noch einmal – unabhängig vom Template – klar werden, was dargestellt werden soll: Zum einen der Content und zum anderen der Pfad zur Kategorie, in der sich der Artikel befindet. Diese Struktur der Kategorien soll durch Verlinkung umgesetzt werden, d. h. wenn jemand auf eine Kategorie in dieser Liste klickt, soll er eine Übersicht aller Artikel in dieser gewählten Kategorie erhalten. So benötigen wir ein Skript, dem wir als Parameter übergeben können, um welche *Content_ID* oder um welche *Category_ID* es sich handelt, damit ent-

weder eine Übersicht oder ein einzelner Beitrag angezeigt wird. Auf diese Art und Weise wird es möglich, diesen Parameter direkt an die URL zu übergeben – so spart man sich die Arbeit mit Sessions. Wenn das System weiter wächst und noch Benutzermanagement oder Versionshistory integriert werden sollen, werden Sessions aus Sicherheitsgründen sinnvoll.

Zusammenfassend kann man also sagen, dass eine Funktion benötigt wird, die den Beitrag an sich ausliest, eine, die die Kategorien ausliest, anzeigt und verlinkt, und eine Prozedur, die die anderen zwei Funktionen entsprechend der Notwendigkeit aufruft.

Zuerst sollte die Auslese-Funktion für die Artikel erstellt werden. Bei dieser Funktion macht sich ein weiterer Vorteil unserer Template-Methode (erst Inhalt fertig stellen, dann an Client senden) bemerkbar: Wenn keine Verbindung zur Datenbank aufgebaut oder nicht aus der Datenbank gelesen werden kann, so gibt es weitreichende Möglichkeiten zur Fehlerbehandlung. So kann man zum Beispiel via *Header*-Anweisung auf eine andere Seite mit einer allgemeinen Fehlermeldung weiterleiten (die eventuell auch gleich eine eMail an den Webmaster sendet) oder gar ein neues Template einlesen, das für Fehlermeldungen „designt“ ist, und eine spezifische Meldung ausgeben.

Global werden nun folgende Variablen für den Verbindungsaufbau definiert:

```
$db_user="CMS"; // Datenbank-User
$db_passwd="vewdOr"; //Datenbank-Passwort
$db_server="localhost"; //Datenbank-Server
$db_dbname="CMS"; //Datenbank-Name
```

Das Auslesen des Contents geschieht mit der Funktion *readContent*. Wenn der Content gelesen wurde, muss noch der Name oder vielmehr die Bezeichnung der Kategorie ermittelt werden; dies kann

Anzeige

man ganz einfach anhand der Category-ID aus dem Artikel der Datenbank entnehmen.

Das Einfügen des Textes in das Template wird später die „Steuerungs-Funktion" übernehmen. Das Ziel ist es, dass der User immer sofort sieht, wo genau er sich befindet, und er ohne weiteres in eine Artikel-Übersicht einer übergeordneten Kategorie springen kann. Diese Funktion wird vom Aufbau und der Funktionsweise her recht ähnlich zu der sein, mit der wir den Pfad zu der aktuellen Kategorie auslesen. Sie finden diese Funktion deswegen nur im Quelltext auf der CD.

Der Aufbau der Funktion ist reverse – wir zäumen das Pferd von hinten auf. Zuerst ermittelt die Prozedur in der Datenbank den Parent der aktuellen Kategorie, dann den Parent dieses Parent – bis es irgendwann keine übergeordnete ID mehr in der Datenbank gibt. Da man vorher nicht wissen kann, wie viele übergeordnete Kategorien es sind, ist es sinnvoll, Arrays und vor allem Array-Funktionen von PHP zu verwenden. Insbesondere spielt *array_unshift* hier eine wichtige Rolle – denn damit fügt PHP in einem Array einen oder mehrere Werte am Anfang ein. Dies kommt bei der reversen Datenermittlung aus der Datenbank zum Tragen, da dann die Daten wieder „richtig herum" sind. Als zweite Array-Funktion, die noch in dieser Funktion genutzt wird, ist *count* zu nennen – da daran schlussendlich erkannt werden kann, wie viele Kategorien insgesamt ausgegeben wurden, und PHP somit weiß, wie oft es die Schleife durchlaufen soll.

Soweit sind die zwei wesentlichen Funktionen fertig, die dritte, die vorhin angesprochen worden ist, die Steuerungsfunktion, ist recht simpel, da sie nur erkennen muss, ob eine Artikel- oder eine Kategorie-ID gegeben ist. Dies kann in unserem System ganz einfach dadurch geschehen, dass geprüft wird, welche der beiden Parameter vorhanden sind. In diesem CMS ist dies durchaus vertretbar, denn hier kann nichts manipuliert werden.

Zurück zu den Templates und den dazu geschriebenen Funktionen – denn jetzt werden Sie endlich benötigt. Im Moment liest der Code zum einen die Daten aus der Datenbank aus und bereitet sie entsprechend den Bedürfnissen eines menschlich-lesbaren Dokuments (man könnte ja auch einfach nur die Parent-Child-Verknüpfungen anzeigen, statt die „Kinder" beim Namen zu nennen) aufbereitet – und die Templates sind auch geladen. Nun muss es daran gehen, beides zusammenzubringen – denn einzeln nützt es weder uns noch dem User etwas. Im Moment werden wir den Content nicht los – und der Benutzer, der ihn gern hätte, bekommt ihn nicht.

Dieser Schritt des Zusammenführens ist nun sehr einfach, da die Daten bereits so vorhanden sind, wie sie angezeigt werden sollen, und zur Aufbereitung der Templates die passenden Funktionen bereitstehen – also scheidet das große Tüfteln vor dem Kodieren aus.

Ohne Tüfteln kodieren

Sicherheit, Performance, Erweiterungen

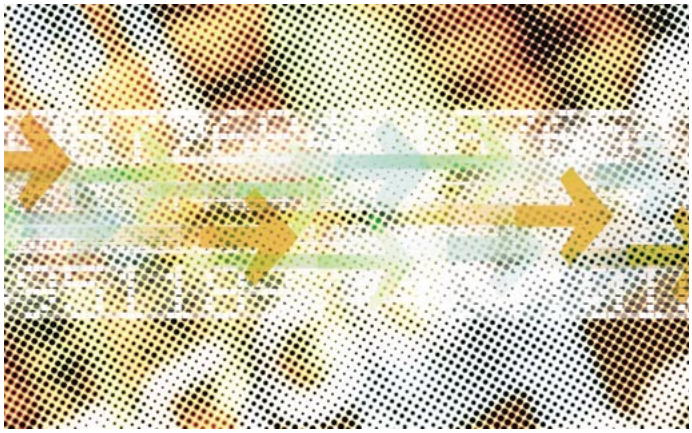
Das CMS kann in vielerlei Richtung weiterentwickelt werden – die Möglichkeiten sind hier sehr weit. Was wohl für eine wirklich gute Anwendung wichtig ist, ist eine Redaktionsschnittstelle, mit der man die Artikel und Kategorien via Browser einpflegen kann. Hier kommen dann auch die Sessions mit Benutzeridentifizierung oder alternativ *.htaccess* zum Tragen, denn es muss sichergestellt werden, dass nicht jeder machen kann, was er will. Interessant neben einem Redaktionsinterface wird wohl auch so etwas wie ein interner Besucherzähler sein, eine Möglichkeit, mit der man nachher auswerten kann, welche Artikel oft angefragt werden und welche nicht – dies lässt sich alles wunderbar mit PHP realisieren – und das mit relativ wenig Zeitaufwand.

Berechtigt ist auch die Frage nach der Performance. Zuerst muss man sich hier Gedanken darüber machen, dass das Template als Datei jedes Mal gelesen wird, sobald ein Artikel angefragt wird. Solange sich die (relativ) zeitgleichen Anfragen im unteren Spektrum bewegen, ist die Zeit dafür sehr marginal, doch sobald diese Zugriffe ansteigen, kann diese Sache zu einem Selbstläufer werden. Denn Festplatten sind im Gegensatz zum RAM immer noch relativ langsam und wenn es viele Anfragen gibt, greift zwar teilweise der Festplattencache, doch wenn der Server nicht nur für das CMS dient, sondern auch noch viele andere Dateianfragen bearbeiten muss, dann „fällt" unser Template mit aller Regelmäßigkeit aus dem Cache und muss von neuem gelesen werden.

Als Lösung kann man das Template auch in die Datenbank übernehmen, indem man dafür eine eigene Tabelle anlegt. Dies hat auch den Vorteil, dass man hier wunderbar – nach entsprechender Programmierarbeit – das Template-System erweitern kann, denn so kann man mehrere Templates verfügbar machen; welches Template jeweils auf einer Seite verwendet wird, kann man durch Übergabe eines entsprechenden Parameters im Link auf den Artikel steuern. Das Ganze bringt auch einen Performancegewinn – wenn man immer noch von sehr vielen Zugriffen ausgeht –, denn der Cache von Datenbanken liegt direkt im Speicher und nicht wie beim HDD-Cache in der Festplatte (von wo der Inhalt des Cache erst einmal durch den halben Datenbus muss...) – und die Strategien der Datenbanken, was denn nun im Cache bleibt und was nicht, sind um einiges ausgefeilter.

Mit dem Einsatz eines Datenbank-basierten Templates öffnen sich natürlich weitere Möglichkeiten, denn nun kann man das Template auch ohne größeren Aufwand auf recht sichere Weise für andere Server verfügbar machen und so etwa für mehrere Webseiten, die auf verschiedenen Servern liegen, das gleiche Design bereitstellen.

Es gibt viele weitere Möglichkeiten, wie man dieses einfache CMS weiterentwickeln kann. Als Fazit kann man zusammenfassend sagen, dass es mit recht geringem Aufwand möglich ist, ein einfaches Content Management System zu schaffen, das sich in der Praxis – vor allem in einem Intranet – gut einsetzen und leicht anpassen sowie erweitern lässt. Es gibt viel zu tun – packen Sie es an!



Effiziente Dokumentation

von Andre Gildemeister

API Dokumentationen mit PHPDoc generieren

PHPDoc ermöglicht einen Weg, um API Dokumentationen von prozeduralem oder objektorientiertem Sourcecode zu generieren und stellt somit die Maxime für eine standardisierte und einheitliche Sourcecode Dokumentation innerhalb von PHP dar. Seit der Veröffentlichung der PECL/phpdoc Extension steht eine Basis für einen PHP2XML Parser bereit, der PHPDoc mit nativem XML Code versorgt, und somit eine Parser Architektur darbietet.

Warum API Dokumentation?

Gut dokumentierte Bibliotheken entscheiden wesentlich über die Akzeptanz eines bestehenden Codes und damit auch über den kommerziellen Erfolg einer Software. So reizvoll gut dokumentierter Sourcecode in den fiktiven Vorstellungen auch aussehen mag, trifft man ihn in der Praxis eher selten an.

Um das Verständnis für den Sourcecode zu intensivieren, trägt eine API Dokumentation in hohem Maße dazu bei. Dokumentationen werden von unterschiedlichen Zielgruppen gefordert, jede hat ihren eigenen Inhalt und Sprache. Unterschätzen Sie niemals die Wichtigkeit und Notwendigkeit des Dokumentierens, auch wenn Sie es für das Unwichtigste der Welt halten.

Mit Hinblick auf die eben angesprochenen Aspekte wurde ein Tool für PHP gesucht, was dem Anwender möglichst einfach und ohne Hilfsmittel eine API Dokumentation generiert. PHPDoc schließt diese Lücke sehr effizient und steigert das Ansehen von Sourcecode beträchtlich.

Zielgruppen und Anwendungsgebiete

PHP ist längst nicht mehr die einst belächelte Skriptsprache. Aus kleinen Skripten sind echte Applikationen gewachsen, die die Benutzung von wieder verwendbaren Sourcecodes in Form von Programmbibliotheken zur Pflicht machen.

Bei komplexen Datenstrukturen oder Objektmodellen kann eine gut generierte API Dokumentation oftmals als erfolgreiche Referenz dienen. Nicht nur für den eigentlichen Anwender, sondern auch für den Entwickler, der die Referenz erneut sichtet und etwaige Schwächen im API erkennt und Unstimmigkeiten beseitigen kann. Inline-Source Dokumentation kann das Software API beschreiben und schwierige Codepassagen erklären.

Was ist PHPDoc?

PHPDoc (www.phpdoc.de) ist eine in PHP geschriebene Adaption von Javadoc. Analog zu Javadoc dient es zur Generierung von API Dokumentationen. Der PHP Sourcecode wird nach Markup respektive „Doc Comments“ gescannt und die gewonnenen Informationen herausgefiltert, die dann die spätere API Dokumentation ergeben. PHPDoc ist nicht zu verwechseln mit dem ähnlich geschriebenen „phpdoc“, was sehr oft für Verwirrung sorgt. „phpdoc“ repräsentiert das PHP Dokumentations-Projekt (www.php.net/docs.php), welches den Schwerpunkt in der Dokumentierung der PHP Extensions sieht.

Sie werden sich vermutlich fragen, welchen Sinn und Zweck PHPDoc bei Ihren PHP-Projekten spielen sollte. API Dokumentationen werden oft vernachlässigt und unterschätzt, obwohl sie bei umfangreichen Projekten oftmals zu einem wichtigen Bestandteil der Programmierung wachsen.

Javadoc ist eine Applikation von Sun Microsystems, die sich seit Jahren mit einer standardisierten Dokumentations-Syntax als etablierte Java API Spezifikation durchsetzt und einen hohen Stellenwert innerhalb von Java genießt. Die Komplexität von Webapplikationen

hat in den letzten Jahren rapide zugenommen und zeigt die gewachsenen Ansprüche in der Unternehmenswelt. PHPDoc versucht, diesen Ansprüchen gerecht zu werden.

PHPDoc als einheitlicher Dokumentations-Standard

Die mit PHP4 integrierte Code Repository namens „PHP Extension and Add-On Repository" löste von Anfang an heftige Diskussionen über Coding und Dokumentations-Richtlinien aus. Viele Entwickler waren sich darüber einig, dass es zur zentral qualitativen Code Bibliothek auch ein etabliertes Dokumentations-Tool im Javadoc-Stil geben müsse. Es sollte vor allem auf prozeduralen und objektorientierten Sourcecode ausgerichtet sein und eine leichte Handhabung und Erweiterbarkeit aufweisen. Des weiteren sollte es flexibel ausgerichtet sein, um vielseitig einsetzbar zu sein.

Standard Sourcecode Dokumentation

Diese Prämisse wurde von Ulf Wendel (Vorsitzender des PHP-Vereins) aufgegriffen und praktisch verwirklicht. Auf dem PHP Kongress 2000 präsentierte er erstmals die PHPDoc Version 1.0 Beta der breiten Öffentlichkeit.

Damals stand allerdings schon fest, dass die PHP Parser Architektur unter intensiver Verwendung einbrechen würde, da die Datenstrukturen in PHP4 nicht für eine native Parser Architektur ausgelegt sind. PHPDoc wurde ins PEAR CVS Repository eingebunden und ist aktuell im Verzeichnis *pear/PHPDoc* zu finden. Jeder wusste, dass eine neue native Parser Architektur gesucht wurde, allerdings nicht in PHP, sondern in C. Gut ein Jahr später wurde von Jan Kneschke (Open Source Entwickler) der lange Zeit erhoffte Parser ins PEAR CVS Repository eingebunden. Auf die Integration mit PHPDoc und die Zukunftsmöglichkeiten des neuen Parser Modells komme ich in den nachfolgenden Abschnitten zurück.

PHPDoc Architektur

Während der Entwicklungsphase von PHPDoc sah man im Gegensatz zu anderen, parallel laufenden Open Source Projekten mit ähnlichen Intentionen bei PHPDoc erstmals ein nahes Ende in Sichtweite. Deswegen wurde über die nicht so performante Architektur nur am Rande gesprochen. PHPDoc besitzt keinen wirklichen Parser, es werden vielmehr Informationen „Dokumentations-Tags" mit PCRE (Perl Compatible Regular Expressions) aus dem Sourcecode gefiltert, analysiert und extrahiert. Dies offenbart einige Schwächen des Designs in Bezug auf die Performance, worauf ich in einem der kommenden Abschnitte intensiv eingehen werde.

PHPDoc ist modular aufgebaut, wobei ein kleiner Kern das Zusammenspiel der folgenden Module kontrolliert:

- Parser
- Analyser
- Indexer
- XML Exporter
- XML Reader
- XML Accessor
- Renderer
- Hilfsmodule: XML Writer, File Handler, ArgV Handler

Der Parser liefert die extrahierten Informationen an die XML Module, wo die Daten letztendlich im XML Format gespeichert und vererbt werden. Danach wird der native XML Code vom Report

Listing 1

```
/**
* Kurzbeschreibung
*
* Ausführliche optionale Beschreibung, diese kann über
*  mehrere Zeilen gehen.
* Die Leerzeichen zwischen den Segmenten sind nicht
* erforderlich.
* Im folgenden sehen Sie eine Liste von Tags, welche die
* Eigenschaften beschreiben und Dokumentationstags
* enthalten.
*
* Die Reihenfolge Übersichtssatz - Kommentarblock -
* Tagliste ist vorgeschrieben.
*
* @param    string      Name der Aktion.
* @return   string      Nachricht, die mit der Aktion
                                        verbunden ist.
* @access   public
* @author   Andre Gildemeister <ag@inocreation.com>
* @version  1.0 14/01/2002 10:12
*/
function login_status($status_id) {
            return( isset($_SESSION['user_id']) ) ?
                                        true : false;
}
```

@author Name [<email>][, ...] Gültigkeit: *class, function, var, define, module, use*	Kontaktadresse zum jeweiligen Programmierer. Es können auch mehrere Autoren aufgeführt werden.
@brother (function(), $variable) Gültigkeit: class, function, var, define, module, use	*@brother* und sein Alias *@sister* ermöglichen eine Informationsvererbung. Geschwister erben die Dokumentation eines anderen Elements und überschreiben möglicherweise Dokumentationsteile.
@const[ant] description Gültigkeit: *define*	Mit *@constant* werden durch define() deklarierte Konstanten beschrieben.
@include description Gültigkeit: *use*	Definiert die Einbindung von externen Dateien.
@link URL [description] Gültigkeit: *class, function, var, module, define, use*	Mittels *@link* können Verweise auf Internetseiten angelegt werden.
@param[eter] (object objectnameltype) [$varname] [description] Gültigkeit: *function*	Die Parameter einer Funktion werden mittels *@parameter* beschrieben.
@return (object objectnameltype) [$varname] Gültigkeit: *function*	*@return* definiert den Rückgabewert einer Funktion. Wird bei einer Funktion kein Wert zurückgegeben, so wird standardmäßig void zurückgegeben.
@see [(class|module:]function()|$varname [, [(class\|module:]function()\|$varname] Gültigkeit: *class, function, var, module, define, use*	Verweise innerhalb von API Dokumentationen werden durch *@see* beschrieben.
@var[iable] (object objectnameltype) [$varname] [description] Gültigkeit: *class, function, var*	Mittels *@var* werden Klassenvariablen beschrieben. Den Wert versucht PHPDoc auszulesen.

Tabelle 1: PHPDoc Dokumentationstags in der Detailbetrachtung

Anzeige

Listing 2

```
<?php
/**
* Example for OOP in PHP4.
*
* The Constructor sets internal class-variables by given
* function parameters in
* constructor call.
*
* @author Andre Gildemeister <ag@inocreation.com>
* @version $Revision: 1.0 $
* @access public
*/
class magic {
 /**
 * Internal string class variable
 *
 * @var string
 */
 var $string_value = „“;

 /**
 * ID of document
 *
 * @var integer
 */
 var $int = 0;

 /**
 * Constructor call by given parameters
 *
 * @param   string variable value
 * @access  public
 * @see     listing_modules()
 */
 function magic($value) {
  if($this->int != 0) {
                $this->string_value = $value;
  }
 }

}
?>
```

Renderer, einem Template basierten HTML-Renderer für die Ausgabe der API Dokumentation verwendet.

Doc Comments in PHPDoc

Die Doc Comments bzw. Dokumentationskommentare, welche einfach im Sourcecode platziert werden, bilden das eigentliche Herz von PHPDoc. Für die Erstellung einer API Dokumentation sollte dem Entwickler ein einfacher Dokumentations-Standard zugemutet werden. Doc

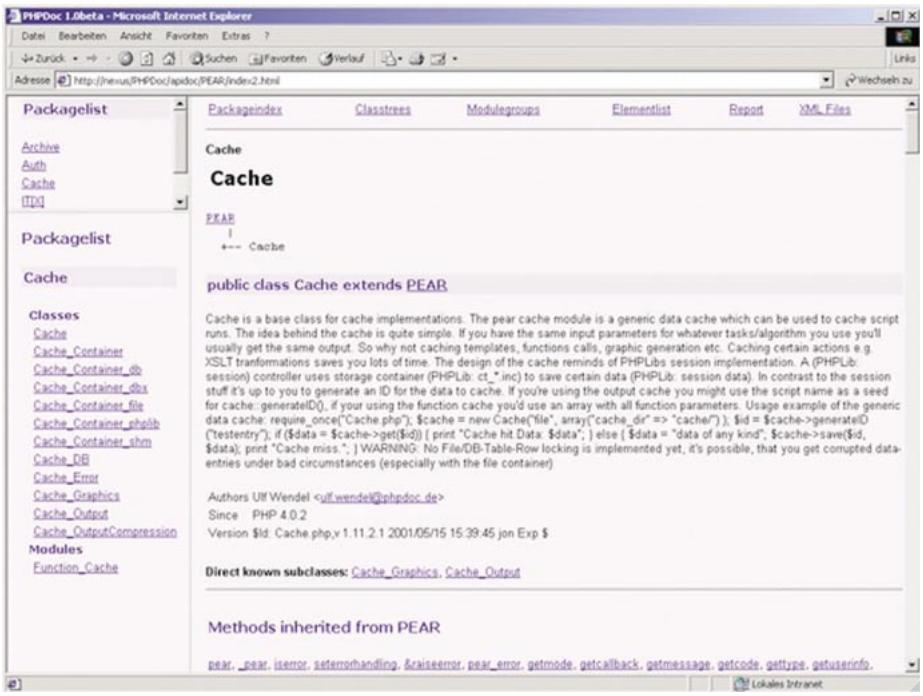


Abb. 1: Generierte PHPDoc API Dokumentation

Comments haben eine einheitliche äußere Form, enthalten Dokumentationstags und stehen vor bestimmten Keywords. Der Dokumentations-Kommentar beginnt mit „/**“, gefolgt von einer beliebigen Anzahl von Zeilen, die mit einem „ *“ beginnen und mit einer Zeile, in der nur „ */“ steht, endet (siehe Listing 1).

Wie Sie sehen, besitzt PHPDoc verschiedene Dokumentationskommentare, welche die Eigenschaften von Elementen beschreiben. Jedes Tag fängt mit einem „@“, gefolgt von einem in Kleinbuchstaben geschriebenen Namen an. In Tabelle 1 sind jeweils die wichtigsten PHPDoc Dokumentationstags mit einer kurzen Erläuterung aufgezählt, welche mit einem kommenden Beispiel ergänzt werden.

Praxisbeispiel anhand einer vollständigen API Dokumentation

In Listing 2 zeige ich Ihnen eine kleine Beispielklasse unter Berücksichtigung von PHPDoc-konformen Doc Comments. Nachdem der Sourecode steht, muss die PHPDoc Applikation nur noch mit den richtigen Daten aufgerufen werden und es wird im Verzeichnis *apidoc/keep/<name>* ein Ordner mit der generierten API Dokumentation erstellt. Template und Layoutanpassungen sind vorbehalten und können jederzeit ohne Programmiertätigkeit vorgenommen werden.

Philosophie des PECL/phpdoc Parsers

Das elementare Hauptmanko, das PHPDoc bis heute besitzt, ist die PCRE basierte „Parser“ Architektur. Komplexe Datenstrukturen weisen PHPDoc immer wieder aufs neue die Grenzen eines in PHP geschriebenen Parsers auf. Vor gut zwei Monaten wurde der lang erhoffte C Parser endlich in die PECL Bibliothek eingebunden.

Das Präfix „PECL“ steht für „PHP Extension C Libary“. Es ist eine C Bibliothek mit dem Zweck, den eigentlichen PHP Core Source so klein wie möglich zu halten, aber trotzdem eine Vielfalt an Modulen anzubieten. Deswegen wurde der ursprünglich als ext/phpdoc angesehene PHPDoc Parser auch aus den soeben genannten Gründen in die PECL Bibliothek verschoben.

Um Ihnen die interne Arbeitsweise von PECL/phpdoc ein wenig näher zu beschreiben, werde ich nachfolgend auf die einzelnen Punkte der Abb. 2 zu sprechen kommen.

Ulf Wendel, Kristian Köhntopp und Jan Kneschke, die damals im Verbund

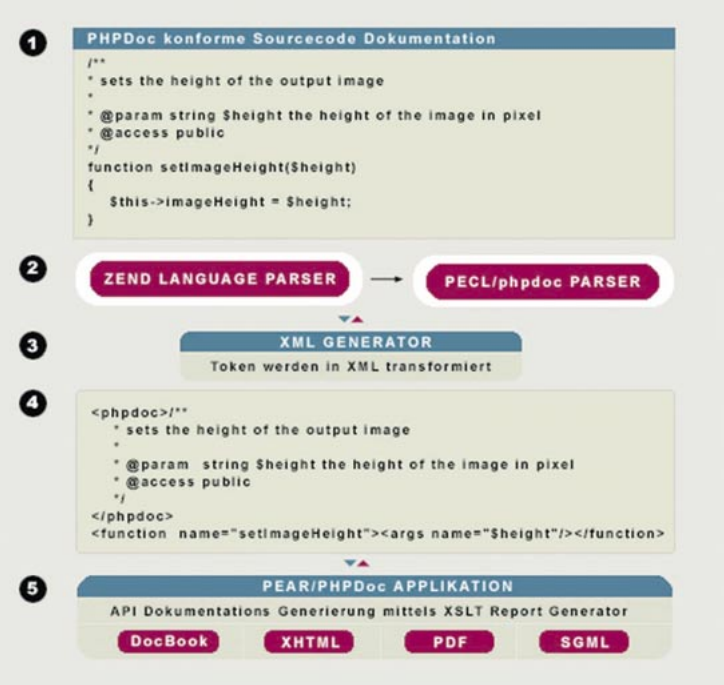


Abb. 2: PHPDoc Framework Architektur

der NetUSE AG über die Zukunft von PHPDoc philosophierten, waren sich insgesamt einig, dass mit XML als universellem Datenaustauschformat das PHPDoc Dokumentenformat feststand. Kristian Köhntopp sah in der Funktion *highlight_string()* einen möglichen Ansatz zur Erstellung eines effizienten C Parsers. *highlight_string()*, das äquivalent wie *show_source()* arbeitet, nimmt die von der Zend Engine zur Verfügung stehenden Token und generiert daraus entsprechendes HTML. Die Idee erwies sich als hervorragend, denn jetzt musste nur noch eine entsprechende Schnittstelle innerhalb einer Extension abstrahiert werden. Diese müsste die farbige HTML-Erstellung, die ohne einen Stack auskam, durch einen Stack-basierenden XML-Generator ersetzen. Dies spiegelte wesentlich mehr Implementierungsaufwand im Vergleich zu *highlight_string()* wieder. Stacks sind eine Art Puffer und arbeiten nach der LIFO Maxime „Last in, first out".

Im Schritt (1) steht eine PHPDoc-konforme Sourcecode Dokumentation als Basis zur Verfügung. Die vom Zend Language Scanner übermittelten Token werden in der zweiten Phase (2) vom Parser syntaktisch analysiert, abgefragt und letztendlich im dritten Schritt (3) in XML generiert. Token sind logische Stücke eines Textes, die vom Parser erzeugt werden. Eine Variable *$my_var* als *T_VARIABLE* Token würde beispielsweise als *<variable name="myvar">* in XML transformiert. Um das XML zu parsen, wurde ein SAX Parser implementiert. Ein Token *T_DOC_COMMENT* würde sicherlich das Optimum darstellen, damit wäre ein gesetzter Doc Comment direkt eindeutig identifiziert und das mühsame Kommentar Filtrierung würde entfallen. Der eigentliche Sinn dieses Tokens wirft aber direkt eine politische Frage auf, denn *T_DOC_COMMENT* würde bei möglicher Implementierung ein Teil der Sprache werden. Diese Fragestellung ist allerdings wegen der intelligenten Parser Architektur nicht unbedingt relevant.

Validiertes XML auf der Grundlage einer DTD ist als Ausgabe der PECL/phpdoc Extension in der vierten Phase (4) zu beobachten. PECL/phpdoc generiert sozusagen nativen XML Code als „Input" für pear/PHPDoc (5), welches für die spätere Verarbeitung und Transformation (DocBook, XHTML, PDF, SGML) zuständig ist. Diese gewünschten Transformations-Module befinden sich aber zurzeit noch in der Planung. Allerdings basiert PHPDoc immer noch auf dem angesprochenen PCRE Prinzip, welches in Zukunft entfernt und an den neuen performanten C Parser angepasst wird.

Fazit/Zukunft von PHPDoc

PHPDoc formuliert einen ersten Vorschlag für ein einheitliches und standardisiertes Dokumentations-Tool in PHP. Durch die Ausrichtung auf objektorientierten und prozeduralen Sourcecode und die finale API Dokumentationausgabe mit Hilfe von XML als Datenaustauschformat mit weitreichenden Transformierungsmöglichkeiten (XSLT), bildet PHPDoc das Tool für API Dokumentationen in PHP.

Ein GUI zur Verfeinerung der Applikation, eine Überarbeitung des jetzigen pear/PHPDoc Codes, ist allerdings dringend notwendig, um auch das Interesse von Unternehmen für kommerzielle Software zu wecken. Einige der implementierten Doc Comments finden erst mit der kommenden Zend Engine 2 einen sinnvollen Anwendungszweck.

Das mangelnde Interesse der PHP Community erschwert allerdings den Weg für ein standardisiertes Sourcecode Dokumentations-Tool in PHP. Das Engagement und die Wichtigkeit von PHPDoc wurden in der vergangenen Zeit in keiner Weise vernachlässigt, allerdings standen die Hauptentwickler meistens alleine da. Aus dem kommerziellen Umfeld kamen nur wenige Kontakte zustande, die eine weitere Entwicklung für ein einheitliches Dokumentations-Tool förderten.

Ich hoffe, mein Artikel hat einen Teil dazu beigetragen, dass sich diese Situation in Zukunft verbessert und sich Ihr Interesse rund um PHPDoc gesteigert hat. ■

Links

- Offizielle PHPDoc Homepage: www.phpdoc.de
- PHPDoc PHP-Kongress 2000 Slides www.phpdoc.de/kongress
- API Dokumentationen in Javadoc java.sun.com/j2se/javadoc/index.html
- Das CVS-Repository von PEAR: cvs.php.net/cvs.php/pear

Schwarze Magie oder Handwerk?

von Alex Aulbach

Reguläre Ausdrücke in PHP – Teil 1: Grundlagen

Man sagt, ein PHP-Entwickler sollte neben der Programmiersprache PHP auch HTML beherrschen. Auch SQL- und JavaScript-Kenntnisse machen sich in einem Bewerbungsbogen gut. Aber reguläre Ausdrücke? Das sind doch diese kryptische Ausdrücke, die so aussehen, als hätte man sich beim Telefonieren auf die Tastatur gelehnt?

In der Tat sehen Ausdrücke wie zum Beispiel `!(?:^<|\s*)([^\s"\'=]+)\s*?(?(?==)=\s*?(?(?=([„\']))[„\'](.*)\2|(\S*)))(?:\s+|>$)!UsS` sogar für Experten ziemlich mysteriös aus, erinnern sie doch an Zauberformeln oder Schwarze Magie. Aber auch für Zauberei gilt: Man kann sie erlernen und der Nutzen ist enorm. Dieser Kurs ist für Anfänger wie Fortgeschrittene konzipiert. Anfängern soll er über die Hürden hinweghelfen, die am Anfang üblicherweise auftauchen, und Fortgeschrittene können den Kurs als ein gutes Nachschlagewerk nutzen.

Was sind nun reguläre Ausdrücke und vor allem, wofür kann man sie verwenden? Praxisbeispiel: Ich musste einen Verzeichnisscanner entwickeln, der für diverse Dateinamen spezielle Konvertierungen anstoßen sollte. Also im Prinzip nur ein Vergleich: Soll die Datei konvertiert werden oder nicht? Da ich von regulären Ausdrücken noch keine Ahnung hatte, musste ich natürlich alles „per Hand" programmieren – und es wurde ein böser Hack daraus. Ich schrieb zum Beispiel „Ist das erste Zeichen im Namen ein Punkt? Ja, dann nächster Name". Und so weiter. Nach zehn solcher Regeln war der Code der Schleife länger als eine Bildschirmseite und bis auf sechs Ebenen eingerückt. Ein Jahr später kam der Kunde und sagte, es hätte sich eine neue Regel ergeben. Ich fragte mich eine Stunde lang, welcher Idiot das wohl programmiert hat und wo ich in diesem Hack die Änderungen reinschreiben soll. Weil, wenn ich es so ändern wollte, wie es am einfachsten gewesen wäre, dann hatte es Seiteneffekte. Also zog ich einen Rattenschwanz von Änderungen hinter mir her und konnte aufgrund des Zeitdrucks nicht vernünftig testen und das rächte sich, als ich am gleichen Abend vor dem Spielfilm vom Chef angerufen wurde, der vom Kunden angerufen wurde, dass etwas nicht mehr geht und das ganze System jetzt hängt. Die Fehlersuche beginnt.

Ich nahm mir – weil's nun sowieso egal war – die Zeit und wühlte in der Dokumentation – man hat ja schon einiges von Patternmatching gehört. Ich fummelte mir etwas zusammen, was zu funktionieren schien und wunderte mich, dass meine Schleife plötzlich nur noch sechs Zeilen lang war und das ganze nur eine halbe Stunde dauerte und ich so noch den Rest des Spielfilms sehen konnte. Das Programm läuft noch heute damit. Die einzige Änderung war, dass der Ausdruck dann in eine Konfigurationsvariable eingebaut wurde, sodass bei einer Änderung der diesbezüglichen Anforderungen dies von den Admins des Kunden erledigt werden kann. Das spart also richtig Zeit (und Geld).

Reguläre Ausdrücke in PHP sind also gut zum Prüfen, ob eine Zeichenkette ein bestimmtes Kriterium erfüllt, zum Überprüfen von Benutzereingaben, Prüfung auf syntaktische Korrektheit von Datenformaten, Entscheidung, um welche Art von Format es sich handelt, Umwandlung eines Formats in ein anderes, Durchsuchen und Auftrennen von Text, Schreiben von Minimal-Parsern und insbesondere zum Schreiben von umfangreichen Artikeln zu dem Thema.

In der Tat sind reguläre Ausdrücke ein absolut unerschöpfliches Thema, denn es gibt kaum einen regulären Ausdruck, den man nicht irgendwie besser schreiben könnte. Ich zum Beispiel kam vor sieben Jahren zum ersten Mal damit in Berührung. Seitdem verwende ich sie fast täglich – und lerne ständig dazu. Trotzdem war ich überrascht, als ich bei den Recherchen zu diesem Artikel auf mir bis dato gänzlich unbekannte Zusammenhänge und etliche falsche Annahmen stieß.

Das zumindest war für mich die Motivation, diesen Artikel zu schreiben, denn ich fand es lästig, ständig in den zerstreut herumliegenden Docs zu wühlen, und die Anfänge des Artikels sind eigentlich nur gesammelte Notizen und URLs, von denen ich merkte, dass ich sie in dem Zusammenhang immer wieder benötigte.

Begrifflichkeiten

Reguläre Ausdrücke – im englischen „Regular Expressions" und daher im Folgenden *Regex* genannt (die Abkürzung Regexp ist auch recht geläufig, nur wird mir nach dessen Benutzung oft „Gesundheit" gewünscht), sind schon lange in der Informatik bekannt.

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

Regexe beschreiben Suchmuster (Patterns) in Zeichenketten (Strings) und können feststellen, ob in einem Suchtext das vorgegebene Suchmuster vorkommt (der Ausdruck *passt*, er „*matcht*“) oder nicht. Abgesehen von dieser Hauptaufgabe gibt es eine Reihe abgeleiteter Funktionen, zum Beispiel das Ersetzen oder Zerteilen von Text.

In diesem Artikel sehen alle Regexe `R(so)` aus (das innerhalb der Klammern ist der Regex) – ich schreibe im Artikel diese so, wie sie in PHP mit einfachen Anführungszeichen geschrieben werden müssten. Suchtexte sehen in diesem Artikel *so* aus, und das was der Regex vom Suchtext findet, sieht so aus.

Die Verarbeitung eines Regex übernimmt die so genannte *Regex-Maschine*. Diese besteht im Wesentlichen aus zwei Teilen. Der erste Teil liest den Regex (das Suchmuster), interpretiert diesen und baut (kompiliert) daraus eine Art Mini-Programm. Der zweite Teil führt dieses Programm aus: Er läuft über den Suchtext, es wird sozusagen für jedes Zeichen im Suchtext untersucht, ob der Regex passt. Kam die Regex-Maschine am Ende des Strings an, ohne etwas zu matchen, wurde das Pattern nicht gefunden.

Das ist natürlich eine sehr grobe Vereinfachung des ganzen Vorgangs, im Literaturverzeichnis finden ich einige interessante Verweise [1].

Geschichte ist meist unbeliebt, in diesem Fall aber tatsächlich notwendig, um zu verstehen, warum reguläre Ausdrücke so sind, wie sie sind und warum es so viele verschiedene Arten von Regexen gibt. Ich beginne den Rückblick mit den 50er Jahren. Damals entwickelte sich gerade die Informatik aus einem Teilzweig der Mathematik. Hier wurden Konzepte und Ideen erstellt, die man heutzutage als Informatiker in Vorlesungen wie Compilerbau vermittelt bekommt. Im Kasten habe ich für Interessierte eine kleine Gedächtnisstütze zur Auffrischung geschrieben – wer sich sehr häufig mit Regexen auseinandersetzen muss, der sollte diese theoretischen Grundlagen kennen.

Erste reale Regex-Implementationen kann man in den 60ern finden, eine breite Anwendergemeinde fanden sie zum ersten Mal im Unix-Editor *ed*.

grep ist ebenfalls ein bekanntes Unix-Program, welches aus *ed* heraus entstanden ist (es ist einfach die Shell-Variante des *ed*-Befehls „Global Regular Expression Print“, gesprochen wird es einfach „gräpp“) und es wird auch heute noch sehr erfolgreich eingesetzt, so erfolgreich, dass es *grep* für praktisch alle Betriebssysteme gibt. Mittlerweile bin ich auch schon mitten in den 70er Jahren angelangt.

Es folgten andere Programme wie zum Beispiel *awk* („Aho, Kernighan, Weinberger“, gesprochen „ohhg“), aber vor allem auch Editoren wie *vi* („wi ei“) oder *emacs* („i-mäx“). Und es entstand dann auch das Problem mit den verschiedenen Dialekten (oder auch Geschmacksrichtungen) der Regexe.

Grüß Gott! Moin, moin!

In den (bekannten Unix-)Editoren *vi* oder *emacs* können Sie mit dem Regex `R(\$var\>)` eine PHP-Variable namens `$var` finden. Mit der Programmiersprache Perl geht das so nicht. Sie müssen den Regex schreiben als `R(\$var\b)`. Und in PHP, wenn Sie die Funktion *ereg()* benutzen, sieht das so aus: `R(\\$var[[:>:]])`

Und hier sehen Sie schon: Es schaut alles zwar prinzipiell gleich aus, aber irgendwie doch anders: In *vi/emacs* steht da `R(\>)`, in *perl* `R(\b)` und in PHP `R([[:>:]])`. Und trotzdem machen alle Regexe das Gleiche.

Für die Praxis hat dies ein paar Auswirkungen. Denn für Sie stellt sich die Frage, welchen der Dialekte Sie erlernen sollten und wo die Unterschiede zu den anderen liegen. Folglich stellt sich die Frage, wo ein PHP-Entwickler mit regulären Ausdrücken in Kontakt kommt. Allein PHP hat von sich aus als einzige verbreitete Programmiersprache gleich zwei verschiedene Regex-Maschinen von vornherein fest eingebaut bekommen. Aber als

Regexe und theoretische Informatik

Obwohl Regexe den Köpfen von Neurophysiologen und Linguisten entsprungen sind, hat sich die theoretische Informatik recht rasch deren Ideen angeeignet. Die Grundlagen, die vor 40 Jahren zur Entwicklung von Compilern gelegt wurden, haben sich bis heute nicht verändert und sind daher heute fester Lehrstoff in einem Informatikstudium.

Noam Chomsky führte 1956 die *Chomsky-Hierarchie* ein. Die Klassen dieser Hirarchie werden in Typ-0- bis Typ-4-Sprachen unterteilt. Der Begriff „Sprache“ ist in dem Zusammenhang natürlich ein mathematisch konstruierter Begriff und besagt, dass eine Sprache aus einer Menge von Wörtern besteht, die aus Zeichen eines Alphabets stammen, die aus einer Menge von Zeichen/Buchstaben kommen. Typ-0 sind Sprachen, die sich nicht mehr mit Hilfe eines Programms erkennen lassen (auch *rekursiv aufzählbar*, natürliche Sprache zum Beispiel), Typ-1 sind *kontextsensitive* Sprachen, die mit erheblichem Rechenaufwand erkannt werden können, Typ-2 sind *kontextfreie* Sprachen. PHP zählt mit leichten Einschränkungen zum Beispiel dazu, aber generell fast jede Programmiersprache, und Typ-3 sind *reguläre* Sprachen, über die wir hier reden. Mit regulären Ausdrücken können die Typ-3-Sprachen (auch rechts- bzw. linkslineare Sprachen genannt) vollständig beschrieben werden. Wer sich für die genauen Zusammenhänge interessiert, kann mehr unter [15] nachlesen.

In vielen Vorlesungen wird nicht erwähnt, dass die meisten realen Implementierungen von Regexen auch Sprachen erkennen können, die *nicht im klassischen Sinn Typ-3* sind. Zum Beispiel erkennt der Ausdruck `R(([a-z]+)\1)` alle Wörter der Struktur *ww*, zum Beispiel `blablа` oder `bonbon`. Diese Sprache ist vom Typ-1!

Für uns als Praktiker sind nur folgende Dinge wichtig: Regexe können in der Praxis mehr, wie die Theorie besagt.

- Das Alphabet ist der benutzte Zeichensatz, bei allen Beispielen lege ich vorerst den Zeichensatz ISO-Latin-1 zugrunde [10], [11].
- In der Praxis ist es oft schwer zu erkennen, ob ein Problem (das Erkennen einer Sprache) mit Regexen lösbar ist. Dafür gibt es auch kein Kochrezept. Mit zunehmender Erfahrung erkennt man dies aber immer leichter und entdeckt sogar Anwendungsfälle, die einem nicht auf den ersten Blick ins Auge fallen.
- Regexe sind nichts anderes wie endliche Automaten und *man kann einen Regex immer in einen solchen umwandeln* (und umgekehrt). Regexe sind demnach nichts anderes als ungeheuer kompakter und trotzdem flexibler Programmcode. Kompakter Code ist in den meisten Fällen gut, wenn es darum geht, fehlerfrei und schnell zu entwickeln, denn wer weniger schreibt, kann weniger Schreibfehler machen und braucht nicht so lange dazu. Das bedeutet allerdings auch, dass man seine Regexe sehr gründlich testen muss (sonst schafft man mehr Fehler in kürzerer Zeit)!

Anzeige

Anzeige

PHP-Entwickler kommen Sie noch an ganz anderen Stellen mit Regexen in Berührung. Im Einzelnen:

- *Erweiterte POSIX-kompatible reguläre Ausdrücke* (kurz **POSIX**) [2], [3], [4], [5], [7] – seit spätestens PHP 2,
- *Perl-kompatible reguläre Ausdrücke* (kurz **PCRE**) [8] – seit PHP 3.0,
- *JavaScript RegExp* [9] – in Zusammenhang mit HTML-Programmierung,
- Reguläre Ausdrücke und deren Verwandte in *SQL* (zum Beispiel MySQL [6]) sowie
- andere Programme – ich erwähnte schon *grep* oder *awk*. Aber auch der benutzte Editor. Oder XSLT. Die gute Beherrschung von Regexen und die Kenntnis der Abweichungen ermöglichen hier enorme Zeiteinsparungen.

Dieser Kurs wird im Wesentlichen die ersten zwei Punkte behandeln, aber das sind schon zwei unterschiedliche Regex-Dialekte. Das heißt, sie sind zwar sehr ähnlich, denn alle Regex-Dialekte entstammen der ursprünglichen Definition aus der theoretischen Informatik, es gibt aber kleine und wichtige Unterschiede, die entstanden, weil die Programme von unterschiedlichen Leuten zu unterschiedlichen Zeiten für unterschiedliche Zwecke programmiert wurden. Es nutzt nichts, sich darüber zu ärgern, denn jeder Programmierer, der Regexe implementierte, „erfand" neue Erweiterungen oder behandelte bestimmte Aspekte eines Regex anders, einfach um damit besser auf die speziellen Bedürfnisse der Anwender einzugehen. Es gibt inzwischen so viele Implementierungen von Regex-Maschinen, dass es auch für Experten unmöglich ist, alle zu kennen oder klar zu entscheiden, ob deren Syntax oder gar Grammatik zueinander kompatibel ist.

Es gab aber auch Versuche zur Standardisierung. POSIX (Portable Operating System Interface for Unix) war zum Beispiel ein Gremium, welches sich um die Standardisierung von Unix-Funktionen kümmerte – also eine quasi unmögliche Aufgabe. Die Geschichte dieses POSIX-Standards ist übrigens höchst interessant und verwirrend. Der Standard hat doch einiges bewirkt; so ist etwa die Funktion *fopen()*, die jeder PHP-Entwickler sicher kennt, so eine POSIX-Funktion. Ohne so einen übergreifenden Standard wäre es nicht möglich, unter Unix und Windows eine Datei auf gleiche Weise zu öffnen. In Bezug auf POSIX-Regexe entstanden zunächst die *Basic Regular Expressions*, die dann 1992 unter dem IEEE Standard 1003.2 oder gebräuchlicher als der POSIX.2-Standard zu *Extended Regular Expressions* erweitert wurden. Diese werden meist mit BRE und ERE abgekürzt, ich belasse es hier aber bei POSIX und meine damit die ERE; die Entwickler der GNU C-Lib-Regex lassen sich darüber übrigens auch recht negativ aus („Bugs: Having two kinds of REs is a botch...").

Diese POSIX-Regexe wurden fast von Anfang an in PHP integriert, das Interface ist standardisiert und somit kann man es durch beliebige POSIX-Regex-Libraries austauschen. In PHP hat man sich aufgrund verschiedener Probleme dafür entschieden, eine solche Lib in der Distribution mitzuliefern, die standardmäßig auch in PHP einkompiliert wird. Dabei handelt es sich um die POSIX-Regex Library von Henry Spencer. Sie befindet sich im Verzeichnis *regex/* der PHP Distribution, dort finden Sie alles Weitere bezüglich dieser speziellen Implementation.

Im Gegensatz dazu entstanden die *Perl-kompatiblen Regexe* [8]. Dazu muss ich auch wieder etwas ausholen. Perl ist eine Skriptsprache, ganz ähnlich wie PHP. Im Unterschied zu PHP sind die Regexe in Perl jedoch in die Sprache integriert. So kennt Perl statt Regex-Funktionen Regex-Operatoren. Das sieht so aus:

```
PHP: if ( preg_match('/hugo/',$source) ) ...
PERL: if ( $source =~ /hugo/ ) ...
```

PCRE jedoch ist – wie der Name schon sagt – eine fast eigenständige Entwicklung, eine zu Perl fast kompatible Library. Seit spätestens PHP Version 3.0.9 ist sie fest in PHP integriert. Sie befindet sich im Verzeichnis *ext/pcre/pcrelib/* der PHP Distribution.

Was sollte man als Entwickler nun lernen? Ginge es in diesem Artikel nur um PHP, dann würde ich ganz klar zu PCRE raten, denn diese Regex-Maschine ist einfach besser: Sie kann mehr, ist besser implementiert, wird ständig weiterentwickelt, ist dadurch fehlerfreier, ist schneller und hat durch Perl auch einen hohen Verbreitungsgrad. POSIX-Regexe eignen sich meiner Meinung nach aber besser für den **Einstieg**, weil sie die schlichtere und allgemeinere Form darstellen und viel älter sind. POSIX-Regexe sind klassisch schlicht, PCRE sind barock und verschnörkelt. Beherrschen Sie POSIX-Regexe, dann können Sie schon sehr viele wunderbare Werkzeuge benutzen – und speziell unter Unix können Sie sich dadurch viel Arbeit, bis hin zu Tagen, sparen, denn es sind dann immer nur Dialekte; die Prinzipien unterscheiden sich nicht. Wegen der weiteren Verbreitung von POSIX und der einfacheren Grammatik werde ich daher im ersten Teil dieses Kurses POSIX-Regexe erklären und im zweiten Teil PCRE.

Ich muss außerdem noch anmerken, dass ich mich auf die spezielle Implementation der Regex-Library von Henry Spencer beziehe, da ich davon ausgehe, dass die Standardeinstellungen beim Kompilieren von PHP normalerweise nicht geändert werden. Sie hat in PHP bezüglich des POSIX-Standards einige Einschränkungen, also wundern Sie sich nicht. Insgesamt macht diese Tatsache den ersten Teil einfacher.

Beispielumgebung 1

```
<?php
$source ='Das ist ein Text in dem hugo vorkommt';
$pattern='hugo';
if ( ereg($pattern,$source) )
        echo „Gefunden: $source";
echo „\n";
?>
```

Wenn das Programm **Gefunden** ausgibt, dann *matcht* das Pattern. Natürlich könnte man das noch viel schöner programmieren, aber das ist alles Luxus.

Auf der CD befindet sich dazu übrigens eine Mini-Testumgebung, die ich immer dazu verwende, um meine Regexe zu testen, es ist nämlich eine schlechte Idee, einen Regex aus dem Stehgreif aufzuschreiben, erfahrungsgemäß trifft man gerade als Anfänger so auf dumme Schreibfehler und kommt nicht weiter. Mit einer Testumgebung kann man sich den Regex systematisch zusammenbauen, kann Teilergebnisse matchen und Schritt für Schritt verbessern.

Ich fange ganz brav an. Wer schon mal einen Browser (oder irgend ein textorientiertes Programm) benutzt hat, weiß, dass in fast allen Fällen eine Suchfunktion eingebaut ist: Sie rufen auf eine mehr oder weniger komfortable Art (STRG-F zum Beispiel) diese Funktion auf, geben in das Feld SUCHEN NACH zum Beispiel `R(hugo)` (siehe Beispiel {3}) ein und starten die Suche. Kommt **hugo** in dem Dokument vor, wird es gefunden und der Browser zeigt das an, indem er das Wort markiert und an die entsprechende Stelle im Browserfenster springt.

Genau so verhält sich unsere Beispielumgebung: **$source** wäre der Inhalt im Browserfenster und **$pattern** der Inhalt des Felds SUCHEN NACH. Für die Regex-Maschine ist so etwas eine leichte Fingerübung, denn diese schreibt ein Suchprogramm, welches den ganzen Suchstring Zeichen für Zeichen nach einem `R(h)` durchsucht. Wenn sie ein *h* gefunden hat, dann merkt sie sich die Position und sucht nach einem *u*, einem *g* und einem *o*. Matcht das `o`, dann matcht der gesamte Ausdruck und der Match ist insgesamt wahr. Sie können sich das in etwa so vorstellen:

```
Suchtext:    Das ist ein Text, in dem hugo vorkommt
Hauptroutine: nnnnnnnnnnnnnnnnnnnnnnnj...X
Unterroutine:          jjj
```

Sobald das `h` gefunden wird, verzweigt die Regex-Maschine in ein „Unterprogramm“, das nach `R(ugo)` sucht. Uns geht es dabei nur ums Prinzip, die Wirklichkeit ist etwas anders. Liefert das einen Match, dann matcht der gesamte Ausdruck: Der Regex ist wahr. Das zieht übrigens noch eine andere Konsequenz nach sich: Es wird immer *der erste Match* gefunden:

```
Suchtext:    Das ist ein Text, in dem hugh und hugo vorkommt
Hauptroutine: nnnnnnnnnnnnnnnnnnnnnnnnj..nnnnnnj X
Unterroutine:          jjn   jjj
```

Schlaue Regex-Maschinen würden sogar merken, dass sie, wenn die Hauptroutine bis drei Zeichen vor Ende des Suchstrings nichts gefunden hat, die Suche abbrechen können, denn der Rest kann sowieso nicht mehr matchen.

Wenn Sie ein wenig damit herumspielen, indem Sie zum Beispiel das Wort *hugo* im *$source*-String verändern, werden Sie merken, dass der Vergleich mit dem Browser so nicht stimmt. Im Gegensatz zum Browser matcht der Ausdruck nicht, wenn Sie *Hugo* (also großes statt kleines h) in die Quelle schreiben:

```
Suchtext:    Das ist ein Text, in dem Hugo vorkommt
Hauptroutine:
nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnX
```

Beim Browser gibt es da meist eine Checkbox, die Sie extra anwählen müssen, damit er exakt das Wort `hugo` findet. Ansonsten findet der Browser „HUGO“ in allen Variationen, also `hugo`, `Hugo`, `HuGo` oder `hUGO` und so weiter.

Bei den Regex ist das nicht so. Wie unpraktisch...

Aber das ist so in Ordnung. Denn die *Nicht-Unterscheidung* zwischen Groß- und Kleinschreibung erfordert zusätzlichen Aufwand. Sie können sich das so vorstellen, dass die Regex-Maschine ja nicht mehr prüfen muss, ob die Zeichenfolge `h`, `u`, `g`, `o` gefunden wird, sondern sie muss jetzt suchen nach `h oder H`, `u oder U`, `g oder G`, `o oder O` {6}. Das ist natürlich viel komplizierter. Aber weil der Fall so häufig auftritt, hat man für diese Art von

Zeichen:	
{1} `R(h)`	matcht `h`
{2} `R([h])`	`h` (das Zeichen der Zeichenklasse h)
Zeichenketten:	
{3} `R(hugo)`	matcht `hugo`
{4} `R(der)`	matcht in *Das Federvieh der Witwe Bolte* das `der` in „Federvieh“ und nicht das Wort „der“.
REGEL: Der erste Match wird gefunden (nicht der zweite oder dritte).	
Zeichenklassen:	
{5} `R([hH]ugo)`	`hugo` und `Hugo`
{6} `R([hH][uU][gG][oO])`	`hugo`, `Hugo`, `hUGO`, `hUgo`, `HuGo` usw.
Hinweis: Gleiches können Sie ohne Zeichenklassen durch die Funktion *eregi()* statt *ereg()* erreichen. Die Funktion *sql_regcase()* (siehe Text) liefert eine in eine solche Klein-Großbuchstaben-Zeichenklasse umgewandelte Zeichenkette.	
REGEL: Verwenden Sie *eregi()* nur, wenn **alle** Buchstaben nicht auf Groß- oder Kleinschreibung unterschieden werden sollen, da es langsamer ist.	
Hinweis: Beachten Sie, dass für *eregi()* die Zeichen Ä und ä trotzdem noch unterschiedlich sind!	
{7} `R([hHuU]go)`	`hgo`, `ugo`, `Hgo` und `Ugo`
REGEL: Eine Zeichenklasse findet immer nur ein Zeichen!	
{8} `R([a-z])`	die Zeichen `a`, `b`, `c` usw. bis `z`
{9} `R([0-9])`	die Zeichen `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8` und `9`
{10} `R([a-zA-Z0-9])`	Klein- und Großbuchstaben sowie Zahlen
{11} `R([20-49])`	die Zeichen `0`, `1`, `2`, `3`, `4` und `9` (nicht Zeichenfolgen 20 bis 49!)
{12} `R([2-4][0-9])`	matcht Zahlen zwischen 20 und 49.
Negierte Zeichenklassen:	
{13} `R([^a-zA-Z0-9])`	das genaue Gegenteil von {10}, also z.B. die Zeichen `!`, `?`, `-`, `:`, `;`, `.`, `=` usw.
{14} `R([^0-9][0-9])`	eine Nicht-Ziffer gefolgt von einer Ziffer, z.B. – 3
Escapen in Zeichenklassen:	
Spezielle Zeichen sind `]` sowie `^` und `-`. `R([])` muss am Anfang stehen, `R(^)` darf nicht am Anfang stehen, `R(-)` muss am Anfang oder Ende stehen. Innerhalb der eckigen Klammern haben Backslashes keine spezielle Bedeutung, der Backslash ist ein Backslash.	
{15} `R([]0-9^.-])`	`]`, Ziffern, `^`, `.` (Punkt) und `-`
{16} `R([^]^0-9.-])`	gegenteilige Klasse zu {15}
{17} `R([+–0-9])`	die Zeichen `+`, `,` und `–`, sowie Ziffern
{18} `R([[.-.]-9])`	die Zeichen `-`, `.` usw., sowie Ziffern (`R([.-.])` ist eine sog. Kollationssequenz).
{19} `R([\-[.].]])` Findet die Zeichen `\` und `]`, das `R(\)` muss in diesem Fall in PHP nicht escapet werden.	

Tabelle1: Zeichen, Zeichenketten und Zeichenklassen

Suche spezielle Möglichkeiten geschaffen. Im PHP-Skript muss die Zeile mit dem *IF* wie folgt geändert werden

```
if ( eregi($pattern,$source) ) # eregi() statt ereg()
```

Jetzt matcht das Ergebnis auf Groß- oder Kleinbuchstaben. Sie können sich das „i“ merken mit „Case-**i**nsensitive“ (übersetzbar als „Umschalt-unempfindlich“); das kommt überall im Zusammenhang mit Regexen vor!

Damit wären die Möglichkeiten ausgeschöpft, die jeder normale Browser bietet. Bei Regex fängt es hier jedoch gerade erst an! Ich erweitere das Problem: Der Regex soll sowohl `Hugo` als auch `hugo` matchen, aber sonst keine anderen Varianten. Ändern Sie das Programm also zurück zur Anfangsversion und setzen einfach ein neues Pattern:

```
$source ='Das ist ein Text, in dem hugo vorkommt';
$pattern='[hH]ugo';
```

Das `R([hH])` im Pattern ist eine so genannte Zeichenklasse. Dieses Pattern matcht, wenn entweder ein „h oder ein H“ kommt, gefolgt von `R(ugo)` {4}.

Es gibt übrigens eine PHP-Funktion, die das für einen erledigt: *sql_regcase()*. Sie wurde ursprünglich in PHP/FI dazu eingeführt, um MySQL-Benutzern die unterschiedslose Suche nach Klein-/Großbuchstaben zu vereinfachen, daher kommt der unvermutete Name. Mittlerweile wird sie jedoch zusammen mit den anderen POSIX-Regex-Funktionen dokumentiert. Ein

```
echo sql_regcase('hugo');
```

gibt folgendes aus:

```
[Hh][Uu][Gg][Oo]
```

Das ist im Prinzip auch das, was die Regex-Maschine bei *eregi()* daraus macht! Sie jagt den Ausdruck intern sozusagen durch die Umsetzungsfunktion *sql_regcase()* und ersetzt alle literalen Buchstaben durch deren Groß-Klein-Zeichenklasse (sehr vereinfacht ausgedrückt, die Wirklichkeit ist wesentlich komplexer – für uns reicht diese Vorstellung der Umsetzung jedoch völlig aus). Daher sollten Sie dies auch wenn möglich vermeiden. Denn bei der Funktion *eregi()* muss doppelter Aufwand betrieben werden: Erstens die interne Umsetzung des Suchpatterns und zweitens die jeweils doppelte Prüfung der Buchstaben (Ist es ein „H“? Nein? Ist es ein „h“?). Gerade bei sehr großen zu durchsuchenden Datenmengen ergeben sich hier beachtenswerte Geschwindigkeitsunterschiede.

Das gilt auch für die dritte Methode: Man lässt den Suchtext vor der Suche über die Funktion *strtolower()* laufen und formuliert den Regex so, dass er nur auf Kleinbuchstaben matcht:

```
If ( ereg($pattern,strtolower($source)) )
  0 #  vorheriges Umwandeln in Kleinbuchstaben
```

Für jede der Methoden gilt: Es kommt auf den Einzelfall an, welche die beste ist. Je nach üblichem Suchtext und Pattern ist die eine oder andere Methode effektiver und man muss sich als Entwickler immer wieder neu entscheiden.

Ich ändere nun die Aufgabenstellung: Sie sollen herausfinden, ob im obigen Beispielsatz eine Zeichenfolge „**Irgendein Kleinbuchstabe gefolgt von einem o**“ vorkommt. Jetzt wird es umständlich, der Regex sieht nun so aus:

```
$pattern='[abcdefghijklmnopqrstuvwxyz]o';
```

Da haben sich die Regex-Entwickler natürlich schon ziemlich bald Gedanken gemacht, wie man das vereinfachen könnte, denn sie fanden es sehr lästig, das immer Gleiche so aufwändig aufzuschreiben. Herausgekommen ist folgende, identische Schreibweise:

```
$pattern='[a-z]o';
```

Also die Zeichen aus der Menge „a bis z“ {8}. Den Rest erledigt die Regex-Maschine, denn eigentlich wird hier nur der entsprechende Zeichencode verwendet, also die ASCII-Zeichen Nummer **97 bis 122,** und daraus die entsprechende Zeichenklasse kompiliert. Um alle Klein- **und** Großbuchstaben zu suchen, könnten Sie folglich schreiben:

```
$pattern='[A-Za-z]o';
```

Das ist eine Zeichenklasse, die **zwei Bereiche** {10} abdeckt. Davon können Sie beliebig viele machen. Als Bereich können Sie aber auch Ziffern verwenden:

```
$source ='-123';
$pattern='[0-9][0-9][0-9]';
```

Das sind drei Zeichenklassen auf einmal. Damit können Sie sich zwar keine Wünsche erfüllen, aber Sie können herausfinden, ob im Suchstring drei Ziffern nacheinander vorkommen – das ist genau betrachtet das Gleiche. Der Ausdruck findet also nicht das Minuszeichen vor *123*, denn das ist entgegen der landläufigen Meinung keine Ziffer! Was ist aber, wenn Sie das Gegenteil wissen wollen, also ob *keine* Ziffern vorkommen? Suchen Sie mal eine Nicht-Ziffer gefolgt von einer Ziffer! Was ist eine *Nicht-Ziffer*?

Das ist ganz einfach: Es gibt 255 Zeichen (in POSIX-Regexen, wir verwenden übrigens den ISO-Latin-1 Zeichensatz [10], [11]), oder auch „eine Menge von Zeichen“. Die Menge enthält eine Teilmenge „Ziffern“, also sind „Nicht-Ziffern“ der ganze Rest. Wie schreiben Sie das als Regex?

```
$pattern='[^0-9][0-9]';
```

Das Zeichen `R(^)` matcht die gegenteilige Zeichenklasse {14}. Sie negiert die Menge der Zeichen, die damit gefunden werden würde; hier würde der Ausdruck auf „jedes Zeichen *außer* Ziffern, gefolgt von einer Ziffer“ matchen. Jetzt würde also das Minuszeichen gefunden werden, es gehört ja nicht zu den Zahlen, dafür würden keine Zahlen mehr gefunden werden. Der Ausdruck passt daher zum Beispiel nicht mehr auf *123-*, weil ja nach der ersten Nicht-Ziffer nichts mehr folgt. Erinnern Sie sich: Ein Regex matcht nicht, wenn er am Ende ankommt, ohne etwas zu finden. Es wird nicht versucht, „irgendwie“ dann doch noch zu matchen, das funktioniert einfach nicht. (Wie Mr. Spock sind Regexe immer logisch. Manchmal treibt einen das in den Wahnsinn.)

Das mit den Ziffern funktioniert übrigens nur deshalb so wunderschön, weil man bei der Definition des ASCII-Zei-

chensatzes die Zeichen 0, 1, 2 ... 9 nacheinander aufsteigend definiert hat. Also nicht verwechseln: Das sind Zeichen, keine Zahlen. Beispiel {11}

```
$pattern='[20-49]';
```

findet entgegen den Erwartungen nicht die Zahlenfolgen *20*, *21*, *22* ... *49*, sondern die **Ziffern** (=**Zeichen**) `0`, `1`, `2`, `3`, `4` und `9`! Eine äquivalente Schreibweise wäre `R([0-49])`. *Eine Zeichenklasse findet immer nur ein Zeichen.*

Die Beispiele {15} bis {19} zeigen, wie man innerhalb von Zeichenklassen escapet (was escapen ist, wird im nächsten Abschnitt erklärt). Überlegen Sie mit: In den Zeichenklassen gibt es spezielle Zeichen, das sind `R(])` (Ende einer Zeichenklasse), `R(-)` (Minus, Bereich) und `R(^)` (Negierung einer Zeichenklasse). Sie können diese Zeichen nun einfach an eine Stelle schreiben, an der sie von der Regex-Maschine nicht vermutet würden. Die Beispiele zeigen das ausführlich. Vorsicht, Sie können die Zeichen hier nicht (wie eventuell gewohnt) durch Backslash escapen, der Backslash hat in einer POSIX-Zeichenklasse keine spezielle Bedeutung.

Literale, vordefinierte Zeichenklassen

Was sind Literale? Hier – also im Zusammenhang mit Regexen – sind Literale Zeichenfolgen, die andere Zeichen, Zeichenklassen und Zeichenfolgen (Anker, nächster Abschnitt) umschreiben. Man vergrößert sich dadurch seinen verfügbaren Zeichensatz.

Der ganze Vorgang nennt sich auch *escapen*, man „flüchtet" durch das Voranstellen eines speziellen Zeichens (bei Regex das Zeichen Backslash) in einen anderen Zeichensatz. Richtig wäre auch *literalisieren* (nicht mit literarisieren verwechseln), man spricht im Zusammenhang mit Zeichenketten aber auch vom *quoten* – also dem Anführen, Anführungszeichen heißen daher auch *Quotes*. In diesem Artikel wurden zum Beispiel alle Regexe durch `R()` gequotet. Einzelne Zeichen werden escapet (vor normaler Interpretation geschützt), damit man sie als Literal erkennen kann. Sie sollten ein paar Grundregeln ({21} bis {24}) dabei beachten und insbesondere, wer hier was interpretiert (siehe Kasten).

Das wohl bekannteste, aber am meisten überschätzte Literal in Regex ist der Punkt {25}. Der hat die Bedeutung „Zeichenklasse aller Zeichen". Genauer gesagt ist die Bedeutung von den Einstellungen der POSIX Regex abhängig, aber in PHP gilt „Alle Zeichen" (ich erwähne das so ausführlich, weil das in PCRE anders behandelt wird). Das heißt, ein `R(.)` in einer Regex matcht genau ein beliebiges

Literale:	
{21} `R(\\)`	der Backslash selbst
{22} `R(\n)`	matcht `n` und nicht wie erwartet Newline!
{23} `R(\*)`	matcht *
Hinweis: POSIX kennt {21} und {25} als einzige Literale.	
REGEL: In POSIX kann man keine binären Daten matchen. Man kann nicht nach dem Zeichencode 0 (NULL) suchen. NULL ist der String-Terminator.	
{24} `R(\( \) \[ \< \> \{ \})`	Escape-Sequenzen für Klammer auf, Klammer zu, eckige
`R(\. | \^ \$ \+ \? \1 \2 \3)`	Klammer auf, und so weiter
Generell müssen alle Zeichen, die sonst eine Bedeutung (Literal, Steuerzeichen) haben, durch Backslashes escapet werden, alle anderen *können* escapet werden. Alle Zeichen erhalten dadurch ihre Bedeutung als einfaches literales Zeichen zurück. In jedem Fall muss aber beachtet werden, dass durch das Quoting in PHP zuvor bereits ein Interpretationsprozess über den Regex stattfand.	
Vordefinierte Zeichenklassen:	
{25} `R(.)` (Punkt)	ein beliebiges Zeichen (wirklich jedes)
REGEL: Benutzen Sie den Punkt nur, wenn es wirklich bedeutungslos ist, welches Zeichen matcht.	
{26} `R([0-9][0-9].[0-9][0-9].[0-9][0-9])`	unklar definiert: `10011001`, `90-60-90`, ...
{27} `R([0-9][0-9]\.[0-9][0-9]\.[0-9][0-9])`	Datum der Form dd.mm.yy (keine Inhaltliche Prüfung, `99.99.99` wäre auch korrekt)
POSIX Zeichenklassen:	
POSIX Zeichenklassen werden innerhalb einer Zeichenklasse angewendet und werden in der Zeichenfolge [:*KLASSE*:] eingebettet.	
alnum	Zeichen und Buchstaben, identisch mit `R([a-zA-Z0-9])`
alpha	Zeichen, im Deutschen identisch mit der Zeichenklasse `R([a-zA-Z])`
blank	Leerzeichen, systemabhängig, normalerweise Leerzeichen und Tabulator. Nicht verwechseln mit „space"!
cntrl	Control-Zeichen (Die Zeichen 1 bis 31, sowie das Zeichen 127)
digit	Ziffern, identisch mit `R([0-9])`
graph	darstellbare Zeichen, identisch mit „print", außer dem Leerzeichen (Zeichen 32)
lower	Kleinbuchstaben `R([a-z])`
print	druckbare Zeichen, die Zeichen 32 bis 126, sowie 160 bis 255
punct	alles außer Controll- oder Alphanumerischen Zeichen: `R([^[:alnum:][:cntrl:]])`
space	Leerzeichen (32), Newline (10), Carriage Return (13), Tabulator (9), Seitenvorschub (8)
upper	Großbuchstaben: `R([A-Z])`
xdigit	Hexadezimalzeichen: `R([0-9a-fA-F])`
{30} `R([[:higher:]][[:lower:]])`	Großbuchstabe gefolgt von Kleinbuchstabe: `Au`, `Go`, `Mu`, `Df` usw.
{31} `R([:higher:][:lower:])`	`:`, `h`, `i`, `g`, `e` oder `r` gefolgt von `:`, `l`, `o`, `w`, `e` oder `r`. Höchstwahrscheinlich nicht das, was man wollte ...
{32} `R([[:alpha:]À-ÖØ-öø-ÿ])`	alle Buchstaben aus dem ISO-Latin-1 Zeichensatz
Hinweis: Nach POSIX gibt es außerdem noch so genannte Kollationssequenzen und Äquivalenzklassen. Henry Spencers Regex-Lib hat dies nicht implementiert, einziger sinnvoller Zweck ist das Escapen, wie in Beispiel {17}bis {19}. Daher erwähne ich sie hier nur der Vollständigkeit halber.	

Tabelle 2: Literale und vordefinierte Zeichenklassen

Escapen in PHP und Regex

Wie Sie vielleicht schon bemerkt haben, wurden alle Beispiele mit einfachen Anführungszeichen (Apostroph) geschrieben. Dieses schützt den Inhalt vor dem PHP-Parser. Nur **** und **\'** werden in PHP in den Apostrophen interpretiert. Das **** wird als **** (Backslash) interpretiert und das **\'** als **'** (Apostroph). Alle anderen Zeichen werden ansonsten 1:1 übernommen, zum Beispiel auch **\t** oder **\n**. Dies hat einige Auswirkungen:

```
<?php
$text1="Text1 mit Newline:\nNL1\nNL2";
 # doppelte Anführungszeichen
$text2='Text2 mit Newline:\nNL1\tNL2';
# einfache Anführungszeichen
echo „$text1\n$text2";
?>
```

Das Ergebnis sieht so aus:

```
Text1 mit Newline:
NL1
NL2
Text2 mit Newline:\nNL1\nNL2
```

PHP interpretiert keine Literale, die es nicht kennt. *\n* ist innerhalb von doppelten Anführungszeichen in PHP ein Literal, aber nicht innerhalb von Apostrophen. Das Problem ist nun, dass bei POSIX-Regexen eine andere Regel greift. In POSIX werden alle mit Backslash escapten Zeichen, die keine Literale sind, als das Zeichen ohne Backslash interpretiert:

```
<?php
ereg("Text1 mit Newline:\nNL1\nNL2",'dummy');
# Regex1
ereg('Text2 mit Newline:\nNL1\tNL2','dummy');
# Regex2
?>
```

Das macht jetzt natürlich wenig Sinn, aber es geht darum, dass Sie verstehen, was passiert. Regex1 würde nun „korrekt" (wollte man wirklich Newline finden oder die Zeichenfolge *\n*?) weiterverarbeitet werden. Bei *Regex2* sieht das aber ganz anders aus; der Regex, nach dem jetzt gesucht würde sieht so aus:

```
Text2 mit Newline:nNL1nNL2
```

Das `R(\n)` wird in der POSIX-Regex-Maschine zu `R(n)`. Das ist eine – je nach Zweck – höchst erwünschte oder unerwünschte Eigenschaft. Ich bin der Meinung, dass sie favorisiert werden sollte, denn sie erspart dem Programmierer, wenn er das Prinzip verstanden hat, sich damit herum zu ärgern, dass PHP die eigentlich für die Regex-Maschine gedachten Literale entliteralisiert. In den meisten Fällen wird ein Regex komplizierter, wenn Sie ihn in doppelte Anführungszeichen stecken (weil ja zwei Literalebenen greifen) und insbesondere im Zusammenhang mit PCRE können höchst unerwartete Seiteneffekte auftreten. Nebenbei werden einfache Anführungszeichen in PHP schneller abgearbeitet. Aber wie immer gibt es natürlich Ausnahmen, bei denen doppelte Anführungszeichen sinn-voller sind.

Zeichen. Als Anfänger macht man zwei häufige Fehler. Der erste ist, dass der Punkt verwendet wird, wenn eine eingeschränktere Zeichenklasse dies auch tun würde. Es ist bequemer. Deswegen sagte ich auch, es wird überschätzt. Ich verwende den Punkt daher nur, wenn es wirklich egal ist, was gematcht werden soll, dann ist er natürlich sehr praktisch.

Ein anderer häufiger Fehler (der auch Profis noch passiert) ist, den Punkt nicht zu escapen. Betrachten Sie das Beispiel {26}. Sie sollten ihre Regexe also systematisch auf diesen Fehler prüfen, wie Sie das bei allen systematischen Programmierfehlern machen sollten – zum Beispiel bei *if($i=0)* statt *if($i==0)*.

Das nächste Beispiel ist identisch zu {27}:

```
$pattern='[[:digit:]][[:digit:]]\.[[:digit:]][[:digit:]]\.[[:digit:]][[:digit:]]';
```

Das sieht jetzt natürlich ein wenig umständlich aus. Ich finde vordefinierte Zeichenklassen (siehe auch [14]) aber oft recht praktisch. Beispielsweise entspricht

```
$pattern='%[[:xdigit:]][[:xdigit:]]';   # Variante 1
```

der Suche nach einem Prozentzeichen und zwei darauffolgenden Hexadezimalzahlen. Das gleiche macht auch

```
$pattern='%[0-9A-Fa-f][0-9A-Fa-f]';   # Variante 2
```

Warum ist Variante 1 besser? Ganz einfach: Es steht da, Sie sehen als Entwickler auf einen Blick, was hier matcht, die Variante 2 ist schwerer zu verstehen. Es wäre guter Programmierstil, diesen Regex zu kommentieren; bei Variante 1 ist das nicht unbedingt notwendig, so lange der Regex nicht viel länger wird.

Den wahren Nutzen bemerken Sie aber erst, wenn Sie dies für die Suche nach Wörtern verwenden. Denn es ist einfach irgendwie schneller und eleganter zu schreiben `R([[:alnum:]])` statt `R([a-zA-Z0-9])`.

Wenn Sie Wörter (in Regex ein höchst biegsamer Begriff) nun immer mit `R([[:alnum:]])` (also ein Zeichen eines „Worts") matchen, haben Sie Regex noch nicht verstanden; Sie müssen das immer wieder neu überlegen! Es kann zum Beispiel eine gute Idee sein, bei der Prüfung des Namens alles zuzulassen, was wie ein Buchstabe aussieht. Das sieht dann (beim ISO-8859-1 Zeichensatz) so aus {32}:

```
$pattern='[a-zA-ZÀ-ÖØ-öø-ÿ]';  # alternativ: [[:alpha:]À-ÖØ-öø-ÿ]
```

`R([a-zA-Z])` kennen Sie schon. Der Rest sind die Zeichen von 192 bis 214, 216 bis 246 und 248 bis 255 – jeder gute Editor

{40} `R(^)`	ankert am Anfang des Suchtexts
{41} `R($)`	ankert am Ende des Suchstrings
{42} `R(^$)`	matcht leeren Suchtext
{43} `R(^\^)`	matcht Zeile, die mit ^ beginnt (`R(^^)` geht auch)
{44} `R(^hugo$)`	eine Zeile (oder Suchstring), in der nur das Wort *hugo* vorkommt
{45} `R($hugo)`	Match wird niemals gefunden (war vielleicht `R(\$hugo)` gemeint?)
{46} `R([[:<:]])`	ankert an Wortanfängen
`R([[:>:]])`	ankert an Wortenden
{47} `R([[:<:]][[:alnum:]][[:alnum:]][[:>:]])`	Ein Zwei-Buchstaben-Wort, matcht zum Beispiel das `in` in *Dies ist ein Text in dem hugo vorkommt*
{48} `R([[:<:]]hugo[[:>:]])`	findet *hugo*, nicht jedoch *Tschepschugow*
{49} `R(\$var[[:>:]])`	findet PHP-Variablen namens `$var`, aber nicht `$variable`
REGEL: Anker benötigen keine Zeichen zum Matchen, siehe {42}, sondern sie krallen sich an bestimmten Positionen im Suchtext ein.	
Hinweis: Die Entwickler der GNU Regex schreiben zum Wortgrenzen Anker: „BUGS: The syntax for word boundaries is incredibly ugly." Dem stimme ich voll zu (Syntax für vordefinierte Zeichenklassen wird missbraucht). Die meisten Regex-Maschinen benutzen für Wortgrenzen übrigens die Literale `R(\<)` und `R(\>)`, und/oder `R(\b)`.	

Tabelle 3: Kontext-Anker

Anzeige

besitzt die Möglichkeit, die Zeichen nach Code einzugeben. Eine Prüfroutine, die das mit diesem Regex prüft, kann dadurch auch Namen wie Åke oder Citroën zulassen – normalerweise gibt es keinen vernünftigen Grund, das nicht zu erlauben.

Kontext-Anker

Anker sind etwas völlig Neues. Denn sie finden keine Zeichen, sondern sie finden Positionen im Text, die bestimmte Kriterien erfüllen. Sie krallen sich dort sozusagen fest und der Rest des Regex muss dann testen, ob er matchen kann oder nicht. Das ist für den Anfang sicherlich schwierig zu verstehen. Beispiel:

```
$source='Das ist ein Text, in dem hugo vorkommt';
$pattern='[[:<:]]hugo[[:>:]]';
```

Sie können sich das so vorstellen, dass der Anker `R([[:<:]])` {46} unter anderen auf die Position **zwischen** dem *Leerzeichen* und dem *h* matcht. Das Problem ist aber: Dort ist ja nichts. Es kann dort nichts matchen. Das wäre auch schlecht, denn wenn es zum Beispiel das *h* matchen würde, dann könnte ja das nachfolgende `R(h)` aus `R(hugo)` nicht matchen.
Das ist auch der Sinn solcher Anker: Sie matchen, *ohne* dass sie ein Zeichen „verbrauchen". Das ist sehr nützlich. Warum? Ich nehme mal an, Sie wollen wissen ob das Wort `R(hugo)` im Suchtext

```
$source='Dies ist ein Text, in dem Dimitri Tschepschugow
vorkommt'; # Test 2
```

vorkommt. [Das ist/war der Polizeipräsident von Moskau – können Sie sich übrigens vorstellen, wie schwierig es ist, ein Wort zu finden, das „hugo" enthält?] Ohne Wortgrenzen-Anker täten Sie sich damit ziemlich schwer. Denn Sie müssten ja einen Regex schreiben, der erkennt, dass die Zeichen davor und dahinter Nicht-Wörter sind, also etwa so:

```
$pattern='[^[:alnum:]]hugo[^[:alnum:]]';
```

Wunderbar sagen Sie, Sie haben alles getestet: Der Regex erkennt nun einwandfrei, dass das Wort `R(hugo)` nicht im Suchtext Test 2 vorkommt, aber sehr wohl in unserem ersten Suchtext. Genau was Sie wollen. Aber irgendwo muss doch der Haken dabei sein? Jawohl! Denn ändern Sie

```
$source='hugo';
```

dann wird erkannt, dass das Wort **nicht** vorkommt. Die Regex-Maschine schaut nur kurz drüber und stellt erleichtert fest, dass die Aufgabe ja primitiv ist und meldet, ohne, dass sie überhaupt zu Vergleichen anfängt zurück, dass das nicht matchen kann, weil sie ja mehr Zeichen matchen müsste als überhaupt im Suchtext stehen.

Anker sollten Sie eigentlich – wenn das möglich ist – immer verwenden, denn sie können einen Regex ungemein beschleunigen. Sie können sich das besonders gut beim ^-Anker vorstellen: Der Rest des Regex muss nur noch am Anfang des Suchtexts matchen. Matcht er nicht, ist die Suche auch schon fertig. Es muss nicht der Rest des Suchtexts gematcht werden.

Intervalle, Quantifier (Wiederholungen)

Nehmen Sie das Beispiel mit dem Datum von vorhin. Es war doch ziemlich umständlich, die zwei Ziffern immer als `R([0-9][0-9])` hinschreiben zu müssen. Es wäre einfacher, wenn Sie sagen könnten, ich möchte `R([0-9])` zwei mal hintereinander. Das geht mit Regex so:

{50} `R(^h{1}$)`	h (identische Schreibweise `R(^h$)`!)
{51} `R(^h{4}$)`	**genau 4 Wiederholungen:** hhhh
{52} `R(^h{1,4}$)`	**eine bis 4 Wiederholungen:** h, hh, hhh oder hhhh
{53} `R(^h{0,}$)`	nichts, h, hh, hhh, hhhh, hhhhh und so weiter.
{54} `R(^h*$)`	**beliebig viele Zeichen:** identisch zu {53}
{55} `R(^h{1,}$)`	h, hh, hhh, hhhh, hhhhh und so weiter
{56} `R(^h+$)`	**mindestens ein Zeichen:** identisch zu {55}
{57} `R(^h{0,1}$)`	nichts oder h
{58} `R(^h?$)`	**höchstens ein Zeichen:** Identisch zu {57}
REGEL: *Quantifier sind gierig.* Sie matchen immer so viel, wie sie nur können und geben es nur wieder her, wenn sie müssen!	
{59} `R([0-9]{2}\.[0-9]{2}\.[0-9]{2,4})`	Datum: `31.12.01` oder `31.12.2001` (aber auch `98.89.989`)
{60} `R([[:<:]][[:upper:]][[:lower:]]*[[:>:]])`	Wörter, die mit Großbuchstaben beginnen
{61} `R([Hh]*[Hh]*)`	Das zweite `R([Hh]*)` matcht nie. Die Regex-Maschine muss dennoch prüfen, ob ein längerer Match möglich wäre.
REGEL: `R(*)`, `R(?)` oder `R({0,...})` Quantifier matchen *immer*, ein so zusammengesetzter Ausdruck wird komplexer. Geben Sie daher `R(+)` oder `R({1,...})` den Vorzug, wenn möglich.	
REGEL: Regex-Maschinen können bei einer kleinen Anzahl von Wiederholungen meist schneller matchen, wenn Sie den Ausdruck ohne Intervalle schreiben. Also statt `R(h{3})` besser `R(hhh)`.	

Tabelle 4: Intervalle, Quantifier (Wiederholungen)

Mit Alternativen können Sie beliebig viele Teilregexe matchen.	
{70} `R(a|b)`	a oder b
{71} `R(hugo|Hugo)`	hugo oder Hugo, siehe {5}
{72} `R("[^"]+"|\'[^\']+\')`	gequoteter String mit doppelten oder einfachen Anführungszeichen: „Mein Name ist Hugo"
{73} `R(rot|grün|blau|gelb)`	rot oder grün oder blau oder gelb
{74} `R([[:upper:]]+|[[:lower:]]+)`	entweder Groß- oder Kleinbuchstabige Wörter: hugo, HUGO
{75} `R(function[[:space:]]+[[:alnum:]_]+ [[:space:]]*\(\)|function[[:space:]]+ [[:alnum:]_]+[[:space:]]*\(.+\))`	PHP-Funktionsaufruf ohne und mit Parameter: `function hugo($bla)`
{76} `R(hugo|hugoboss)`	Wenn der Suchtext *hugoboss* enthält, dann wird hugoboss gefunden, obwohl hugo genau so matchen würde und hugoboss erst die zweite Alternative ist.
REGEL: POSIX-Regexe finden immer den **ersten und längsten** Treffer. Siehe auch {61}	

Tabelle 5: Alternativen

```
$pattern='[0-9]{2}\.[0-9]{2}\.[0-9]{2}';
```

Das `R({2})` (Zahl in geschweiften Klammern) bewirkt, dass das vorhergehende Zeichen zwei mal gefunden werden muss, damit der Regex matcht. Viel Sinn macht das nicht unbedingt, denn der Ausdruck ist nur minimal kürzer, aber die Regex-Maschine braucht jetzt länger und der Ausdruck wird unlesbarer. Aber Sie können jetzt zum Beispiel sogar erkennen, ob es sich um eine zwei- oder vierstellige Jahrezahl handelt, das ging vorher nicht:

```
$pattern='[0-9]{2}\.[0-9]{2}\.[0-9]{2,4}';
```

Hm, stimmt natürlich nicht ganz. Die Jahreszahl könnte jetzt zwei, **drei** oder vier Stellen haben.

Man kann mit Quantifiern als Anfänger auch böse reinfallen. Ein typischer Fehler ist:

```
7$pattern='[hHuU]{2}go';
```

Siehe auch {7}. Das matcht einwandfrei den Suchtext **hugo, Hugo, hUgo** oder **Hugo**. Aber auch **UHgo** und **HHgo**. Oder **Uhgo**. Also aufpassen, wenn Sie Zeichenklassen mit Quantifiern mischen. Sehr beliebte Fehlerursache ist in diesem Zusammenhang der Stern, weil hier ja hinzukommt, dass auch nichts gematcht werden kann.

Quantifier in POSIX (und jeder Regex) sind **gierig**. Das soll heißen, dass sie die maximale Menge von Zeichen matchen, die ihnen erlaubt ist, und nur widerwillig wieder loslassen. Genauer gesagt heißt es da, dass ein Regex immer den ersten und gleichzeitig längsten Match findet. Beispiel:

```
$pattern='[Hh]*[Hh]*';
```

In diesem Fall wird das zweite `R([Hh]*)` nie etwas finden. Er verlangsamt die ganze Suche nur ungemein, da die Regex-Maschine ja prüfen muss, ob sie nicht vielleicht doch etwas findet. Die Probleme, die sich aus diesen Zusammenhängen ergeben, werden unter dem Begriff *Backtracking* zusammengefasst und der zweite Teil dieses Kurses beschäftigt sich ausführlich damit.

Für den Moment muss man sich zwei Dinge merken: Ein Regex wie zum Beispiel `R([Hh]*)` matcht **immer**; der Regex liefert immer wahr. Ist das Ganze ein Teil eines größeren Regex, dann ist das Verhalten des Gesamt-Regex dadurch insgesamt schwieriger vorauszusagen und die Regex-Maschine infolgedessen langsamer. Lassen Sie daher der Regex-Maschine durch die Formulierung des Regex möglichst wenig Entscheidungsspielraum, matchen Sie immer so viel wie möglich.

Alternativen

Speziell bei Alternativen kann das Backtracking noch weitere unangenehme Folgen haben, denn POSIX-Regex-Maschinen (nicht PCRE!) sind gezwungen, in so einem Fall alle Alternativen zu testen, denn sie können ja nicht sicher sein, dass in einer weiter rechts stehenden Alternative nicht vielleicht ein Match gefunden wird, der länger ist. Das Beispiel {76} ist genau so ein Fall. Das Verhalten hat zahlreiche Vor- und Nachteile, Sie müssen es aber immer im Hinterkopf bewahren, denn sonst suchen Sie vergeblich nach Fehlern in Ihren Regexen.

{70} und {71} sollten keine Probleme bereiten. Bei {72} fangen aber die Augen zu schwirren an. Außerdem werden ein paar neue Ideen eingeführt. Sie müssen lernen, das zu lesen. Suchen Sie einfach erst mal nach dem R(|) (Aufpassen, wenn ein R(\) davorsteht, dann ist das Zeichen | gemeint.). Links steht nun R(„[^“]+“). Die R(„) am Anfang und Ende matchen das Zeichen „. Dazwischen wird eine Folge von Zeichen (mindestens eines) gematcht, die keine Anführungszeichen enthalten. Für den rechten Teil ist der Regex identisch, nur wird statt „ das einfache Anführungszeichen ‘ gemacht, das hier wegen PHP escapet werden muss.

{75} ist auch so ein unübersichtlich langer Regex. Verfahren Sie hier genauso: Sie müssen die Operatoren in der umgekehrten Reihenfolge ihrer Priorität aufdröseln, also zuerst werden die Alternativen getrennt, dann die Quantifier aussortiert, dann die feststehenden Zeichen und Anker gesucht. `R([[:space:]]+)` zum Beispiel matcht mindestens ein Leerzeichen. Aber auch Tab, Newline und Carriage Return! Das ist wichtig, denn die Syntax von PHP sieht vor, dass man eine Funktion in PHP auch so schreiben kann:

```
function
    hugo
    ($bla)
```

Die Klammern in `R(\(\))` müssen escapet werden, da es Regex-Steuerzeichen sind. Betrachten Sie die rechte Seite. Dort steht `R(\(.+\))`. Sie erinnern sich sicher, dass ich geschrieben habe, Sie sollten den Punkt nur verwenden, wenn es fast egal ist, was matcht. Hier wäre etwas wie `function  hugo())` ein erlaubter Match!

Denn die Regel mit dem ersten und längsten Treffer betrifft wie schon erwähnt auch Quantifier. In Bezug auf {75} heißt das zum Beispiel, dass der Suchtext *`function ())`* vom rechten Teil des Regex gematcht würde, obwohl der linke genauso passen würde.

Gruppen, Zwischenergebnisse, Teilregexe

Gruppen sind etwas völlig Neues, denn sie erweitern die Patternsuche nicht im eigentlichen Sinn. Aber sie sind das, was Regexe erst so richtig interessant macht, denn Sie können nun nicht nur bestimmen, ob ein Regex matcht oder nicht, sondern auch, *was* er matcht! Im Prinzip bilden Gruppen so etwas wie Unterprogramme innerhalb eines Regex und Sie bekommen zusätzlich die Ergebnisse dieser Teilregexe zurückgeliefert.

Beispiel {81} findet zum Beispiel etwas wie *function hugo($bla)*. Das *print_r()* im Testprogramm liefert dabei folgendes Ergebnis:

Beispielumgebung 2

Für die Gruppen benötigen Sie eine neue Beispielumgebung:

```
<?php
$source='Dies ist ein Text der hugo enthält';
$pattern='(.*)';
if(ereg($pattern,$source,$matches)){
        echo „Gefunden $source\n“;
        print_r($matches);
}
?>
```

```
Array
(
  [0] => function hugo($bla)
  [1] => hugo
  [2] => $bla
  [3] =>
  ...
  [9] =>
)
```

Ich nummeriere die Register [0], [1], [2] usw. im Folgenden mit #0, #1, #2 usw. durch. In #0 findet sich immer der gesamte Match. Sie können sich das so vorstellen, dass der Regex selbst in Klammern eingeschlossen ist. Die Klammern werden einfach konsequent von links nach rechts durchgezählt. #1 ist demnach die erste Klammer. Das soll in unserem Fall den Funktionsnamen matchen. Das klappt auch hervorragend, genau wie bei #2, welches die Parameterliste sein soll.

Innerhalb der Klammern gelten ohne Ausnahme alle bisher bekannten Regeln (Klammern sind so etwas wie Unterprogramme). Das ist insofern wichtig, als dass Sie beim Erstellen eines komplexen Regex das Problem in viele kleine Probleme unterteilen können und am Ende eigentlich nur Ihre einzelnen Regexe zusammenbauen müssen. Sie können zum Beispiel Alternationen verwenden {82} oder die Klammern schachteln {84}. Sie müssen dabei beachten, dass sich die Anzahl der Klammern nicht vervielfacht, also die Klammer #1 im Beispiel {83} bleibt die Klammer #1; nachfolgend durch die Wiederholung gefundene Werte überschreiben den Eintrag immer wieder. Der zuletzt gefundene Eintrag bleibt im Register #1 stehen. Es hindert einen natürlich niemand daran, den Regex so wie in {84} zu schreiben und sich dem Problem der Trennung von #1 in einem nachfolgenden Programmteil zuzuwenden.

{85} sollte mittlerweile kein Problem mehr sein. Aber {86}, das sieht sehr lang und kompliziert aus. Wenden Sie die Techniken des Zerteilens des Regex an: Oberste Priorität haben die Klammern. Denken Sie sich die jedoch zunächst einfach als Black Box. Dann sieht der Regex nämlich gleich ganz harmlos aus. Ich bin mir sicher, dass Sie selbst draufkommen werden, was der Regex matcht. Scheuen Sie sich nicht, die Testumgebung zu benutzen und die geklammerten Teilregexe einzeln auszutesten.

Gruppen	
Gruppen werden durch runde Klammern gebildet. Sie können damit auch Teilergebnisse eines Regex einfangen.	
{80} `R((.*))`	Der komplette Suchtext wird eingefangen.
{81} `R(function ([^(]+)\(([^)]+)\))`	Suche nach dem Schlüsseltext `R(function )`, dann mindestens ein Zeichen, das nicht Klammer auf ist, gefolgt von Klammer auf und Klammer zu und innerhalb dieser Klammer fange einen Text aus mindestens einem Zeichen, das nicht Klammer sein darf, ein.
Hinweis: Die Klammern in einem Regex werden von links nach rechts durchgezählt. Der komplette Regex selbst erhält die Nummer #0, die erste Klammer die Nummer #1, die zweite Klammer #2 usw. Die Variable *$matches* ist nach Ausführen des Regex ein Array, welches genau diese Reihenfolge der Gruppen wiedergibt. *$matches[0]* ist immer der komplette Match/Regex. Sie können sich das so vorstellen, als wäre um den Regex selbst noch mal eine Klammer.	
REGEL: POSIX kann maximal zehn Klammern matchen!	
Alternieren in Klammern:	
{82} `([0-9]{2})\.([0-9]{2})\.([0-9]{2}|[0-9]{4})`	Fängt ein Datum (#1:Tag, #2:Monat, #3:Jahr) ein, wobei das Jahr entweder zwei oder vierstellig sein darf.
Wiederholungen von Klammern:	
{83} `([0-9]{2}\.){2}([0-9]{2}|[0-9]{4})`	Wie {82}, liefert jedoch nicht wie erwartet #1:Tag, #2:Monat, #3:Jahr, sondern #1:Monat und #2:Jahr. Die Wiederholung der ersten Klammer „überschreibt" den gefundenen Tag in #1 mit dem nachfolgenden Monat!
Schachteln von Klammern:	
{84} `(([0-9]{2}\.){2})([0-9]{2}|[0-9]{4})`	Wie {83}, liefert jetzt aber #1:**Tag.Monat**., #2:**Monat**, #3:**Jahr**.
{85} `([[:<:]]([[:alnum:]]+)er[[:>:]])`	Matcht Wörter, die auf „er" enden, z.B. *Leiter*, *Radfahrer* usw. Die äußere Klammer #1 matcht jeweils das ganze Wort, die innere Klammer #2 matcht nur `Leit`, `Radfahr` usw.
{86} `R((3[01]|[12][0-9]|0?[1-9])\.((1[012]|0?[1-9])|(Jan|Feb|M[aä]r|Apr|Mai|Ju[nl]|Aug|Sep|Okt|Nov|Dez))\.([0-9]{2}|[0-9]{4}))`	matcht Datum. #1: Tag (1 bis 31), #2: Monat, entweder #3 (1-12) oder #4 (`Jan-Dez`), #5: Jahr (`00`-`99` oder `0000` bis `9999`), matcht z.B. `31.Feb.2002`.
REGEL: Sie sollten zu matchende Klammern-Zeichen immer mit `R(\)` escapen, auch wenn es laut POSIX möglich ist, nicht matchende schließende Klammern ohne Backslash zu schreiben. {87} `R(hugo(boss)?)` Siehe {76}, ist aber schneller	

Tabelle 6: Gruppen, Zwischenergebnisse, Teilregexe

Beispiel: Testen und Konvertieren von Datumsangaben

Generell gilt, dass Datumsangaben mit Regexen nicht auf Korrektheit zu matchen sind. Sie können nur – wie im Beispiel {86} – sehr weit gehen. Auch hier stellt sich wieder die Frage, was schneller ist: Setzen Sie sich eine halbe Stunde hin, um so etwas wie {86} zu fabrizieren, oder programmieren Sie das doch schneller „per Hand", indem Sie wie in {82} etwas matchen, was wie ein Datum aussieht und dann zum Beispiel mit

```
if ( ( $timestamp=strtotime("$matches[3]-$matches[2]-
                                $matches[1]")) === -1)
        echo "Datum ist Fehlerhaft";
```

prüfen, ob das Datum korrekt war. Es kommt wirklich auf den Zweck an.

Ein schöner Zweck, den Sie mit Regexen erreichen können, ist auf verschiedene Datumformate zu testen und diese jeweils korrekt zu konvertieren. Dafür sind Regexe die ideale Wahl. Denn die Funktion *strtotime()* ist nur auf englische oder internationale Daten geeicht. So eine Funktion könnte z.B. so aussehen wie in Listing 1.

Sie sehen schon, mit Datumsangaben und Regexen kann man viele schöne Beispiele machen und man kann immer irgendetwas verbessern – ich komme im zweiten Teil noch mal darauf zurück, wie Sie diese Funktion optimieren könnten. Jedenfalls sollten die Zeiten, in denen der Benutzer gezwungen wurde Tag, Monat und Jahr aus drei Select-Boxen auszuwählen oder sich ganz genau an ein be-

stimmtes Format halten zu müssen, mit der Beherrschung von Regexen ein Ende haben.

Beispiel: Telefonnummern verifizieren

Ein weitaus ergiebigeres Beispiel sind Telefonnummern. Ich nehme an, wir haben irgend ein Eingabeformular gegeben, bei dem das Eingabefeld

```
Telefonnummer:<br>
<input type="text" name="telnr"><br>
++Landeskennziffer (Vorwahl) Nummer-Durchwahl
```

auftaucht. Im Lauf der Zeit stellt sich heraus, dass die User den Hinweis unter dem Eingabefeld nur zu 70 Prozent beachten. Der Rest schreibt das irgendwie und jetzt haben Sie einige tausend Einträge, die eine andere Form von Telefonnummer haben. Ihre Aufgabe ist es nun, die Nummern, die nicht dem Format entsprechen, korrekt umzuformen, oder wenn das Ganze nicht einer Form entspricht, die Sie erkennen können, einen Fehler zu melden. Ausdrücklich ist es nicht Ihre Aufgabe, herauszufinden, ob die Nummer korrekt (=anrufbar) ist. Das ist eine andere Baustelle.

Wo fangen Sie als Entwickler, der alles in gegebener Zeit korrekt abwickeln will, an? Eine gute Idee ist es, sich die entsprechenden Normen zu Telefonnummern anzusehen. Das ist wie bei einem Architekt, der auch nicht Häuser baut, ohne zu wissen, wie fest und tragfägig die Materialien sind, die er verbaut. Man findet Empfehlung Nummer E.123 der ITU (www.itu.int) als „Notation for national and international telephone numbers“ und kann sie für 20 Schweizer Franken runterladen.

Jetzt wissen Sie zum Beispiel, ob die Vorgabe tatsächlich stimmt und was ebenfalls korrekt ist. Der nächste Schritt wäre, sich alle Telefonnummern in der Datenbank anzusehen, die *nicht* in der richtigen Form sind. Das hängt jetzt von der Datenbank ab, ob Sie solche Datensätze direkt finden können, mit MySQL ist das kein Problem, denn „rein zufälligerweise“ benutzt MySQL auch POSIX-Regexe. Folglich sieht eine Query, die prüft, ob eine Telefonnummer dem oben beschriebenen Format entspricht (ich habe es der Übersichtlichkeit wegen etwas laxer gemacht), so aus:

```
SELECT telefonnummer FROM adressen WHERE telefonnummer NOT REGEXP '^\+\+[0-9]{1,2} \([0-9]{2,5}\) [0-9]{3,12}(-[0-9]{1,8})?$'
```

Die letzte Klammer mit dem Fragezeichen dahinter muss ich erklären: Das ist eine Gruppe, die optional matcht, wenn eine Durchwahl da ist.

Mit anderen Datenbanken müssen Sie entweder eine andere Regex lernen oder sich ein PHP-Programm schreiben, welches die Datensätze mit diesem Regex einzeln testet. Wichtig ist aber nur, dass Sie eine Liste mit falsch formatierten Telefonnummern erhalten (Testdaten). Sie haben nun nach ein bis zwei Stunden Arbeit folgende Liste von beispielen gefunden, von denen Sie denken, dass sie beispielhaft für jeweils eine ganze Reihe von ganz ähnlichen falsch formatierten Telefonnummern sind:

- 0123/456789
- 0049 123 456789
- 49 (123) 45 67 89
- ++49 123/456789
- 49123456789
- 0123 45 67 89
- 0049123/456789
- und zahlreiche andere Beispiele; wie gesagt, dem Erfindungsreichtum der User ist erfahrungsgemäß kein Limit gesetzt...

Viele Beispiele lassen sich natürlich überhaupt nicht in die richtige Form überführen, *49123456789* zum Beispiel – was ist die Vorwahl und was der Rest? Oder *0049123/456789*. Wieso nicht? Das *0049* kann doch nur die Landeskennziffer sein? Nein, Landeskennziffern kön-

Listing 1

```
function strtotime_de ($str) {
 # Datum in Form von 01-10/00 (DD.MM.JJJJ)
 if ( ereg('([0-9]{1,2})[^0-9]([0-9]{1,2})[^0-9]
                    ([0-9]{2}|[0-9]{4})',$str,$matches) )
  return(strtotime(„$matches[3]-$matches[2]-
                                    $matches[1]“));
 # Datum in Form von 01. November 2000 oder 1-Nov-00
 elseif ( ereg('([0-9]{1,2})[^[:alnum:]]*
   ([[:alpha:]]{3,9})[^[:alnum:]]+([0-9]{2}|[0-9]{4})',
                                    $str,$matches) ) {
  switch (strtolower($matches[2])) {
  case 'januar' :
  case 'jan':
   $mon=1; break;
  case 'februar' :
  case 'feb' :
   $mon=2; break;
  ... und so weiter bis dezember ...
  default:
   return(-1);
  }
  return(strtotime(„$matches[3]-$mon-$matches[1]“));
 }
 elseif ( ... und so weiter bis alle GNU Date-Input-Formate
                                    umgesetzt sind ...
 }
 return(-1);
}
```

Listing 2

```
function reformat_phonenr($str) {
 # 0123/456789, 0123 45 67 89 usw.
 if ( ereg('^[[:space:]]*([^0-9]*0([1-9][0-9]+))[^0-9]+(([0-9]+[[:space:]]*)+)(-([0-9]{1,8}))?[[:space:]]*$',$str,
                                                                    $matches) ) {
   $matches[3]=ereg_replace('[[:space:]]','',$matches[3]);
   return(„++49 ($matches[2]) $matches[3]$matches[5]“);
 }
 # 0049 123 456789, 49 (123) 45 67 89, ++49 123/456789 usw.
 if ( ereg('^[[:space:]]*((00|\+\+)?([1-9][0-9]?))[^0-9]+([1-9][0-9]+)[^0-9]+(([0-9]+[[:space:]]*)+)
                                        (-([0-9]{1,8}))?[[:space:]]*$',$str,$matches) ) {
  $matches[5]=ereg_replace('[[:space:]]','',$matches[5]);
  return("++$matches[3] ($matches[4]) $matches[5]$matches[7]“);
 }
 return(false);
}
```

nen auch Einstellig sein (1 ist z.B. USA. Wissen um Normen bewahrt vor groben Fehlern). Aber die meisten anderen lassen sich durchaus korrekt konvertieren. Sie sollten zur Sicherheit auch eine korrekte Nummer in die Beispiele miteinbeziehen, etwa *++49 (123) 456789*, denn falls die Prüfung auf Korrektheit der Nummern einen Fehler hat, dann erhält ihre Funktion womöglich eine korrekte Telefonnummer und es macht Sinn, dass diese ihn ohne zu murren weiterverarbeitet. Als Nächstes müssen wir Regexe finden, die auf möglichst viele Beispiele matchen, ohne versehentlich ein anderes Beispiel zu finden.

Wie ich schon erwähnte, sollten Sie bei so komplexen Aufgaben das Problem in kleinere Probleme aufteilen. Hier bietet sich die Trennung von Landeskennziffer-Vorwahl und Nummer-Durchwahl an. Zum Matchen des Nummer-Durchwahl-Teils können Sie zum Beispiel den Regex `R((([0-9]+[[:space:]]*)+)(-[0-9]{1,8})?[[:space:]]*$)` nehmen. Das matcht zum Beispiel auf *45 67 89-987*, in #1 steht dann `45 67 89` und in #3 steht `-987`, oder die Nummer *4567 89*, dann stünde in #1 `4567 89`. Mit einem beherzten

```
$matches[1]=ereg_replace('[[:space:]]',"",$matches[1]);
```

entfernen Sie anschließend die überflüssigen Leerzeichen. So könnten Sie anschließend zur Vorwahl alle auftauchenden Nummern matchen. Dazu müssen Sie aber erst herausfinden, was die Vorwahl ist.

Mit etwas Überlegung und Probieren bekommen Sie mit `R(^[[:space:]]*(0([1-9][0-9]+))[^0-9]+)` den Fall mit der einfachen Vorwahl ohne Landeskennziffer heraus (wenn keine Landeskennziffer gefunden wird, dann wird 49 vorausgesetzt); in #2 findet sich anschließend die Nummer ohne führende Null. `R(^[[:space:]]*((00|\+\+)?([1-9 ][0-9]?))[^0-9]+(([1-9][0-9]+)) [^0-9]+)` fände demnach die Beispiele mit Landeskennziffer, wobei in #3 die Landeskennziffer und in #4 die Vorwahlnummer stünde. Kombinieren Sie nun alle Regexe, dann erhalten Sie die in Listing 2 dargestellte Funktion.

Sicher berücksichtigt das nicht alle Fälle. Beim Testlauf findet man zum Beispiel eine Nummer wie *(0123) 456789*. Entsprechende Debug-Ausgaben decken den Fehler auf und man muss die Funktion um eine neue Regel erweitern oder einen Regex entsprechend anpassen.

Auf der CD befindet sich ein kleines Testprogramm, mit dem Sie üben können.

Sicher gibt es optimalere Lösungen, wir prüfen zum Beispiel nicht die Länge der Nummern. Vielleicht schaffen Sie es ja, diese zwei Regexe zu einem zu verknüpfen oder auf das Ersetzen der Leerzeichen zu verzichten. Als Schreiber dieses Artikels wäre ich natürlich sehr stolz, aber als Entwickler sollten Sie wissen, wann Sie aufhören sollten, und wann sich Kosten und Nutzen die Waage halten. Regexe sind einfach ein tolles Werkzeug, perfekter Code kommt damit nicht von allein heraus.

Ausblick

Wenn Sie alle Beispiele ausprobiert und verstanden haben, sollten Sie jetzt genügend Übung haben, um komplexere Probleme selbst zu lösen. Die Alltagsprobleme werden Sie von selbst einholen: Beachten Sie, dass fast jede Regex-Maschine anders ist, andere Tücken und Ausnahmen hat.

Verzweifeln Sie nicht, es ist erstaunlich viel Verwandtschaft vorhanden und meist benötigen Sie gar nicht die Tricks, bei denen die Unterschiede zum Tragen kommen – das Beispiel mit den Telefonnummern funktioniert zum Beispiel mit PCRE genau so.

Ich habe einige wichtige Aspekte von POSIX-Regexen im ersten Teil nicht behandelt. Das wären Backtracking, Rückwärtsreferenzen, Suchen und Ersetzen, sowie Aufsplitten. Nicht nur, weil der zuständige Redakteur sonst im Kreis gesprungen wäre, sondern auch, weil wir das im zweiten Teil machen. Neben vielen weiteren Beispielen geht es dort um PCRE und dessen Abgründe.

Alexander Aulbach ist Diplom Informatiker (FH) und seit 1996 selbstständiger Web-Applikations-Entwickler. Zurzeit arbeitet er hauptsächlich für die fidion GmbH. Kontakt: aulbach@fidion.de.

Literatur & Links

[1] „Reguläre Ausdrücke“, Jeff Friedl, O’Reilly, Deutsche Ausgabe, ISBN 3-930673-62-2

(Anmerkung: Die Bibel der Regex)

POSIX Regex

[2] regex(5)-man-page:
www.cee.odu.edu/cgi-bin/cee_man-cgi?regex+5

[3] posix-regex-functions:
sunland.gsfc.nasa.gov/ info/regex/POSIX_Regex_Functions.html

[4] RE_FORMAT(7):
www.tac.eu.org/cgi-bin/man-cgi?re_format+7

[5] IEEE1003.2 (IEEE-Standards kaufen):
standards.ieee.org/catalog/olis/posix.html

[6] MySQL Regexp (mit p):
www.mysql.com/documentation/mysql/bychapter/manual_Regexp.html

[7] Fehlermeldungen in POSIX:
(PHP gibt diese Meldungen aus)
sunland.gsfc.nasa.gov/info/regex/POSIX_Regular_Expression_Compiling.html

PCRE

[8] Homepage von PCRE:
www.pcre.org/

Sonstige

[9] JavaScript RegExp:
developer.netscape.com/docs/manual/js/core/jsref/regexp.htm

Zeichensätze

[10] ASCII-Charsets:
czyborra.com/charsets/iso646.html

[11] und dazugehörige Controll-Codes:
www.cs.tut.fi/~jkorpela/chars/c0.html

Einführungen und Übersichten

[12] Einführung in Reguläre Ausdrücke:
www.rrz.uni-hamburg.de/RRZ/W.Wiedl/Skripte/CGI-Perl/Regulaere_Ausdruecke/

[13] Übersicht über alle möglichen Regexe (Kathryn A. Hargreaves, Karl Berry):
www.cs.utah.edu/dept/old/texinfo/regex/regex_toc.html

[14] ctype(3)-man-page (libc ctype-Macros):
www.tac.eu.org/cgi-bin/man-cgi?ctype+3

[15] Theoretische Informatik-Skripten zum Beispiel unter:
math.uni-heidelberg.de/logic/skripten/theoinf/

Brücke an Maschinenraum

von Stephan Schmidt und Gerd Schaufelberger

Interprozesskommunikation mit PHP4 und Shared Memory

PHP Skripte werden vom Betriebssystem genau wie alle anderen Prozesse ausgeführt und laufen somit in einem gekapselten Speicherbereich. Dieses Verhalten ähnelt einer „Sandbox", in der man tun und lassen kann, was man will, ohne auf die Welt außerhalb des Sandkastens Rücksicht nehmen zu müssen.

Dadurch wird ein hohes Maß an Sicherheit gewährleistet. Andererseits wird damit die Kommunikation zwischen den parallel laufenden Prozessen unterbunden, da jeder Prozess in der Illusion lebt, den Server für sich alleine zu haben. Was jedoch, wenn die Applikation explizit verlangt, dass mehrere gleichzeitig laufende Prozesse Daten austauschen?

Interprozesskommunikation ist etwas, was man normalerweise nicht mit Webapplikationen in Verbindung bringt. Es bedeutet, dass mehrere gleichzeitig laufende Prozesse untereinander Daten austauschen. Da PHP hauptsächlich dazu verwendet wird, HTML Seiten dynamisch zu erstellen und an den Client auszuliefern, ist die Lebensdauer eines PHP Prozesses nicht besonders hoch, es kommt also kaum zu gleichzeitig laufenden Prozessen. Es gibt jedoch besondere PHP Anwendungen, bei denen ein Skript erst dann beendet wird, wenn auch die gesamte Anwendung geschlossen werden soll. Ein Beispiel hierfür wäre ein Web-Chat, bei dem das Skript, welches die eingehenden Nachrichten ausgibt, kontinuierlich im Hintergrund läuft, während ein anderes Skript auf die klassische Weise aufgerufen wird, um per HTML Formular eingegebene Nachrichten zu versenden. Um diese neuen Nachrichten zu erhalten, wird nun doch eine Möglichkeit der Interprozesskommunikation benötigt.

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

Vom Gästebuch zum Chat

Diese Arbeitsweise der getrennten Ein- und Ausgabe von Daten ist auch schon von anderen Webapplikationen bekannt. Sie wird zum Beispiel in einer der einfachsten PHP Applikationen, einem Gästebuch, eingesetzt. User A trägt Daten in ein HTML Formular ein und schickt das Formular ab. Die Daten werden an den Server gesandt und dort gespeichert, meist in einer Textdatei oder Datenbank. User B sieht sich das Gästebuch später an und bekommt dabei auch den Eintrag von User A zu sehen.

Es kommunizieren also auch hier zwei Prozesse miteinander: ein Prozess nimmt Daten entgegen, ein anderer gibt sie wieder aus. Das Kommunikationsmedium ist die Textdatei oder Datenbank. Der große Unterschied zum Chat liegt in der zeitlichen Verzögerung. Während beim Gästebuch Stunden oder Tage zwischen Eingabe und Ausgabe liegen können, muss die Ausgabe bei einem Chat unmittelbar nach der Eingabe erfolgen, damit der Benutzer den Eindruck einer realen Kommunikation erhält. Konkret bedeutet dies, dass beim Gästebuch der Benutzer eine URL anfordert, die ihm alle aktuellen Gästebuch-Einträge auflistet. Erfolgt danach ein weiterer Eintrag, wird die Liste erst nach einem Reload der Seite aktualisiert. Eine manuelle Aktualisierung der Daten ist für einen Chat nicht schnell genug, es entsteht nicht der Eindruck einer unmittelbaren Kommunikation. Um zu verhindern, dass das Skript neu geladen werden muss, wird es einfach nicht beendet und gibt somit auch die Kontrolle über die Ausgabe nicht ab.

Kommunikationsmedien

Das Ausgabeskript läuft nun ständig in einer Schleife und erwartet Nachrichten, um sie als HTML wieder auszugeben. Die zwei gleichzeitig auf dem Server laufenden Prozesse müssen nun miteinander kommunizieren. Es bieten sich verschiedene Medien für diese Kommunikation an: wie bereits angesprochen, können Daten in Datenbanken oder Textdateien gespeichert und danach von einem anderen Prozess wieder ausgelesen werden. Diese Kommunikationsmedien kommen auf den ersten Blick natürlich auch für einen Chat in Betracht, soll dieser Chat jedoch auf hoch frequen-

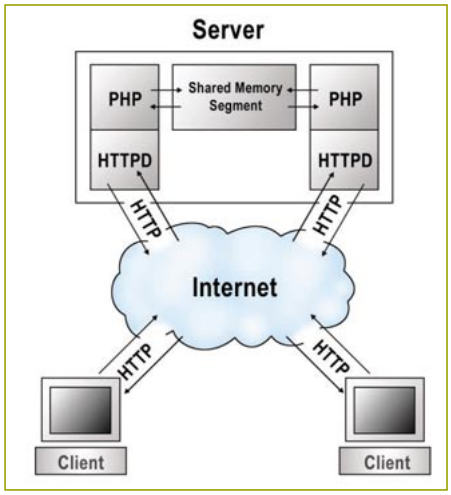


Abb. 1: Interprozesskommunikation mit Shared Memory

tierten Webseiten zum Einsatz kommen, wird er schnell an seine Grenzen stoßen. Um das Gefühl einer unmittelbaren Kommunikation zu vermitteln, sollte mindestens einmal pro Sekunde überprüft werden, ob neue Nachrichten vorhanden sind, und diese sollten auch ausgegeben werden. Stellt man sich nun vor, dass 20 PHP Skripte ein- bis zweimal pro Sekunde versuchen, die selbe Textdatei zu öffnen und Daten daraus zu lesen, wird einem schnell klar, warum Textdateien für Chats kein geeignetes Kommunikationsmedium darstellen: die Zugriffszeiten auf das Medium sind zu hoch. Da eine Datenbankabfrage auch einen Zugriff auf die Festplatte bedeutet, scheiden Datenbanken ebenfalls als Kommunikationsmedium für einen Chat aus.

Shared Memory

Es wird also ein Medium benötigt, das schnelleren Zugriff als eine Festplatte ermöglicht. Hier bietet sich die Verwendung des Hauptspeichers an. Mit Hilfe der Shared Memory Funktionen von PHP ist es unter Unix Systemen möglich, ein Speichersegment anzulegen, auf das alle laufenden Prozesse zugreifen können. In dieses Speichersegment können z.B. globale Systemvariablen geschrieben werden. Der Zugriff auf ein solches Speichersegment von PHP ist mit wenigen Zeilen Code zu realisieren:

```
$shmId = shm_attach( 2002, 1024 );
if( !$shmId )
  die( "Kein Zugriff auf Shared Memory Segment möglich." );
```

Der Befehl *shm_attach()* erwartet als ersten Parameter einen Integer Wert als eindeutige Kennung des Speichersegments. Diese Kennung muss in allen Skripten, die auf das Shared Memory Segment zugreifen wollen, identisch sein. Hier liegt auch ein Sicherheitsproblem des Shared Memory. Jedes Programm (nicht nur PHP Skripte), das ein Speichersegment mit der verwendeten Kennung anbindet, kann auf die darin gespeicherten Daten zugreifen und diese verändern. Um diese Sicherheitslücke wenigstens teilweise zu schließen, kann als dritter Wert optional eine Zugriffsberechtigung gemäß dem Unix Dateisystem übergeben werden. Der zweite Parameter des Funktionsaufrufs definiert die Größe des angelegten Speichersegments in Byte, in diesem Fall also ein KByte.

Beim ersten Aufruf der Funktion mit einer Kennung wird das Speichersegment angelegt (und enthält noch keine Daten), bei jedem weiteren Aufruf wird kein neues Segment erstellt, sondern der Zugriff auf das zuvor erstellte Segment ermöglicht. In unserem Beispiel des Web-Chats legt der erste Benutzer, der den Chat betritt, das Speichersegment an und jeder weitere Besucher bindet das Segment an das Skript an. Die Funktion gibt eine eindeutige ID für die Anbindung zurück, oder *false,* falls die Anbindung fehlgeschlagen ist.

Lesen und Schreiben von Werten

Nachdem das Shared Memory Segment angelegt wurde, sollen natürlich auch Daten darin gespeichert werden. Prinzipiell können alle Arten von Daten im Shared Memory gespeichert werden, man sollte jedoch darauf achten, dass die Größe eines Speichersegments vom Betriebssystem begrenzt ist. Es empfiehlt sich also nicht, Grafikdateien oder Ähnliches im Shared Memory abzulegen. Daten werden beim Speichern im Shared Memory serialisiert. Möchte man wissen, wie viel Speicher eine Variable im Shared Memory Segment belegen wird, so kann folgender PHP Code verwendet werden:

```
strlen( serialize( $variable ) );
```

Zum Speichern wird die Funktion *shm_put_var()* verwendet:

```
shm_put_var( $shmId, 1, "Interprozesskommunikation mit PHP" );
```

Diese Anweisung speichert den String *„Interprozesskommunikation mit PHP"* unter der Variablenkennung *1* im zuvor angebundenen Speichersegment. War das Schreiben erfolgreich, gibt die Funktion *true* zurück, war nicht mehr genügend Speicher im Segment frei, ist der Rückgabewert *false*.

Leider können als Variablennamen im Shared Memory nur Integer Zahlen, wie hier z.B. *1*, verwendet werden. Um die Les- und Wartbarkeit eines Skripts zu erhöhen, empfiehlt sich der Einsatz von Konstanten als Variablenkennung.

```
$value = shm_get_var( $shmId, 1 );
```

liefert den Wert, der der Variablen *1* im Shared Memory zugewiesen wurde, zurück. Die Daten bleiben trotzdem im Speicher erhalten. Möchte man Daten endgültig aus dem Speichersegment entfernen, wird die Funktion

```
shm_remove_var( 1 );
```

verwendet. Dabei wird der von der Variable belegte Speicher wieder freigegeben.

Beenden der Anbindung

Bei Beendigung des PHP Skripts wird die Anbindung des Shared Memory Segments automatisch beendet. Möchte man die Anbindung vorzeitig beenden, wird die Funktion

```
shm_detach( $shmId );
```

verwendet, die als einzigen Parameter die von *shm_attach()* zurückgegebene ID erwartet. Das Speichersegment und die darin enthaltenen Daten bleiben erhalten und können nach erneuter Anbindung wieder genutzt werden. Sollen das Segment und die enthaltenen Daten endgültig gelöscht werden, muss die Funktion

```
shm_remove( 2002 );
```

verwendet werden. Hierbei muss jedoch die Kennung des Segments als Parameter übergeben werden und nicht die ID, die

zurückgeliefert wurde. Die Funktion gibt *true* zurück, wenn das Speichersegment gelöscht werden konnte, ansonsten *false*.

Konkurrierende Zugriffe

Nachdem nun mehreren Prozessen uneingeschränkter Zugriff auf ein Speichersegment und die darin enthaltenen Daten gewährt wurde, kann es zu konkurrierenden Zugriffen kommen. Das bedeutet, dass z.B. ein Prozess Daten ausliest, die gerade von einem anderen Prozess geändert werden. Folglich erhält der lesende Prozess unvollständige oder korrupte Daten.

Es muss also ein Mechanismus implementiert werden, der dem File Locking ähnelt und sicherstellt, dass nicht mehrere Prozesse gleichzeitig die selben Daten modifizieren. Bevor ein Prozess Zugriff auf die Daten im Shared Memory erhält, muss überprüft werden, ob bereits ein anderer Prozess die Daten modifiziert. Ist dies der Fall, so muss die Ausführung des Skripts angehalten werden, bis die Ressource wieder frei ist. Danach setzt der Prozess ein Flag, mit dem allen anderen Prozessen signalisiert wird, dass die Daten jetzt bearbeitet werden. Erst danach darf der Prozess wirklich auf das Speichersegment zugreifen. Nach Beendigung des Zugriffs gibt er das Speichersegment durch Zurücksetzen des Flags wieder zum öffentlichen Zugriff frei.

Damit zwischen Überprüfung und Setzen des Flags kein anderer Prozess das Speichersegment blockieren kann, ist die Geschwindigkeit des Locking-Mechanismus von höchster Bedeutung. De facto müssen beiden Operationen innerhalb eines Prozessortaktzyklus erfolgen. Deshalb stellen Multitasking-Betriebssysteme Semaphore Funktionen zur Verfügung, die den exklusiven Zugriff auf eine Systemressource ermöglichen. Auch in PHP sind Semaphore Funktionen implementiert und mit den Shared Memory Funktionen in einer Extension zusammengefasst.

Semaphore Funktionen

Analog zu den Shared Memory Funktionen wird jede Semaphore durch eine eindeutige Kennung identifiziert, die in allen Prozessen identisch sein muss. Um einen Zugriff auf eine Semaphore zu ermöglichen, wird zunächst mit

```
$semId = sem_get( 1974 );
```

eine Semaphore ID geholt, die später von weiteren Semaphore Funktionen verwendet wird. Als erster Parameter wird eine eindeutige Kennung der Semaphore erwartet. Möchte man nicht nur einem Prozess den Zugriff auf die Semaphore ermöglichen, kann als zweiter Parameter angegeben werden, wie viele Prozesse gleichzeitig die Semaphore anfordern können. Als dritter, optionaler Parameter können wieder die Zugriffsrechte festgelegt werden. Soll nun eine Semaphore angefordert werden, so wird die Funktion

```
sem_acquire( $semId );
```

verwendet, die als einzigen Parameter die von *sem_get()* zurückgegebene ID erwartet. Diese Funktion blockiert die Ausführung des Skripts so lange, bis die Anforderung der Semaphore durchgeführt werden kann. Bei Beendigung des Skripts erfolgt eine automatische Freigabe der Semaphore und es wird eine Warnung ausgegeben, dass die Semaphore nicht freigegeben wurde. Um sie manuell wieder freizugeben, wird

```
sem_release( $semId );
```

verwendet.

Wie der Zugriff auf ein Shared Memory Segment mit Locking ablaufen sollte, verdeutlicht Listing 1.

Kapselung in einer Klasse

Wie Listing 1 zeigt, sind für jeden Zugriff mehrere Funktionsaufrufe nötig: Zuerst wird die Semaphore angefordert, danach das Speichersegment modifiziert und dann die Semaphore wieder freigegeben. Dabei muss jedes Mal die korrekte ID der Semaphore oder des Shared Memory Segments übergeben werden. Um dies zu vereinfachen, kann eine Klasse verwendet werden, die alle Shared Memory und Semaphore Funktionen kapselt und damit den Zugriff erheblich erleichtert. Mit der Klasse *patSHMC* auf der beigefügten CD-ROM wird Ihnen solch ein Hilfsmittel zur Verfügung gestellt. Die Verwendung der Klasse verdeutlicht Listing 2: anstatt mehrere Variablen zu verwalten, lässt man dies von einer Klasse erledigen. Zusätzlich lässt sich so auch ein universelles Fehlermanagement implementieren.

Architektur des Messengers

Nachdem Sie mit allen Funktionen, die den Umgang mit Shared Memory ermöglichen, vertraut sind, geht es im letzten Teil um die Architektur einer Applikation, die Shared

Listing 1

```
$shmK      =          2002;
$semKey    =          2003;

// Speichersegment anbinden
$shmId  = shm_attach( $shmKey, 1024 );

// Semaphore Handle holen
$semId     = sem_get( $semKey );

// Semaphore anfordern
sem_acquire( $semId );

// Wert speichern
shm_put_var( $shmId, 1, „Interprozesskommunikation
                                        mit PHP“ );

// Wert zurückholen
echo shm_get_var( $shmId, 1 );

// Semaphore freigeben
sem_release( $semId );

// Anbindung beenden
shm_detach( $shmId );
```

Listing 2

```
require_once(“include/patSHMC.php“);

$shmKey    = 2002;
$semKey    = 2003;

// Neue Instanz erzeugen
$shm       = new  patSHMC( $shmKey, 50000,
                                        $semKey );

// Semaphore anfordern (Speichersegment locken)
$shm->lock( patEXCLUSIVE_LOCK );

// Wert speichern
$shm->put_var( 1, “Interprozesskommunikation mit
                                        PHP“ );

// Wert zurückholen
echo $shm->get_var( 1 );

// Semaphore freigeben
$shm->lock( patRELEASE_LOCK );

// Anbindung beenden
$shm->detach();
```

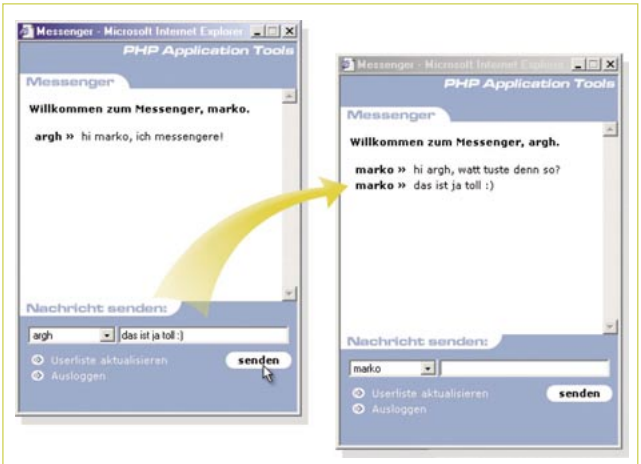


Abb. 2: Screenshot der Messenger Applikation

Memory nutzt, um Daten zwischen gleichzeitig ablaufenden Prozessen auszutauschen. Als Beispiel soll ein Messenger programmiert werden, der es ermöglicht, einem anderen eingeloggten Benutzer eine private Nachricht zu schicken. Um die Applikation möglichst simpel zu halten, werden weder Authentifizierung noch Sessions verwendet. Das Hauptaugenmerk liegt auf der Kommunikation der verschiedenen Prozesse, zu den anderen Themen finden Sie Informationen im *PHP Special* oder folgenden Ausgaben des *PHP Magazins*.

Der Programmablauf ist äußerst simpel: Nachdem der Benutzer seinen Namen eingegeben hat, werden zwei Frames angezeigt, einer, in dem alle Nachrichten an den Benutzer angezeigt werden, und einer, in dem er selbst einen Empfänger wählen und eine Nachricht eingeben kann, die dann an den Empfänger geschickt wird.

Dazu müssen im Shared Memory verschiedene Daten gespeichert werden. Zum einen eine Liste aller angemeldeten Benutzer und zum anderen alle eingehenden Nachrichten pro Benutzer. Für beides empfiehlt es sich, Arrays zu verwenden. Meldet sich ein Benutzer am Messenger an, wird sein Benutzername an das Array mit allen angemeldeten Benutzern angehängt. Dadurch wird automatisch eine für diese Session gültige User ID erzeugt: der Index des Arrays, an dessen Position der Benutzername steht. Für die Liste der Nachrichten ist auch ein Array optimal geeignet, wobei in diesem Array für jeden Benutzer ein neues Array mit den Nachrichten für den Benutzer gespeichert wird. Dadurch ist es einfach möglich, mit

```
array_push( $messages[$uid], $newMessage );
```

neue Nachrichten einzufügen.

Nach dem Aufbau des Framesets befindet sich der Ausgabeframe in einer Endlosschleife, in der er jede halbe Sekunde Daten aus dem Shared Memory liest und überprüft, ob im Array mit seinen eingehenden Nachrichten Daten vorhanden sind. Ist dies der Fall, so werden diese mit Hilfe von Templates formatiert und per HTTP an den Browser gesendet. Danach läuft die Endlosschleife weiter.

Soll eine Nachricht an einen Benutzer geschickt werden, so wird im zweiten Frame ein PHP Skript aufgerufen, dem drei Variablen übergeben werden: Der Name des Absenders, die User ID des Empfängers und der Text der Nachricht. Das PHP Skript holt das Array, in dem Nachrichten gespeichert werden, aus dem Shared Memory, hängt die neue Nachricht an das Array mit Nachrichten für den Empfänger an und schreibt das Array danach wieder zurück. Meldet sich der Benutzer wieder ab, so wird er einfach aus der im Shared Memory gespeicherten Liste entfernt.

Ein lauffähiges Beispiel des Messengers, das jedoch nicht in Produktionsumgebungen eingesetzt werden sollte, finden Sie auf der beigefügten CD-ROM.

Ausblick

Die einfache Nachrichtenübermittlung ist nur ein simples Beispiel dafür, was mit Shared Memory möglich ist. Durch die Serialisierung der Daten lassen sich auch Objekte in ihrem aktuellen Zustand im Speicher ablegen und zu einem späteren Zeitpunkt von einem anderen Prozess wieder „aufwecken". Genauso kann Shared Memory zum Speichern von PHP Sessiondaten verwendet werden, ein Container hierfür ist schon vorhanden. Denkbar wäre auch, dass ein Prozess einen anderen durch Funktionsaufrufe, die über Shared Memory transportiert werden, steuert, was eine Art „Remote Procedure Call" darstellt.

Fazit

Wie das Beispiel des Messengers zeigt, ist Interprozesskommunikation in PHP möglich, man sollte jedoch beim Einsatz von Shared Memory genau abwägen, ob dies die geeignete Lösung für das Problem darstellt. Es empfiehlt sich z.B. nicht, sensitive Daten wie Passwörter im Shared Memory zu speichern, genauso wenig wie Daten, die über einen längeren Zeitraum erhalten bleiben sollen, da alle Daten im Shared Memory nach dem nächsten Neustart des Systems nicht mehr zur Verfügung stehen.

Stephan Schmidt und Gerd Schaufelberger sind Web Application Developer bei der Metrix Internet Design GmbH (www.metrix.de) in Karlsruhe und Gründungsmitglieder der PHP Application Tools (www.php-tools.de).

Tipps

Beachten Sie bitte folgende Punkte beim Entwickeln eines eigenen Chats:

- *set_time_limit(0)*; damit das Skript nicht abbricht.
- *flush()* nach jeder Ausgabe (explizites Senden an den Client).
- erhält ein Browser über längere Zeit keine Daten, meldet er einen Timeout, also sollten Sie regelmäßig Daten senden (z.B: HTML Kommentare).
- *register_shutdown_function()*, um sicherzustellen, dass der Benutzer korrekt abgemeldet wird.

Links

- Semaphore und Shared Memory Funktionen: www.php3.de/manual/de/ref.sem.php
- Klasse *patSHMC*: www.php-tools.de
- PHP Optionen und Informationen: www.php3.de/manual/de/ref.info.php
- Funktionen zur Ausgabesteuerung: www.php3.de/manual/de/ref.outcontrol.php

Denn sie wissen nicht...

von Till Gerken

Dynamische Funktionen, Callbacks und Plugins

Dynamische Funktionen sind eine sehr elegante und extrem effiziente Methode, komplexe Programmabläufe und logische Strukturen übersichtlich abzubilden. Sie können selbst in kleinen Programmen schon ein nützliches Helferlein sein, lästige verschachtelte *if()*- und *switch()*-Blöcke ersetzen und den Aufbau einer modularen Programmstruktur erleichtern.

Einführung

Dynamische Funktionen haben ihren Ursprung in der Assemblerprogrammierung. Für eingeschworene PHP-Programmierer gehören sie damit praktisch zu „Vorkriegstechniken“, dennoch sind sie deswegen nicht schlecht und gerade auch in PHP sehr sinnvoll. Von einer dynamischen Funktion bzw. einem dynamischen Funktionsaufruf spricht man dann, wenn eine Funktion aufgerufen wird, die man eigentlich gar nicht kennt. Für diejenigen, die mit dynamischen Funktionen noch nicht in Berührung gekommen sind, mag dies zunächst ein Widerspruch sein.

Angenommen, Sie möchten den Bootloader von Linux um ein grafisches Interface erweitern. Nun ist beim Bootvorgang der Rechner aber praktisch vollkommen leer, d.h. es gibt keine Bibliothek, die Ihnen das Darstellen von Grafiken erlaubt und es gibt auch noch kein Betriebssystem, was Sie dabei unterstützen würde. Wie ist es aber dann möglich, die passende Auflösung einzustellen und ein Menü zu zeichnen? Für solche Dinge ist auf der Hauptplatine des Rechners ein Chip mit dem BIOS (Basic Input/Output System) vorhanden, welches speziell für die Hardware angepassten Code enthält, um grundlegende Zugriffe auf die Systemkomponenten zu erlauben. Dieses BIOS legt an einer bestimmten Stelle im Speicher des Computers eine Tabelle ab, in der alle Adressen der einzelnen BIOS-Funktionen abgelegt sind. Möchten Sie nun die Auflösung ändern, müssen Sie nur die Adresse der entsprechenden Funktion aus der Tabelle laden und dorthin springen. Dies ist praktisch ein Umgang mit einer Blackbox: Sie wissen zwar, was gemacht wird, aber nicht, wie es gemacht wird. Da Sie also eine Funktion aufrufen, die Sie nur indirekt über Ihre Adresse in einer Tabelle ansprechen, ist dies ein dynamischer Funktionsaufruf.

Hier zeigt sich auch schon das Hauptmerkmal aller dynamischen Funktionsaufrufe: sie sind immer indirekt. Die Funktionen werden ausschließlich über Platzhalter und nie direkt aufgerufen. In Hochsprachen ist dieser Platzhalter dann anstelle des eigentlichen Funktionsnamens immer eine Variable.

Die einfachste Anwendung

In PHP lassen sich dynamische Funktionsaufrufe wie folgt gestalten (dieses Beispiel ist auch auf der Heft-CD enthalten):

```
function hello()
{
        print("Hello world!\n");
}

$hello_func = "hello";

$hello_func();
```

Zunächst wird die Funktion *hello()* implementiert. Danach wird einer Variablen der Funktionsname als String zugewiesen. In der letzten Zeile ist schließlich der interessanteste Teil des Programms: die Variable wird zum Aufruf der vorher zugewiesenen Funktion benutzt. PHP erlaubt also die Benutzung von Stringvariablen anstelle von Funktionsnamen – man muss nur die eigentliche Funktion durch eine beliebige Variable ersetzen, die den Namen der aufzurufenden Funktion enthält.

Wo ist nun also der Sinn, eine Funktion nicht direkt über ihren Namen, sondern über eine Variable, die nur ihren Namen

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

enthält, anzusprechen? Schließlich läuft beides auf dasselbe hinaus. Der Sinn besteht darin, dass der Funktionsname, wenn er sich in einer Variablen befindet, weitergegeben werden kann, beispielsweise über Aufrufparameter an eine andere Funktion. So können nicht nur einzelne Funktionsnamen, sondern ganze Modulinterfaces, bestehend aus mehreren Funktionsnamen, an andere Programmteile exportiert werden, ohne dass diese Programmteile die genauen Funktionsnamen kennen müssen. Da PHP auch variable Funktionsargumente unterstützt, müssen diese Programmteile im Extremfall noch nicht einmal die Aufrufsyntax der exportierten Funktionen kennen. Würde man den Faden noch weiter spinnen, könnte man Module konstruieren, die die Namen und Aufrufsyntax ihrer Funktionen exportieren (z.B. durch eine XML-Beschreibung) und die aufrufenden Programmteile könnten dann aus dieser Information wiederum dynamisch den Aufrufcode generieren. Die einzelnen Programmteile müssten also erst „lernen", miteinander umzugehen.

Vergleiche optimieren

Zuerst jedoch zu den einfacheren Versionen dynamischer Funktionsaufrufe und ihrem Nutzen. Das reine Zuweisen eines Funktionsnamen zu einer Variablen wäre noch nichts Besonderes, wenn die Variable im weiteren Programmablauf nicht sinnvoll eingesetzt werden würde. Eine sehr nützliche Eigenschaft von dynamischen Funktionsaufrufen ist die Möglichkeit, häufige, gleichförmige Vergleiche zu optimieren. Benutzt man bei einer Webapplikation beispielsweise zwei Bildschirmtypen, einen für normale User mit einer durchschnittlichen Anzahl von Aktionen und Optionen und einen für Experten, welcher auch weitergehende Möglichkeiten bietet, könnte man beim Rendering oft auf folgendes Konstrukt stoßen:

```
if($is_poweruser)
{
 render_advanced_section(HEAD_OPTIONS);
}
else
{
 render_normal_section(HEAD_OPTIONS);
}

....

if($is_poweruser)
{
 render_advanced_section(MAIN_OPTIONS);
}
else
{
 render_normal_section(MAIN_OPTIONS);
}
```

Dadurch, dass beide Bildschirmtypen gemeinsame Elemente enthalten, muss bei jedem unterschiedlichen Element von neuem entschieden werden, welche Form der Darstellung gewählt werden soll. Ähnliche Probleme könnten auch bei einem Shop auftreten, der einmal Bestellmöglichkeiten für Kunden aus Europa und einmal für Kunden aus den übrigen Ländern enthält. Natürlich gibt es die Möglichkeit, den kompletten Code in zwei Kopien zu pflegen, aber dies ist unsauber und erschwert die Wartung erheblich, vor allen Dingen, wenn es mehr als nur zwei Möglichkeiten gibt – was wäre, wenn man zwischen Kunden aus Europa, den USA und dem Rest der Welt unterscheiden muss? Dann eröffnet eine Filiale in Kanada, die dann natürlich die kanadischen Kunden getrennt behandeln soll usw. usf.

Dynamisches Rendern

Abhilfe schaffen die dynamischen Funktionsaufrufe in einer überraschend klaren Art und Weise: zu Beginn des Programms wird festgestellt, welches Interface ausgegeben werden soll, z.B. so:

```
if($is_poweruser)
{
 $render_function = "render_advanced_section";
}
else
{
 $render_function = "render_normal_section";
}
```

Anschließend wird das gesamte Rendering nur noch über diese Funktion ausgeführt:

```
$render_function(HEAD_OPTIONS);
....
$render_function(MAIN_OPTIONS);
```

Diese Art der Programmierung hält den Code sauber und überschaubar, als Zusatzeffekt werden alle *if()*-Statements praktisch „wegoptimiert". Greift man zur Wurzel des Renderings, lässt sich hier noch eine viel tiefgreifendere Veränderung durchführen: mit dynamischen Funktionen ist es sehr bequem, das komplette Rendering zu abstrahieren. Viele Seiten bieten die Möglichkeit, ihre Inhalte nicht nur über HTTP abzurufen, sondern sie auch in einer Druckversion herunterzuladen oder von einem Handy mittels WAP anzufordern. Während sich eine Klartext-Druckversion noch einfach mit Hilfe eines modifizierten Templates realisieren lässt, wird es bei PDF oder gar WAP schon schwieriger, denn diese Formate lassen sich nicht durch den Austausch eines Templates ausgeben.

Der einfachste Ansatz, dynamisch und ausgabeformatunabhängig zu rendern, ist folgender: am Anfang des Programms wird eine Variable gesetzt (durch eine Konfigurationsdatei, aus einer Datenbank oder einfach nur durch einen Parameter), welche angibt, welche Ausgabemethode benutzt werden soll. Aus dieser wird dann ein Funktionsname generiert, der die Funktion angibt, welche schlussendlich das Rendering übernehmen soll.

```
$render_method = "html";
$render_function = "render_".$render_method;
....
$render();
```

Diese Art des Renderings setzt voraus, dass alle auszugebenden Inhalte vom Programm aufbereitet werden und mit einem einzigen Funktionsaufruf „ausgespuckt" werden können. Dies ist bei komplexeren Programmen allerdings in den seltensten Fällen eine akzeptable Möglichkeit.

Anmerkung: Da PHP eine Skriptsprache ist und deswegen keinen direkten Systembezug zur Hardware hat, gibt es in PHP keine Funktionsadressen, sondern nur Funktionsnamen. Dies bedeutet, dass dynamische Funktionsaufrufe nie über Funktionsadressen, sondern ausschließlich über Stringvariablen mit dem Funktionsnamen als Inhalt durchgeführt werden können. Die Stringvariablen mit dem Funktionsnamen können jedoch, wenn benötigt, über PHP-eigene Referenzen angesprochen werden.

Eine bessere, aber leider bei PHP nicht so häufig anzutreffende Art der Ausgabe ist das Zusammenfassen aller ausgaberelevanten Funktionen in einem Renderinglayer, ähnlich einem Datenbankabstraktionslayer. Dieses Renderinglayer ist dann dafür verantwortlich, eine formatunabhängige API zur Verfügung zu stellen, welche alle Aufgaben zur Formatierung und Ausgabe von Inhalten übernimmt. Da die meisten PHP-Applikationen für das Web geschrieben werden und damit auf HTML ausgerichtet sind, emuliert das Renderinglayer idealerweise HTML auf allen Ausgabeformaten, denn dies benötigt für die herkömmliche Programmierung am wenigsten Aufwand für Umstellung. So sind von der Programmlogik her WML, PDF, Klartext oder auch XML gleich zu behandeln, alleine das Renderinglayer ist dafür verantwortlich, korrekt formatierte Ausgaben zu liefern. Unterstützt ein Ausgabeformat ein bestimmtes Feature nicht, so muss das Renderinglayer dies entweder durch andere Formatierungsmethoden emulieren oder lässt es schlicht und einfach weg. Ein Beispiel zum Erstellen eines Formulars sähe damit wie folgt aus:

```
render_form_start($action);
render_text("Eingabefeld 1:");
render_form_input_box($name, $value);
render_form_submit("Abschicken");
render_form_end();
```

Im Renderinglayer könnten jetzt dynamisch anhand der wie im vorherigen Beispiel gesetzten Variable *$render_method* die entsprechenden Funktionen zur Formatbehandlung aufgerufen werden, d.h. die einzelnen Funktionen *render_form_start()*, *render_text()* usw. sind dafür verantwortlich, den Funktionsnamen *render_form_start_html()*, *render_text_html()* (je nach Konfiguration) zu erstellen und dann diese Funktion jeweils aufzurufen.

Als Alternative lässt sich dies auch direkt im aufrufenden Code realisieren, was zwar nicht genauso sauber ist, aber unter Umständen von Vorteil sein kann – wenn man beispielsweise gemischte Ausgabeformate benutzt.

Da PHP keine Funktionsnamen der Schreibweise *render_form_input_box_{$render_method}()* unterstützt, um den Funktionsnamen direkt beim Aufrufen zu generieren, muss ein anderer Weg gefunden werden. Hier hilft der Umweg über *call_user_func()*, welche einen String als Funktionsnamen akzeptiert. Strings können wie oben benötigt zusammengesetzt werden und erlauben dadurch einen „Workaround" um das *render_form_input_box_{$render_method}*-Problem:

```
call_user_func("render_form_input_box_".$render_
                    method, $arg1, $arg2, ...);
```

So bekommt also jede Funktion des Renderinglayers das Ausgabeformat als Suffix, zu implementieren wären damit *render_form_start_html()*, *render_text_html()* etc. Um damit nicht jede Funktion in einem *call_user_func()* „verschwinden" zu lassen und den Sourcecode so unlesbar zu machen, gibt es noch die Möglichkeit, einen generischen Wrapper zu schreiben:

```
function render($type, $what, $args)
{
 $func_name = „render_".$what."_".$type;
 $func_name($args);
}

render("html", „form_start", $args);
```

Hier geht alles durch einen einzigen Wrapper, der aus seinen Parametern den aufzurufenden Funktionsnamen zusammenbaut und dann die Argumente weitergibt. Vereinfachenderweise wurde hier nur ein Parameter *$args* benutzt, in einer komfortableren Version könnte natürlich eine Funktion benutzt werden, die transparent zum Aufruf der eigentlichen Funktion (wie hier *render_form_start_ html()*) eine variable Anzahl von Argumenten akzeptiert und diese mit Hilfe von *func_get_args()* dynamisch an die dynamische Funktion weitergibt. Man sieht also, die Art und Weise solcher Funktionsaufrufe lässt diverse Spielarten zu.

Callbacks

Um sich jedoch nicht hier zu verlieren, soll eine weitere Form der dynamischen Funktionen vorgestellt werden, die so genannten Callbacks (englisch „Rückruf"). Callbacks sind Funktionen, die ebenfalls nicht direkt aufgerufen werden, sondern die von einer anderen Funktion mittels eines dynamischen Funktionsaufrufs „zurückgerufen" werden. Man kann sich das wie ein Telefonrückrufsystem vorstellen, bei dem der Anrufer seine Nummer (oder die eines anderen) hinterlässt und dann automatisch zurückgerufen wird.

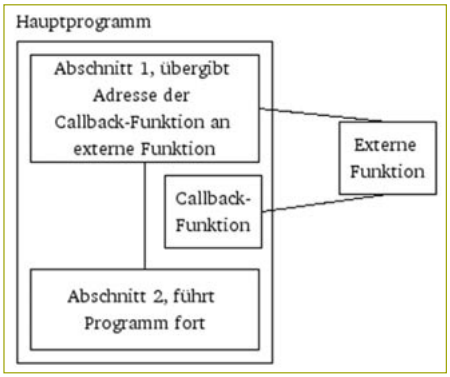


Abb. 1: Prinzip von Callbacks

Solche Callbacks sind sehr beliebt bei Algorithmen, die mit abstrakten Datentypen umgehen, oder auch bei ereignisorientierter Programmierung. Ein einfaches Beispiel lässt sich in PHP bei den Arrayfunktionen finden: *usort()*. Die *usort*-Funktion sortiert ein Array beliebiger Datentypen nach seinen Werten. Da der Sortieralgorithmus jedoch nicht über die Beschaffenheit der Daten Bescheid wissen kann und damit auch nicht feststellen kann, ob ein gegebenes Element *A* größer, kleiner oder gleich einem zweiten gegebenen Element *B* ist, muss der Programmierer eine eigene Vergleichsfunktion mitliefern. Diese Vergleichsfunktion wird dann als Callback für die Elementvergleiche aufgerufen:

```
function compare ($a, $b)
{
 if ($a == $b)
  return 0;

 return ($a > $b) ? -1 : 1;
}

$shuffle = array (3, 2, 5, 6, 1);

usort ($shuffle, "compare");
```

In diesem Beispiel wird *usort()* als Callback die Funktion *compare()* übergeben, welche zwei Integerwerte auf ihre

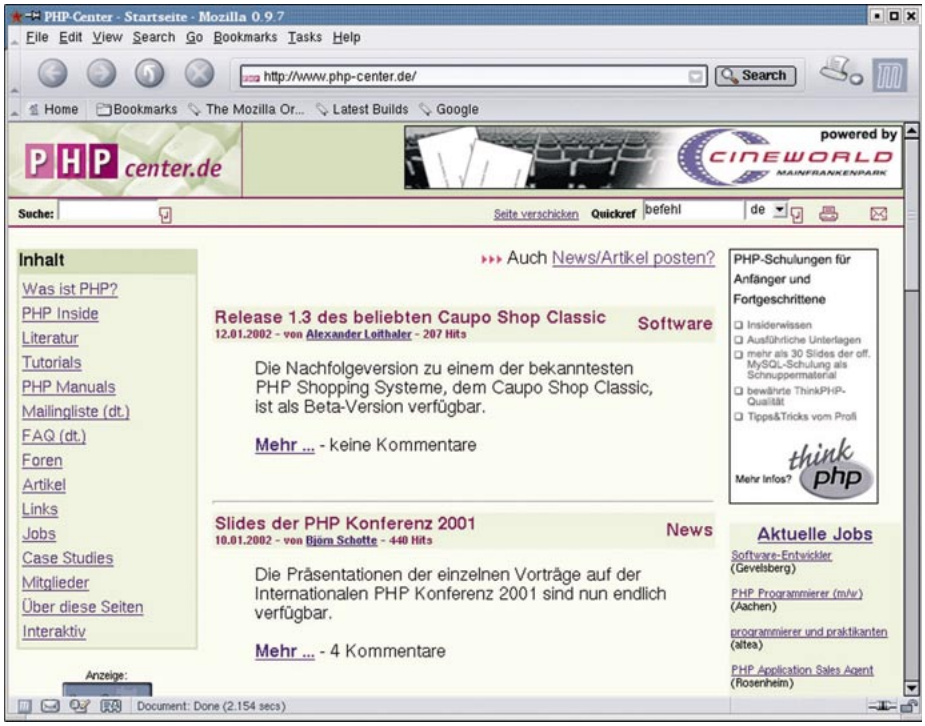


Abb. 2: PHP-Center Portal

Wertigkeit überprüft. Bei Gleichheit gibt die Funktion *0* zurück, ist *a* größer als *b*, gibt die Funktion *-1* zurück, sonst *1*. Anhand der Rückgabewerte kann *usort()* feststellen, in welche Reihenfolge es die Elemente zu bringen hat. Natürlich ist das Vergleichen von zwei Integern trivial, bei der Sortierung von Objekten ist jedoch eine Sortierung ohne die „Hilfe" des Programmierers unmöglich.

Um das „Innenleben" von *usort()* etwas zu illustrieren, enthält Beispiel 2 zu diesem Artikel (auf der Heft-CD) die Implementation eines *Bubblesort*-Algorithmus, welcher ebenso wie *usort()* von einer Callback-Funktion zum Vergleich zweier Elemente Gebrauch macht.

Callbacks sind unter anderem auch äußerst nützlich, um auf Ereignisse im Programmablauf zu reagieren. Beispielsweise ermöglicht das Skript phpIRC (www.phpwizard.net/projects/phpIRC) die Verarbeitung des IRC-Protokolls. Um auf bestimmte Ereignisse wie z.B. das Betreten eines Raumes durch einen User, das Verlassen eines Raumes oder einfach nur auf den Empfang von Nachrichten zu reagieren, können Callbacks registriert werden, die jedes Mal beim Eintritt dieses Ereignisses aufgerufen werden.

```
function onjoin_callback()
{

}

irc_init();
irc_connect("irc.belwue.de", 6667);
irc_join("#php.de");
irc_add_callback(IRCCB_ONJOIN, "onjoin_callback");
irc_idle();
```

Dieser Programmauszug initialisiert phpIRC, verbindet sich zu einem Server, betritt den Channel *#php.de* und registriert dann einen Callback, der immer dann aufgerufen wird, wenn jemand den Channel betritt. *IRCCB_ONJOIN* gibt das Ereignis an, „Joinen" bedeutet im IRC, einen Raum zu betreten. phpIRC wartet also jetzt, bis im Protokollstrom das Ereignis „jemand betritt #php.de" gefunden wird. Daraufhin sucht es seine Callbackfunktionen ab, bis es *onjoin_callback()* gefunden hat, welches auf dieses Ereignis passt. *onjoin_callback()* wird von phpIRC aufgerufen, anschließend wird das Programm normal fortgesetzt.

Plugins

Durch ereignisorientierte Programmierung lässt sich ein extrem flexibler Programmaufbau realisieren, der nicht nur leicht zu durchschauen, sondern auch vom Aufbau her extrem skalierbar ist. Das Programm schickt sich praktisch selbst Nachrichten und einzelne Programmteile können diese Nachrichten abfangen und bearbeiten. Der gesamte Code ist damit rund um einen Nachrichtenverteiler aufgebaut, was ein vergleichsweise kleines Grundwerk benötigt (um das Programm in seiner einfachsten Form zum Laufen zu bekommen, muss nur dieser Nachrichtenverteiler implementiert sein) und was als positiven Nebeneffekt eine modulare Programmstruktur „erzwingt", denn jeder Programmteil muss sich jeweils um einen Nachrichtenempfänger gruppieren.

Um dies zu illustrieren, wurde hier auf kleine Codeabschnitte im Text verzichtet, dafür ist auf der Heft-CD ein Mini-Framework enthalten, welches aus einem fertigen, eventbasierten Pluginsystem besteht.

Plugins sind die Verschmelzung der drei oben beschriebenen Konzepte – dynamische Funktionen, Callbacks und eventbasierte Programmierung. Plugins (zu Deutsch „Steckmodule") sind Programmteile, die während der Laufzeit zu einem fertigen Programm hinzugefügt werden können. Dies ist extrem nützlich, um ein Programm während des Betriebs zu erweitern und zu verändern. Viele Programme in nahezu allen Sparten setzen mittlerweile Plugins ein: MP3-Player, um während des Abspielens von Musik Spektralanalysen grafisch auszugeben, Bildbearbeitungsprogramme, um Import- und Exportfilter nachzuladen usw.

In PHP ist so etwas sehr einfach zu realisieren und auch sehr sinnvoll, beispielsweise in Verbindung mit Portalen. Jedes Portal erfüllt immer einen Satz gewisser Grundanforderungen: Benutzerverwaltung, Menüführung, ein Ausgabeformat mit getrennten Abschnitten für Menü, Hauptteil, Seitenteil.

Dies ist praktisch der kleinste gemeinsame Nenner aller Portale. Darauf bauen dann die portaleigenen Funktionalitäten wie Newsticker, Forum, Utilities, Messaging und dergleichen auf. Häufig werden Portalsysteme für jede Anwendung komplett neu geschrieben (oder nur zu einem gewissen Grad auf existierendem Code aufgebaut), die spezialisierten Funktionen

werden dann meist hart kodiert. Trotz physikalischer Modularisierung in einzelne Dateien sind die jeweiligen Module jedoch alle voneinander abhängig. Entfernt man eines, ist der Rest des Portals ohne Anpassungen nicht mehr lauffähig. Ebenso verhält es sich mit dem Portal selbst, das Grundsystem kann ohne die Module nicht funktionieren.

Dies führt zu einer Vermischung von Basiscode und darauf aufbauendem Code, was eine gravierende Folge hat: den Verlust von Universalität. Der Basiscode des Portals könnte beispielsweise doch auch für einen Weborganizer, eine spezialisierte Dokumentenablage und eine Wissensdatenbank benutzt werden, um nur ein paar Anwendungen zu nennen.

Eine bessere Lösung ist die Kapselung aller Funktionalitäten wie Newsticker, Forum etc. in Plugins. Das Grundsystem lädt dann einen bestimmten Satz von Plugins, der die Funktionalität der Applikation definiert. Hier kommt eine wichtige Änderung der Denkweise zum Tragen, nämlich nicht mehr das Programmieren nach monolitischer Funktionalität, sondern das Programmieren nach voneinander unabhängigen Modulen.

Im auf der CD enthaltenen Beispielsystem funktioniert dies folgendermaßen: *main.php* ist die Hauptdatei, die das eigentliche Grundsystem darstellt. In *main.php* sind Routinen zum Registrieren der Plugins, zum Behandeln von Events und zur Ausgabe enthalten. Um das System überschaubar zu halten, wurde auf Templating verzichtet und stattdessen aufgesplittetes HTML verwendet.

Ein Plugin wird eingebunden, indem es in der *modules.inc.php* per *include()* aufgerufen wird. Das *include()* veranlasst die Übersetzung und Ausführung des Plugincodes. Zum Zeitpunkt des Einbindens ist das einzige, was ein Plugin tun muss, das Aufrufen der Funktion *plugin_register()* in *main.php*. Dieser Funktion wird ein Deskriptor übergeben, der angibt, welche Funktion im Plugin zur Behandlung von Nachrichten dient, welche Version das Plugin hat, welchen Namen es hat usw. Anhand des Pluginnamens kann jedes Plugin von anderen Plugins abgefragt werden. Dies ist wichtig, wenn ein Plugin z.B. mit einem anderen Plugin Daten austauschen möchte.

plugin_register() speichert nun den Plugindeskriptor in einem Array mit allen anderen Plugins. Damit ist das Plugin registriert und „angemeldet“ und wird in Zukunft bei der Verteilung von Nachrichten mit einbezogen.

Nachdem alle Plugins eingebunden sind, werden zuerst alle Events verteilt. Es gibt zwei Typen von Events: gerichtete Events und Broadcasts. Gerichtete Events gehen nur an ein einzelnes Event, beispielsweise wenn ein Modul den Fokus bekommt (also im nächsten Schritt dargestellt werden soll), Broadcasts gehen an alle Events. Wenn ein Nutzer zum ersten Mal *main.php* aufruft, wird an alle Plugins ein Initialisierungsevent geschickt.

Man muss sich hier noch einmal verdeutlichen, dass das Grundsystem praktisch keine Ahnung von dem hat, was es wirklich tut, das Einzige, was es macht, ist, Plugins zu verwalten und Nachrichten an diese zu verteilen. Die Endapplikation entsteht rein dynamisch durch die Funktionalität der einzelnen Plugins! Sehr schön lässt sich dies demonstrieren, wenn man die *modules.inc.php* editiert und ein paar der *include()*-Statements auskommentiert. Am besten ist der Einbau eines Fehlers in eines der Plugins: PHP wird zwar einen Parse Error generieren, das Programm selbst bleibt jedoch lauffähig. Das Plugin wird schlichtweg nicht eingebunden und beeinflusst das übrige System durch seine Fehlerhaftigkeit in keiner Weise.

Anhand der Liste der eingebundenen Plugins generiert das Grundsystem eine Taskleiste, eine Titelleiste und eine kleine Tray. In der Titelleiste wird der Name des aktuellen Plugins zusammen mit einem von dem Plugin selbst zu bestimmenden String dargestellt (z.B. „Dateieditor - neu_datei.txt“). Im Hauptteil des Schirms können Plugins ihre eigene Ausgabe darstellen, die Tray wird, ähnlich wie die Systray unter Windows, mit von den Plugins generierten Icons bestückt.

Um mit dem Benutzer zu interagieren, wird alles über Events gesteuert. Links und Formulare, die von den Plugins ausgegeben werden, werden mit einem Eventcode verbunden. Alle Links und Formulare zeigen als Ziel jeweils wieder auf *main.php* mit dem jeweiligen Eventcode. Klickt der User nun auf einen Link, wertet *main.php* aus, für wen dieses Event gedacht war und schickt es an das entsprechende Plugin. Dadurch, dass das Plugin nun nur noch Events an sich selbst schickt und diese mit Events von anderen Modulen verarbeitet, ist die gesamte Programmierlogik um den Eventhandler gruppiert. Dies ermöglicht eine Programmierung ähnlich wie unter Windows, die ganze Ausgabelogik, Sessionlogik und was Webapplikationen sonst noch alles mit sich bringen, entfällt, da dies ausschließlich Aufgabe des Grundsystems ist. Das Renderinglayer des Beispielsystems ist noch etwas trivial, es werden beispielhaft nur Formulare und Links unterstützt. Idealerweise sollte alles HTML-abhängige aus den Plugins verschwinden.

Da jedes Event eine eigene Eventrange besitzt, die es beim anfänglichen Registrieren automatisch zugewiesen bekommt, ist es für das Grundsystem einfach festzustellen, welches Event für welches Plugin bestimmt ist. Damit Plugins sich untereinander Events schicken können, müssen sie zuerst die Eventrange des Empfänger bestimmen. Als Beispiel hierzu dient der Module Dumper, der alle im System installierten Plugins abfragt und ausgibt.

Schaut man sich den Sourcecode genauer an, ist sogar das Grundsystem ähnlich wie die übrigen Plugins aufgebaut, der einzige Unterschied ist, dass es die für die anderen Module benötigte API zur Verfügung stellt. Das vorliegende System ist so bereits sehr flexibel einsetzbar, als denkbare Erweiterungen wären jetzt nur noch ein Usermanagement mit Loginseite (ebenfalls als Plugin), eine Erweiterung des Renderinglayers (falls gewünscht) und ein Hinzufügen eines Template-Systems nötig, um es auch in größeren Anwendungen einzusetzen.

Da in einem Artikel leider zu wenig Platz ist, um detailliert auf den Quelltext einzugehen, wurde der Code ausführlich dokumentiert. Mit der Studie eines der kleinen Beispielmodule sollte sich recht schnell die Funktionalität erschließen. Einem Einsatz im „echten Leben“ steht also nichts im Wege.

XMetal 3 verfügbar

SoftQuad Software hat die Verfügbarkeit von XMetal 3 bekannt gegeben. Der validierende XML-Editor soll die Erstellung und Wiederverwendung von Business Content erleichtern. XMetal soll in der aktuellen Version Kollaborationsfunktionalitäten, Support für XML Schemas, HTML- und PDF-Vorschauen, ein Java API sowie WebDAV-Befähigung bieten.
www.softquad.com/

Microsoft und IBM führen neue Web Services-Allianz an

Unter dem Namen Web Services Interoperability Organization wollen eine Reihe führender IT-Unternehmen gemeinsam an der Entwicklung von Standards für Web Services-Infrastrukturen und -Anwendungen arbeiten. Eine Führungsrolle könnten dabei Microsoft und IBM spielen, die bereits in verschiedenen Standardisierungsgremien kooperieren. Weitere Branchengrößen wie Bea, Sun, Intel und Oracle sollen ebenfalls einen Beitritt zur WSIO erwägen.
Aufbauend auf den Standards wie XML, SOAP und UDDI will die Allianz zusätzliche Industriestandards für webbasierte Geschäftsprozesse definieren und dabei vor allem Probleme der Interoperabilität von Plattformen – etwa zwischen Betriebssystemen, Hardwareplattformen und Sprachen – adressieren.
www.ws-i.org/

Open3.org stellt Messaging Server 2.1 vor

Open3.org hat Version 2.1 des Open Source-Produkts Open3.org JMS Messaging Server freigegeben. Die Java-basierte Software ist nach Aussage der Entwickler vollständig konform mit der JMS 1.0.2 Spezifikation und bietet native XML-Unterstützung. Downloads werden für Linux, Unix und Windows angeboten. Das Open Source-Projekt Open3.org arbeitet an der Implementierung einer vollständigen Plattform für Enterprise Integration.
www.open3.org/

eXcelon bringt XIS in Version 3

Die eXcelon Corporation hat Version 3 ihres eXtensible Information Servers (XIS) veröffentlicht. Das XML Datenbank-Management-System wurde in der neuen Version vor allem schneller und effizienter gemacht, teilte das Unternehmen mit. Außerdem ist eXtensible jetzt mit J2EE-Umgebungen kompatibel und es wurde zu den neuen XML-Standards wie XQuery, XPath 1.0, XSLT 1.0 und XML Schema 1.0 upgedatet. Eine Lizenz für fünf Entwickler ist ab 30.000 US-Dollar zu haben. Außerdem bietet das Unternehmen eine kostenlose Testversion von XIS.
www.exln.com/

SVG Editor WebDraw verfügbar

Der amerikanische Grafiksoftware-Anbieter Jasc hat die kommerzielle Verfügbarkeit des SVG Tools WebDraw bekannt gegeben. Das Programm soll die native Bearbeitung von SVG-Dateien erlauben, die Erstellung für Animationen unterstützen und Zugriff auf den SVG-Quellcode bieten. Neben der kommerziellen Version steht eine Evaluierungsversion des nur für die Windows-Plattform verfügbaren Tools auf der Unternehmensseite zum Download bereit.
www.jasc.com/products/webdraw/

HP gibt FO-Tool als Open Source frei

Hewlett-Packard hat das Formatting Objects-Autorentool FOA in Version 0.2.0 unter die Mozilla Public License gestellt. Das Java-basierte Programm soll Entwicklern über eine grafische Oberfläche die Generierung von XSL-FO Ausgaben aus XSLT Stylesheets erlauben, ohne selbst XSLT oder XSL-FO Code schreiben zu müssen. Neu in der aktuellen Version sind die Unterstützung von Tabellen und eine interaktive Vorschau.
foa.sourceforge.net/

Jabber bietet Archivierung für das Instant Messaging

Jabber Inc., Anbieter der XML-basierten Instant Messaging Software Jabber, erweitert seine Kommunikationsplattform um zentrale Log-Mechanismen. Die Jabber Communications Platform in der Version 1.2 erstellt zentrale Logs für den gesamten Instant Messaging Verkehr, ein Feature, das für Unternehmen, die auf Jabber zurückgreifen, von Bedeutung ist. Finanzdienstleister können nun den gesamten IM-Austausch mit ihren Kunden serverseitig loggen.
www.jabber.com/

Nusphere unterstützt PHP-basierte Web Services

Die Nusphere Corporation hat die integrierte Unterstützung für die Entwicklung Open Source-basierter Web Services angekündigt. Das Paket für Entwickler soll aus Nuspheres IDE PHPEd und Open Source Technologien wie Apache, Perl und MySQL geschnürt werden. Das Unternehmen will mit dieser Lösung eine Alternative zu Ansätzen wie Microsoft .NET und J2EE bieten, die zum großen Teil auf proprietäre Tools setzten und Entwickler in Herstellerabhängigkeiten zu bringen drohten. Auf der Nusphere Website werden u.a. White Papers und Anwendungsbeispiele zur Verfügung gestellt, mit denen gezeigt werden soll, dass es sich bei PHP um „das schnellste und kostengünstigste" Tool zur Entwicklung von Web Services handelt.
www.nusphere.com/

XForms Last Call Release

Die XForms Working Group des W3C hat einen Last Call Working Draft für XForms 1.0 veröffentlicht. Im Changelog kann nachvollzogen werden, welche Elemente sich im Vergleich zur Vorgängerversion geändert haben. Die Gruppe betont, dass die Änderungen sich vor allem auf kleine Verbesserungen beschränken. XForms ist eine XML-Applikation, die eine neue Generation von Webformularen einleiten soll. XML-Forms werden in drei Bestandteile – Datenmodell, Instance Data und User Interface – zerlegt, sodass Präsentation und Inhalt getrennt werden können.
www.w3.org/TR/xforms

Gründlich eingeseift

SOAP Web Services mit PHP

Web Services und speziell das Simple Object Access Protocol SOAP [1] sind momentan in aller Munde. Mit SOAP gelingt die Anbindung an größere Softwareeinheiten wie Billing Systeme oder Application Server. Schaut man hinter den Hype der Web Services, so findet man einen portablen, in vielen Sprachen und Plattformen einsetzbaren Standard zur Interoperabilität. Auch für PHP gibt es eine Reihe von SOAP-Implementierungen, von denen hier eine vorgestellt und in einem praktischen Beispiel eingesetzt werden soll.

Dr. Volker M. Göbbels

Unter dem Stichwort Interoperabilität versteht man die Möglichkeit, von einem Rechner (Client) aus Aktionen, Programme oder Routinen auf einem anderen Rechner (Server) aufzurufen und Ergebnisse dieser Aktion zurückgeliefert zu bekommen. In der Unix-Welt ist diese Technik schon lange unter dem Stichwort Remote Procedure Calls oder RPC bekannt. Ein Analogon hierzu stellt XML-RPC dar. Ein deutlich modernerer Standard mit ähnlicher Funktionalität ist SOAP. Im Widerspruch zu seinem Namen beschäftigt sich SOAP aber in keiner Weise mit Objekten im Sinne der OOP.

Das Publizieren von Applikationsdiensten per Webschnittstelle ist allerdings eine neue und bisher nur ansatzweise genutzte Methode, Rechner irgendwo auf der Welt flexibel zu verbinden. Die Nutzung des Web als Transportbasis hat den Vorteil, dass der Datenverkehr ungehindert über den auf den meisten Firewalls und Gateways freigeschalteten Port 80 laufen kann. Änderungen an der Infrastruktur sind daher nicht nötig. Die Daten werden in XML ja zudem im ASCII-Format übertragen.

Transportprotokolle

SOAP Messages können prinzipiell über jedes Netzwerkprotokoll übertragen werden, welches Texte transportieren kann. Eingesetzt werden allerdings lediglich HTTP und SMTP. SMTP hat jedoch den Nachteil, asynchron zu arbeiten, d.h., die Nachricht kommt genau dann beim SOAP Server an, wenn dieser sie vom Mail Server abholt. Daher hat sich HTTP als Standard-Transportprotokoll für SOAP Messages durchgesetzt.

Zusätzlich bietet HTTP über die HTTP Response die Möglichkeit, aus der per Standard als Einwegübertragung definierten SOAP Message durch Rückgabe einer HTTP Response, die wieder eine SOAP Message enthält, ein Zweiwegeprotokoll zu machen. Die Verwendung eines Webservers (in unserem Fall der Apache) ermöglicht auch den Einsatz von PHP im SOAP Serverpart, ohne selbst einen Socketserver schreiben zu müssen. Das PHP SOAP Serverskript liegt einfach im document tree des Webservers und ein Aufruf der URL dieses Skripts setzt den SOAP Server für diese eine Abfrage in Gang. Etwas komplizierter wird das Szenario dadurch, dass der beim SOAP Server anfragende Rechner in unserem Fall selbst ein Webserver bzw. ein PHP Skript ist (siehe Abb. 1).

Der doppelte Indianer

Zum Experimentieren mit SOAP benötigt man aber nun nicht gleich zwei getrennte Webserver. Client- und Serverfunktionalität können mit einem Server bedient werden. Um den SOAP Client- und Serverteil deutlicher vom Webtraffic zu trennen und ein „Abhören“ des Transports im Netz zu ermöglichen, lassen wir die SOAP Kommunikation nicht über Port 80 sondern über den privaten Port 20080 laufen. Die hierzu nötige Konfiguration des Apache zeigt das folgende Lis-

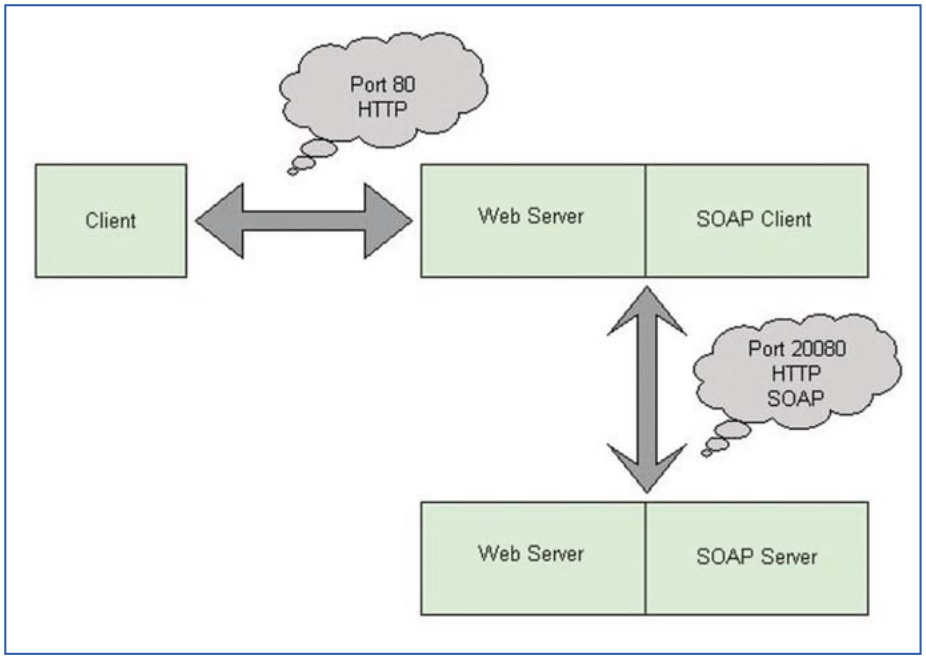


Abb. 1: Ein-Server-Szenario für SOAP Anwendungen

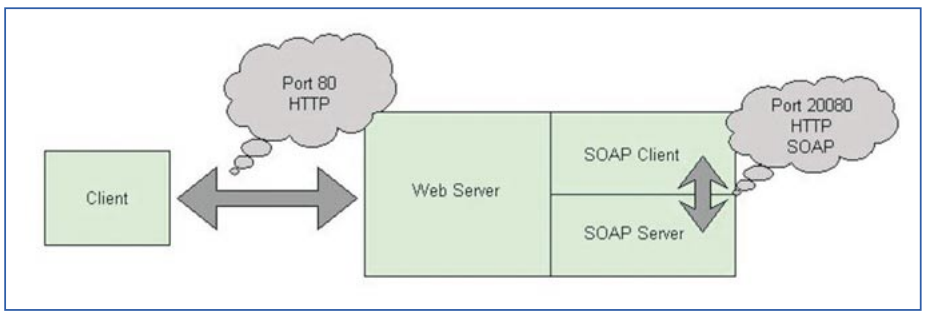


Abb. 2: Ein-Server-Szenario zum Test von SOAP Applikationen

ting, welches Ausschnitte aus der *httpd.conf* enthält:

```
ServerRoot „/opt/prj/cw“
...
Listen 80
Listen 20080
...
DocumentRoot „/opt/prj/cw/htdocs“
...
<VirtualHost 192.168.0.99:20080>
 DocumentRoot /opt/prj/cw/soap
 <ifModule mod_dir.c>
  DirectoryIndex index.html index.php
 </ifModule>
</VirtualHost>
```

Startet man den so konfigurierten Apache, zeigt ein Blick in die Liste der offenen Ports die korrekte Arbeitsweise:

```
paradiso:/usr/local/web # lsof -i
...
httpd     766     root     16u     IPv4
6006      TCP *.soap (LISTEN)
httpd     766     root     17u     IPv4
6007      TCP *.http (LISTEN)
...
```

Diese „port based virtual host“ Konfiguration lässt den Apache auf zwei Ports horchen: Webport 80 zeigt Dokumente ab dem document root */opt/prj/cw/htdocs*, Port 20080 verweist auf ein document root in */opt/prj/cw/soap*. Hier liegt das später noch vorzustellende Serverskript *SOAP_SERVER.php*. In dieser Konfiguration wird kein *NameVirtualHost* definiert. Die Struktur dieses Setups zeigt Abbildung 2.

In */opt/prj/cw/soap* liegen auch die Klassen der SOAP Implementierung SOAPx4 von Dietrich Ayala [2], *class.soap_server.php* und *class.soap_client.php*. Dieses Verzeichnis sollte in *php.ini* daher auch im *include_path* aufgeführt werden.

Anatomie eine SOAP Message

Auch wenn sich eine SOAP Implementierung um das Ein- und Auspacken von Funktionsaufruf und Antwort kümmert, sollte man über den grundsätzlichen Aufbau einer SOAP Message informiert sein. Insgesamt gibt es drei Typen von Messages, die alle den in Abbildung 3 dargestellten Aufbau besitzen.

Unterschieden werden SOAP Aufrufe, Responses und Fault Messages. Allen gemeinsam ist der Aufbau bestehend aus einem Envelope, einem optionalen Header und einem Body, der die eigentliche Nachricht und ihre Parameter, auch *payload* genannt, enthält. Wie bei einem XML-basierten Protokoll nicht anders zu erwarten, stellen alle drei Typen valide XML Dokumente dar, unterliegen jedoch ein paar Einschränkungen:

- Das Rootelement des Dokuments heißt immer Envelope.
- Das Dokument darf keine Schemadefinition enthalten, weder in Form einer DTD noch als XSchema. Dies würde auch wenig Sinn ergeben, da die Syntax einer SOAP Message genau festgelegt ist.
- Der Namespace für Envelope, Header und Body ist immer SOAP-ENV.

Da ein konkretes Beispiel oft mehr sagt als tausend Worte, sehen Sie in Listing 1eine einfache Message aus unserem kommenden Fallbeispiel, in dem eine Adressdatenbank in MySQL anhand eines Nachnamens abgefragt werden soll.

Dieser Aufruf enthält einen Envelope, der eine Reihe von Namespaces definiert. Die wichtigsten hiervon sind:

- *xsd* (XML Schemadefinition) und *xsi* (XML Schemadefinition Instance) zur Definition der verwendeten XML Datentypen.
- *SOAP-ENC* für das Encoding, das heißt das Verpacken oder Serialisieren der Datenstrukturen.

- *SOAP-ENV* als Namespace für die SOAP Message selbst.
- *ns6* als Namespace der eigentlichen Funktionsaufrufe und Parameter.

In diesem speziellen Fall wird die Routine *getAdr* mit genau einem Parameter aufgerufen. Dieser Parameter ist ein Array (*struct*) mit dem Namen des PHP-Arrays (*query*), mit dem der Aufruf erfolgte, und enthält lediglich ein Element, nämlich *name=„huber“*.

Diese Message enthält keinen Header. Ein Header dient dazu, zusätzliche Daten mit dem eigentlichen Aufruf zu verschicken. Typische Beispiele sind Transaktionsdaten, wie sie ein Beispiel aus der SOAP Recommendation des W3C [1] zeigt:

```
<SOAP-ENV:Header>
 <t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnder-
stand="1">
  5
 </t:Transaction>
</SOAP-ENV:Header>
```

Das Attribut *mustUnderstand=1* besagt, dass der Empfänger dieses Element namens *Transaction* mit dem Wert *5* verstehen muss. Kann er das nicht, muss er eine SOAP Fehlermeldung zurückliefern und darf den Aufruf nicht weiter ausführen.

Das einzige weitere standardisierte Attribut zu einem Headerelement ist *actor*. Sollte eine SOAP Message nicht direkt an den ausführenden Rechner (den so genannten Endpoint) geschickt werden, sondern einen oder mehrere SOAP-Proxies durchlaufen, also Rechner, die SOAP Messages lesen und interpretieren bzw. weiterleiten können, kann mit dem Actor der tatsächliche Endpunkt definiert werden. In den meisten Standardfällen wird man SOAP Header nicht benötigen.

Die Antwort auf unsere Anfrage könnte aussehen wie in Listing 2.

Die Antwort besteht laut Standard aus einem einzigen Element mit dem Namen der aufgerufenen Routine (*getAdr*) und angehängtem „Response“. Darin befinden sich alle zurückgegebenen Daten, in diesem Fall wieder ein Array mit dem Namen der antwortenden Routine. Darin enthalten sind die eigentlichen Daten.

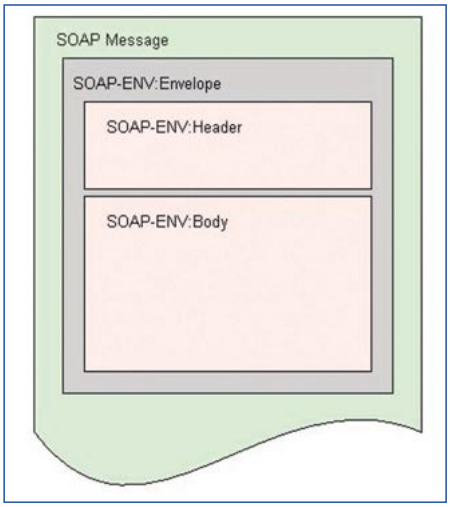


Abb. 3: Allgemeiner Aufbau einer SOAP Message

Der letzte Typ SOAP Messages sind schließlich die Fault Messages. In diesen wird eine Fehlermeldung vom SOAP Server auf den Client transportiert. Ein typisches Beispiel aus der SOAP Recommendation finden Sie in Listing 3.

Fault Messages haben wieder genau ein Element zum Inhalt. Dieses hat den Namen *Fault* und kann eine Reihe Subelemente enthalten:

Anzeige

- *faultcode* enthält einen maschinell verarbeitbaren Fehlercode und ist zwingend vorgeschrieben. Der SOAP Standard definiert eine kleine Anzahl dieser Codes (Client, Server, VersionMismatch, MustUnderstand). Diese Faultcodes gehören zum Namespace *SOAP-ENV* und können durch eigene „Subklassen" von Fehlern erweitert werden, indem die eigene Fehlerbezeichnung mit einem Punkt angehängt wird, z.B. *Server.NotFound*.
- *faultstring* enthält eine menschenlesbare Fehlermeldung und muss angegeben werden.
- *faultactor* bezeichnet den Ort bzw. die URL, analog zum *actor* Attribut, an dem der Fehler aufgetreten ist. SOAP Applikationen, die nicht der Endpunkt einer SOAP Message sind, müssen einen *faultactor* mit dem eigentlichen Ursprung des Fehlers anlegen. Der tatsächliche Endpunkt der Message, an dem der Fehler aufgetreten ist, kann einen solchen *faultactor* aufnehmen.
- *detail* ist dazu gedacht, Inhalte zu transportieren, die durch einen Fehler bei der Ausführung der Applikation auf dem Server erzeugt wurden, also den Inhalt des Body betreffen. In diesem Fall ist das *detail* Element Pflicht. Sollte der Fehler durch das SOAP Framework auf dem Server entstanden sein oder durch den Header der Message, so darf kein *default* Element angegeben werden.

Listing 1

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/
                                XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/
                                    soap/encoding/"
 xmlns:si="http://soapinterop.org/xsd"
 xmlns:ns6="http://soapinterop.org"
 SOAP-ENV:encodingStyle="http://schemas.
                        xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
 <ns6:getAdr>
  <query xsi:type="ns6:struct">
   <name xsi:type="xsd:string">huber</name>
  </query>
 </ns6:getAdr>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 2

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
                                    soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/
                                XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/
                                    soap/encoding/"
 xmlns:si="http://soapinterop.org/xsd"
 SOAP-ENV:encodingStyle="http://schemas.
                        xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
 <getAdrResponse>
  <getAdr>
   <nachname xsi:type="xsd:string">Huber</nachname>
   <vorname xsi:type="xsd:string">Gerhard</vorname>
   <strasse xsi:type="xsd:string">Benediktstr. 3</strasse>
   <plz xsi:type="xsd:string">50020</plz>
   <ort xsi:type="xsd:string">Koeln</ort>
   <tel xsi:type="xsd:string">0221/123456</tel>
  </getAdr>
 </getAdrResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 3

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://
                    schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
 <SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>Server Error</faultstring>
  <detail>
   <e:myfaultdetails xmlns:e="Some-URI">
    <message>
     My application didn't work
    </message>
    <errorcode>
     1001
    </errorcode>
   </e:myfaultdetails>
  </detail>
 </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4

```
<?php

include("class.soap_client.php");
include("class.soap_server.php");

$server = new soap_server;

$server->add_to_map(
 "getAdr",
 array("struct"),
 array("struct")
);

function getAdr($qdata)
{
 $db=mysql_connect("localhost","root","v1");
 mysql_select_db("adressen");
 $query="select * from adressen where nachname
                                    ='".$qdata["name"]."';";
 $res=mysql_query($query);
 $dump=mysql_fetch_assoc($res);
 $data["nachname"]=$dump["nachname"];
 $data["vorname"]=$dump["vorname"];
 $data["strasse"]=$dump["strasse"];
 $data["plz"]=$dump["plz"];
 $data["ort"]=$dump["ort"];
 $data["tel"]=$dump["tel"];
 return $data;
}

$server->service($HTTP_RAW_POST_DATA);

?>
```

Hier werden Sie geholfen

Nachdem wir nun schon gesehen haben, wie eine Anfrage und eine zugehörige Antwort aussehen, ist es natürlich interessant, zu testen, wie viel Aufwand die tatsächliche Implementierung in PHP bedeutet. Zum Glück kann man dabei auf fertige (so gut das bei Open Source Projekten geht) Frameworks wie das von Dietrich Ayala [2] zurückgreifen. Weitere Implementierungen, auch in anderen Sprachen, finden sich bei SoapWare.org [3].

Unser Testbeispiel soll eine Webpage mit einem Formular zeigen, in dem man einen Nachnamen angeben kann. Mit diesem Namen als Parameter soll das Form-Script als SOAP Client dann den SOAP Server nach den gesamten Adressdaten zu diesem Namen fragen. Das Serverskript soll diese Daten in einer MySQL Tabelle der folgenden Struktur suchen:

```
CREATE TABLE adressen (
 id int(11) NOT NULL default '0',
 vorname varchar(20) default NULL,
 nachname varchar(20) NOT NULL default '',
 strasse varchar(40) default NULL,
 plz varchar(10) default NULL,
 ort varchar(20) default NULL,
 tel varchar(20) default NULL,
 PRIMARY KEY (id),
 KEY vorname (vorname,nachname,tel),
 KEY id (id)
) TYPE=MyISAM;
```

Herr und Meister

Beginnen wir mit dem Serverskript *SOAP_SERVER.php* (Listing 4). Als Erstes müssen die Server und Client Klassen eingebunden und ein *soap_server* Objekt instanziiert werden:

```
include(„class.soap_client.php“);
include(„class.soap_server.php“);

$server = new soap_server;
```

Im Server muss auch die Client-Klasse inkludiert werden, da in dieser eine Reihe allgemeiner Definitionen zu finden sind. Dann bringt man dem Server bei, dass er nun eine neue Routine kennt:

```
$server->add_to_map(
 „getAdr“,
 array(„struct“),
 array(„struct“)
);
```

add_to_map wird mit drei Argumenten aufgerufen: dem Namen der einzubinden-

Listing 5

```
function getAdr($qdata)
{
 $db=mysql_connect(“localhost“, “user“, “password“);
 mysql_select_db(“adressen“);
 $query=“select * from adressen where nachname
                                    ='“.$qdata[“name“].“';“;
 $res=mysql_query($query);
 $dump=mysql_fetch_assoc($res);
 $data[“nachname“]=$dump[“nachname“];
 $data[“vorname“]=$dump[“vorname“];
 $data[“strasse“]=$dump[“strasse“];
 $data[“plz“]=$dump[“plz“];
 $data[“ort“]=$dump[“ort“];
 $data[“tel“]=$dump[“tel“];
 return $data;
}
```

Listing 6

```
<?

include(“class.soap_client.php“);

$nserver=“paradiso“;// welchen Server konnektieren?
$method=“getAdr“;        // welche Methode aufrufen?

$method_params[“getAdr“][“query“] = array(“name“=> $name);

$servers[$nserver] = array(
                “soapaction“ => “urn:soapinterop“,
                “endpoint“ => „http://$nserver:20080/SOAP_SERVER.php“,
                “methodNamespace“ => “http://soapinterop.org“,
                “name“ => $nserver);

print “<form action='$PHP_SELF' method='post'>“;
?>
Name: <input name=“name“ type=“text“ size=“30“>       
<input name=“__debug“ type=“checkbox“ value=“1“>   Debugging Infos?
</form>
<hr>
<?
if($name){
 $server = $servers[$nserver];
 $soap_message = new soapmsg($method,$method_params[$method],$server[“methodNamespace“]);
 $soap = new soap_client($server[“endpoint“]);
 if($return = $soap->send($soap_message,$server[“soapaction“]))
 {
  // check for valid response
  if(get_class($return) == “soapval“)
  {
   // fault?
   if(eregi(“fault“,$return->name))
    $status = “failed“;
   else
    $status = “passed“;
  }
  else
   print “Client could not decode server's response<br>“;
 }
 else
  print “Was unable to send or receive.“;
 if($__debug==1)
 {
  print “<strong>Request:</strong><br><xmp>$soap->outgoing_payload</xmp><br>“;
  print “<strong>Response:</strong><br><xmp>$soap->incoming_payload</xmp>“;
 }
 $data=$soap->response->decode();
 print “<table>“;
 foreach($data[“getAdr“] as $k => $v)
  print “<tr><td><b>$k</b></td><td>$v</td></tr>“;
 print “</table>“;
}

unset($soapmsg);
unset($soap);
?>
```

den Routine, einem Array mit den Typen der Input-Parameter und einem ebensolchen mit den Output-Parametern. Die Typenbezeichnungen entsprechen den XML Schema Datentypen (int, float, string, base64, struct u.a.).Diese Routine sollte man natürlich dann auch definieren (siehe Listing 5).

Als Letztes bleibt nur noch, den Server an die Arbeit zu schicken:

```
$server->service($HTTP_RAW_POST_DATA);
```

Die Client-Seite

Das Client Skript (Listing 6) beginnt mit dem obligatorischen Include der Clientklasse und der Definition des Servernamens und der Parameter für den SOAP Aufruf:

```
include(„class.soap_client.php“);

$nserver=“paradiso“;      // welchen Server konnektieren?
$method=“getAdr“;         // welche Methode aufrufen?

$method_params[$method][„query“]=array(„name“=>
$name);
```

Anschließend legt man in *$servers* die Verbindungsdaten zur Kontaktaufnahme mit dem SOAP Server an. Das Sammeln in einem Array ist vom API der SOAP Implementierung zwar nicht vorgeschrieben, erleichtert aber bei Skripten mit Kontakt zu verschiedenen SOAP Servern die Verwaltung der Daten.

Danach baut man ein ganz gewöhnliches Web Formular auf, welches im Prinzip nur ein Textfeld enthalten müsste. Damit man aber etwas von der dahinter liegenden Technik sieht, bauen wir eine Checkbox mit einem Debuggingflag ein, sodass neben dem üblichen Output auch der rohe Request und die Response ausgegeben werden.

Wie bei PHP-basierten Formularen üblich, überprüft man dann anhand einer Formularvariable, ob ein Submit erfolgt ist. Falls ja, folgen eine Reihe von Fehlertests und bei korrekter Antwort die Ausgabe der Ergebnisse.

Schauen wir uns zum Schluss die Serverparameter und Kontaktaufnahme zum SOAP Server etwas genauer an. Das *$servers Array* enthält eine Reihe von Angaben, die bei der Instanziierung der SOAP Message mittels der *soapmsg* Klasse benötigt werden:

- *endpoint* definiert als URL den Endpunkt der Aktion, also die URL, die für den HTTP Request verwendet werden soll. Daher enthält *endpoint* auch den Namen des PHP Skripts.
- *name* definiert einfach noch einmal den Servernamen, für den Fall, dass mehrere Server in *$servers* verwaltet werden.
- *methodNamespace* bestimmt, welche URL als Bezeichner für den Namespace der aufgerufenen Methode (in unserem Fall *ns6*) benutzt werden soll.
- *soapaction* ist ein von der W3C SOAP Recommendation vorgeschriebenes Feld für den HTTP Header, welches den beteiligten Servern, Proxies oder Firewalls deutlich machen soll, was der eintreffende Request genau für eine Funktion hat. Das Feld muss nicht gefüllt sein, wenn aus der Request URL alle nötigen Details des Requests hervorgehen. In diesem Fall darf das Feld leer sein. Im Header muss es trotzdem vorhanden sein.

Um den SOAP Server nun zu kontaktieren, erzeugt man eine SOAP Message mit den Parametern Methodenname, Parameter-Array und Methoden-Namespace:

```
$soap_message = new soapmsg($method,$method
        _params[$method],$server[„methodNamespace“]);
```

Dann instanziiert man einen neuen SOAP Client mit Angabe des Endpunktes der Aktion:

```
$soap = new soap_client($server[„endpoint“]);
```

Zum Schluss weist man den Client an, die Message mit einer bestimmten SOAP Action zu verschicken:

```
$return = $soap->send($soap_message,$server
                                [„soapaction“])
```

Der Returnwert der *send* Methode sollte ein Objekt der Klasse *soapval* sein. Falls dem so ist, kann man das im SOAP Server per *return* abgeschickte Array mit

```
$data=$soap->response->decode();
```

dekodieren. Das Ergebnis ist ein echtes PHP Array, wie ursprünglich von *getAdr* zurückgegeben. Die SOAP Implementierung führt das Kodieren und Dekodieren auch komplexer PHP-Returnwerte automatisch und völlig transparent durch.

Falls man die Low Level Kommunikation zwischen SOAP Server und Client sehen möchte, kann man den Inhalt der Variablen *outgoing_payload* und *incoming_payload* der Client-Klasse abfragen.

Ende gut, alles gut?

Wie wir gesehen haben, ist die Verwendung von SOAP in PHP leicht und gut verständlich. Die in diesem Beitrag verwandte SOAP Implementierung von Dietrich Ayala bietet über den reinen Remote Call hinaus, wie er hier vorgestellt wurde, auch eine WSDL Implementierung zur Beschreibung der implementierten SOAP Methoden.

Dem Einsatz von SOAP im PHP-Umfeld sind kaum Grenzen gesetzt. So ist es möglich, Java-basierte Storage Container oder Application Server anzubinden.

Auch existieren schon Business Applikationen wie Billing Systeme mit einer SOAP Schnittstelle, sodass der Zugriff auf eine Online-Rechnung oder die Änderung der eigenen Kundendaten mittels einer Webpage ohne direkten Eingriff in das Abrechnungssystem möglich werden.

Dr. Volker Göbbels ist Geschäftsführer der Arachnion GmbH & Co. KG. Die Arachnion (www.arachnion.de) befasst sich mit der Beratung und Implementierung im Bereich Web Applikationen und Billing Systeme sowie Schulungen im XML und PHP Umfeld. Dr. Göbbels ist Mitglied im Team ThinkPHP (www.thinkphp.de).

Links & Literatur

[1] www.w3.org/TR/SOAP/

[2] dietrich.ganx4.com/soapx4/

[3] www.soapware.org/

Anzeige

Anzeige

Anzeige

Ein starkes Duo!

XML-basierter Datenaustausch zwischen PHP und Macromedia Flash

Flash mit PHP? Warum auch nicht? Beide haben mittlerweile eine Reife erreicht, die neue Möglichkeiten zum Leben erweckt – ein Datenaustausch zwischen PHP und Flash ist nun sogar auf XML Basis möglich... Das ist ein guter Zeitpunkt für eine kleine Entdeckungsreise, um die Kommunikationsmöglichkeiten zwischen PHP und Flash sowie die Anwendungsmöglichkeiten dieses starken Duos zu erkundschaften.

Sebastian Mordziol

Weiterhin von Vielen als Grafiker-Spielzeug angesehen, hat Flash mittlerweile mit Version 5 doch manches für uns Programmierer zu bieten. Mit den neuen Datenschnittstellen und dem weiter ausgebauten ActionScript beinhaltet Flash nun seine eigene – wenn auch noch nicht vollwertige – Entwicklungsumgebung. Mit diesen neuen Möglichkeiten haben sich auch die Tore zum dynamischen Datenaustausch zwischen Server und Flash geöffnet. Hier kommt PHP ins Spiel – obwohl andere serverseitige Sprachen wie ASP hier auch geeignet wären (kleiner Scherz am Rande...), bietet PHP im Gesamtbild mehr Möglichkeiten, mit Flash umzugehen (allein schon mit den existierenden Libraries zur Flash-Generierung wie zum Beispiel den Ming und LibSWF Libraries [1, 2]).

Vorteile von Flash gegenüber HTML & DHTML

Mit HTML und DHTML kann schon sehr viel realisiert werden und Flash kann diese zum Teil auch nicht ganz ersetzen. Heute noch wird Flash meistens erwähnt, wenn es um Animationen, Soundspielereien und sonstiges in Webseiten geht – das ist aber noch nicht die ganze Geschichte. Flash ist plattformunabhängig, sofern der Player oder das Browserplugin beim Client vorhanden sind, sprich das Aussehen und die Funktionalität bleiben immer gleich – was man von DHTML leider noch nicht behaupten kann. Dazu kommt noch, dass viele der Elemente, die die Usability einer Webseite verbessern können, wie zum Beispiel Drag & Drop Operationen, Mouseovers und das Bewegen von Elementen, viel einfacher zu realisieren sind, da diese Funktionalitäten schon vorhanden und direkt einsetzbar sind.

Weiterhin gibt es Schlüsselwörter wie Streaming und Flexibilität. Ganze Teile einer Flash-Anwendung können je nach Bedarf nachgeladen werden; Informationen können nachträglich vom Server abgefragt und dargestellt werden. Ein einfaches Beispiel hierfür wäre ein in verschiedene Module aufgeteiltes Gästebuch, das nach Bedarf zum Beispiel das Modul für Bewertungen nachlädt oder die Einträge für die nächste Seite vom Server abfragt und darstellt, wenn der Benutzer dort hin will. Das sind schon mal nicht unbedeutende Vorteile gegenüber DHTML, aber es geht noch weiter: Flash hat auch eine Schnittstelle zu JavaScript ... was damit möglich ist, lasse ich mal so im Raum stehen. Werfen wir lieber erst einmal...

... einen Blick unter die Motorraube.

Flash-Anwendungen/-Animationen werden im Programm selbst entwicklet (ActionScripts können aber auch ausgelagert und eingebunden werden) und in einem speziellen Format gespeichert (.fla). Aus dieser Entwicklungsumgebung heraus werden die funktionsfähigen Flash-Dateien exportiert (.swf – Kürzel für ShockWave Flash). Diese werden mit dem *OBJECT*-Tag (Internet Explorer) und dem *EMBED*-Tag (Netscape) in eine HTML-Seite eingebunden.

ActionScript selbst, Flashs eigene Programmiersprache, beruht weitgehend auf JavaScript: jene, die sich mit dieser Sprache auskennen, werden sich hier schnell zurechtfinden. Mit ActionScript-Objekten wie zum Beispiel dem *Math* oder *String* Objekt sowie mit allen Flash-eigenen Aktionen und Funktionen lassen sich komplexe Anwendungen weitgehend erstellen. Das wirklich Interessante gibt es allerdings seit Version 5: Flashs eigenen DOM-basierten XML Parser. Somit wird die Ausgabe von strukturierten Daten möglich – ideal für Frontends serverseitiger Applikationen... Ein passendes Beispiel wäre hier wieder unser Gästebuch, dessen Einträge mittels PHP von einer beliebigen Datenquelle geholt und in XML vorbereitet werden können (z.B. aus einer Datenbank), an Flash weitergegeben, dort nach Wünschen des Designers dargestellt, mit optimaler Benutzerführung vom Endbenutzer bearbeitet und an den Server zurückgeschickt werden können.

Kommunikationsmöglichkeiten Flash <> PHP

Es gibt mehrere Möglichkeiten, Flash Daten zu übergeben – welche davon man einsetzt, hängt vor allem davon ab, was man

> **Quellcode**
> Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

erreichen will: Flash kann mit einem Server kommunizieren, aber auch über JavaScript Aufrufe mit dem Browser und der HTML-Seite, in der es läuft, austauschen.

Skalare Variablen können zum Beispiel direkt an die URL des Flash angehängt werden, ganz wie beim Eingeben einer URL:

```
gaestebuch.swf?seite=willkommen
```

Sehr weit kommt man damit natürlich nicht, wenn man eine größere Menge an Daten übertragen möchte, da man das Längenlimit der *GET*-Methode beachten muss, und die Variablen können nur einmal (beim Start) gesetzt werden.

Interessanter wird es, wenn man Variablen vom Server abfragen kann – die einfachste Variante dafür ist der ActionScript-Befehl *loadVariables()*. Hier wird eine URL angegeben, von der die Variablen geladen werden. Das kann ein beliebiges Textfile sein, in dem die Variablen URL-Encoded hinterlegt werden oder ein PHP-Skript, das die Variablen URL-Encoded zurückgibt (per *echo*).

Im ActionScript würde man zum Beispiel das PHP Skript so aufrufen:

```
LoadVariables(
"http://www.irgendwo.de/gaestebuch.php?laden=seite1",
                                        "_root","GET");
```

Der erste Parameter ist die URL von der die Variablen geholt werden sollen, der zweite gibt an, in welche Ebene von Flash sie geladen werden sollen, und der dritte, mit welcher Methode die Daten geholt werden sollen – *GET* oder *POST*.

Das PHP-Skript dazu könnte dann so aussehen, um die Variablen *name* und *vorname* für die *seite1* zu senden:

```
switch( $laden )
{
        case "seite1":
                echo "name=bar&vorname=foo"
                break;
}
```

Die zwei Variablen *name* und *vorname* stehen dann in Flash zur Verfügung, sobald die Daten übertragen sind. Um Daten später wieder zum Server zurückzusenden, kann man den ActionScript-Befehl *getURL* verwenden – dabei werden alle im Flash vorhandenen Variablen an eine bestimmte URL über *GET* oder *POST* geschickt.

```
getURL („http://www.irgendwo.de/gaestebuch.php",
                                        „_blank", „POST");
```

Der zweite Parameter dieses Befehls ist, in welchem Browserfenster die URL aufgerufen werden soll.

Auf diese Weisen kann man beliebig viele Informationen vom Server abfragen und wieder zurückschicken, allerdings nur skalare Variablen. Für ein Gästebuch würde diese Methode natürlich nicht ausreichen – da braucht man eine Möglichkeit, die Daten in einem strukturierten Format übertragen zu können. Hier kommt Flashs DOM-basierter XML-Parser ins Spiel. DOM steht für „Document Object Model" – jene die ein wenig mit DHTML zu tun haben, werden das wiedererkennen, da die neue Generation Webbrowser (Internet Explorer 5 / Netscape 6) mittlerweile auch das DOM unterstützt.

Das DOM ist eine Norm, die den Skriptsprachenzugriff auf beliebige Elemente eines Auszeichnugssprachen-Dokuments beschreibt. Im Webbrowser wird diese Norm eingesetzt, um per JavaScript auf HTML zuzugreifen, in Flash wird sie benutzt, um per ActionScript auf XML Daten zuzugreifen. Dafür gibt es in ActionScript das *XML* Objekt. Dieses hat Methoden, um XML einzulesen, zu parsen, zu verändern, vom Server abzufragen und wieder dorthin zurückzusenden...

XML lässt sich sehr schön mit Baumstrukturen darstellen, und das ist eine gute Gelegenheit, die ganze Theorie mal ein bisschen konkreter zu gestalten: das folgende Beispiel ist auch auf der CD-ROM enthalten – es besteht aus einer Flash Datei, die von einem PHP Skript XML anfordert und dieses baumartig darstellt (siehe Abb. 1). Das PHP Skript ist sehr einfach gehalten:

```
echo „<guestbook><name>Superdolles Gästebuch</
        name><author>The Argh</author></guestbook>"
;
```

Das ActionScript, das die Darstellung generiert, sieht so aus:

Abb. 1: Die ausgelesenen Daten als Baumstruktur

```
01 var myXMLObj          =          new XML();
02 myXMLObj.onLoad      =          dataLoaded;
03 myXMLObj.load( "xmltree.php" );
04
05 function dataLoaded( isLoaded )
06 {
07          if( isLoaded )
08                    drawNode( myXMLObj, 15, 70 );
09 }
```

In Zeile 01 wird das XML Objekt erzeugt, anschließend, in Zeile 02, wird die Funktion definiert, die aufgerufen werden soll, sobald alle Daten geladen und geparst sind. In Zeile 03 wird das XML von dem PHP-Skript angefordert. Die Funktion ruft in Zeile 05 eine weitere Funktion, *drawNode()*, auf, die sich darum kümmert, die Baumstruktur zu erzeugen (siehe Listing 1).

Hier kommt das DOM zum Vorschein mit den Nodes und den verschiedenen Funktionen, um durch die XML-Elemente wie *childNodes*, *firstChild*, *parentNodes* usw zu gehen. Hier eine kurze Beschreibung der Schlüsselzeilen des Beispiels:

- Z. 05: Die Funktion bekommt das XML Objekt mit den Daten in der Variable Node übergeben.
- Z. 07: Node-Typen: 1 für Elementnode, 3 für Textnode.
- Z. 12: Das grafische Element für den Elementnode wird erstellt und in Z. 13 und 14 auf seine Position gesetzt.
- Z. 15: Dem grafischen Element wird der Name des Elements übergeben.
- Z. 22: Die Hauptschleife, die durch Neuaufrufe der Funktion *drawNode()* die XML Daten durchläuft.

Bis jetzt wurden immer nur Daten vom Server geladen, aber wirklich interessant wird es, wenn man auch Daten zurückschicken kann – dafür gibt es in Flash den Befehl *xml.sendAndLoad()*, der auch genau das macht, was er sagt: XML zum Server schicken und die Antwort abrufen. Damit kommen wir dem Gästebuch wieder einen Schritt näher, da man so zum Beispiel weitere Einträge anfordern oder neue Einträge abschicken kann, die vom Server validiert werden, und der Server schickt die Antwort zurück.

Einzige Hürde bei dieser Methode ist, dass die Daten, die von Flash geschickt werden, in den POST Daten enthalten sind – auf einem Apache-Server muss man diese also aus der Variable $HTTP_RAW_POST_DATA entnehmen. So kann man beliebig XML hin und her senden, was für eine Applikation wie ein Gästebuch oder gar einen Flash-Shop vollkommen ausreicht. Möchte man allerdings eine übertragungsintensive Applikation wie zum Beispiel einen Chat – wo es schlecht wäre, jedes Mal eine HTTP Verbindung mit dem Server herzustellen – so realisieren, kann man Flashs XML Socket Funktion verwenden. Mit diesem Socket kann eine ständige Verbindung zum Server erstellt werden, über die beliebig viele Daten fließen können.

Das ActionScript, um ein Socket zu erstellen, sieht so aus:

```
myXMLSocket = new XMLSocket();
myXMLSocket.connect( "www.irgendwo.de", 1024 );
```

Die einzige Problematik dabei ist der zweite Parameter beim *connect*, der Port, über den die Verbindung laufen soll – aus Sicherheitsgründen kann dieser nicht kleiner als 1024 sein. Eine experimentelle, dennoch einfach Lösung dafür besteht darin, PHPs Socket Funktionen [3] zu verwenden. Wenn man PHP als CGI Modul verwendet, kann man einen einfachen TCP/IP Server auf einem beliebigen Port laufen lassen, mit dem sich Flash verbinden kann.

Aus einer etwas anderen Kategorie der Kommunikation PHP <> Flash gibt es auch noch XML-RPC – einen Flash Client dafür gibt es auch schon einsatzbereit bei Sourceforge [4]. XML-RPC gibt einem die Möglichkeit, Funktionsaufrufe in XML abzubilden und von einem XML-RPC Server Script ausführen zu lassen und eine Antwort auf die gleiche Weise kodiert wieder zurückzubekommen. Eine Anwendung hierfür wäre zum Beispiel die Funktionalität in einer Web-Applikation wie einem Gästebuch, den XML-RPC Server zu fragen (anhand der Versionsnummer und dem Namen der Applikation), ob ein Update zur Verfügung steht. Der Administrator des Gästebuchs könnte dann im Adminbereich eine Meldung bekommen, dass er sein Gästebuch updaten kann.

Listing 1: Baumstruktur erzeugen

```
01 var depth = 100;
02 var xindent = 38;
03 var yindent = 18;
04
05 function drawNode( node, xpos, ypos )
06 {
07 if( node.nodeType == 1 )
08 {
09  if( node.nodeName != null )
10  {
11   var name = "enode" add depth;
12   duplicateMovieClip( "elementnode", name, depth );
13   setProperty ( name, _x, xpos );
14   setProperty ( name, _y, ypos );
15   eval(name).text = node.nodeName;
16   depth++;
17   xpos = xpos + xindent;
18   ypos = ypos + yindent;
19
20  }
21
22  for( var i=0; i < node.childNodes.length; i++ )
23   ypos    = drawNode( node.childNodes[i], xpos,
                                                ypos );
24
25  return ypos;
26 }
27 else if( node.nodeType == 3 )
28 {
29  var name = "tnode" add depth;
30  duplicateMovieClip( „textnode", name, depth );
31  setProperty ( name, _x, xpos );
32  setProperty ( name, _y, ypos );
33  eval(name).text = node.nodeValue;
34
35  depth++;
36  ypos = ypos + yindent;
37
38  return ypos;
39 }
40 else
41  return false;
42 }
```

Überzeugt?

Das Thema ist sehr umfangreich – man könnte ein ganzes Buch darüber schreiben :) Ich fand es auf jeden Fall interessant genug, um ein paar Anregungen zu verteilen. Ich sehe Flash als geniales Tool, um Web-Applikationen noch intuitiver und einfacher bedienbar zu machen. Ein tolles System ist nichts, wenn es keiner verstehen oder bedienen kann... Flash muss nicht zwangsweise immer für das gesamte Frontend eingesetzt werden – man kann es auch ganz gezielt nur an bestimmten Stellen verwenden, an denen es Sinn macht, wie zum Beispiel eine Baumnavigation, um durch die Einstellungsmenüs der Applikation zu gehen. In DHTML würde man für das gleiche wieder mit Browserunterschieden kämpfen, und die Programmierung dahinter ist ein ganzes Stück aufwändiger. Das Beispiel der Baumstrukturen (Abb. 1) könnte natürlich auch noch erweitert werden – wenn es darum geht, Strukturen darzustellen und zu bearbeiten, ist Flash geradezu ideal, da man Elemente auf die einfachste mögliche Weise per Drag&Drop hinzufügen, verändern und löschen kann (z.B. Bearbeiten der Seitenstruktur einer Webseite, Workflowmanager in einem CMS...). Ich selber bin überzeugt, und setze es auch schon ein, wo es nur geht – ich verzichte nie auf eine Gelegenheit, HTML aus dem Weg zu gehen...:)

Sebastian „The Argh" Mordziol ist seit 4 Jahren als Web Designer tätig und ist vor 2 Jahren in die PHP-Applikationsentwicklung gerutscht. Er ist zurzeit bei der Karlsruher Web Design Agentur Metrix tätig und nebenbei Mitgründer der PHP Application Tools Seite. Spezialisiert hat er sich auf Applikationsdesign und Benutzerführungsoptimierung.

Links & Literatur

[1] LibSWF:
www.php3.de/manual/de/ref.swf.php

[2] Ming:
www.php3.de/manual/de/ref.ming.php

[3] Sockets:
www.php3.de/manual/de/ref.sockets.php

[4] Sourceforge Flash XML-RPC Client:
www.sourceforge.com/projects/xmlrpcflash/

David gegen Goliath

von Frederik M. Kraus

MySQL mit InnoDB auf der Überholspur?

Jeder professionelle PHP Entwickler stellt sich mehrfach täglich die Frage: „Wie kann ich Daten am besten speichern?". Die häufigste Antwort auf diese Frage ist MySQL. Nur, was machen, wenn es sich nicht mehr um einfache Webapplikationen sondern komplexe Anwendungen handelt und Funktionen wie Transaktionen, „Referentielle Integrität" oder erhöhte Performance unabdingbar sind?

Spätestens zu diesem Zeitpunkt werden die Rufe nach großen Datenbanksystemen wie Oracle laut. Bisher haben allein der finanzielle und administrative Aufwand davon abgehalten, diese bei solchen Anforderungen einzusetzen und somit zur Suche nach anderen Lösungen gezwungen. Mit InnoDB muss sich MySQL auch vor den ganz Großen nicht mehr verstecken und kann durchaus überzeugen.

Als der Finne Heikki Tuuri mit der Entwicklung von InnoDB anfing, hat er es sich zum Ziel gesetzt, mit InnoDB funktionell gesehen einen Oracle Klon zu schaffen, der erheblich schneller als Oracle selbst ist. Aus diesem Grund wurden viele Oracle Konzepte wie „Multiversioned Concurrency Control", auch bekannt unter der Bezeichnung „Consistent Read", implementiert und teilweise sogar optimiert. Obwohl bis Anfang 2001 InnoDB als eigenständiges Produkt ausgerichtet war, ist sie heute als Tabellentyp in MySQL fast vollständig integriert und kann so neben den gängigen MySQL Tabellentypen koexistieren. Im Folgenden werden wir die umfangreichen Einsatzmöglichkeiten von InnoDB genauer beleuchten und diese anhand von Beispielen leichter verständlich machen.

Installation

Die Installation von InnoDB gestaltet sich sowohl auf Linux als auch Windows recht unkompliziert. Einzige Voraussetzung ist, dass man die MySQL-MAX Distribution verwendet, die Berkeley DB und InnoDB Tabellentypen beinhaltet. Für Windows-Systeme empfiehlt sich die Binary-Distribution. Sollte man den MySQL-MAX Quellcode selbst übersetzen wollen, ist es notwendig, dass die Configure-Anweisung den Ausdruck *--with-innodb* beinhaltet:

```
configure --with-innodb ...
```

Nach einem erfolgreichen Kompilieren bzw. einer erfolgreichen Installation muss nun die MySQL Konfigurationsdatei, meistens unter dem Namen *my.cnf* bzw. bei Windows unter dem Namen *my.ini* zu finden, für InnoDB angepasst werden.

Bei einem Windows NT Rechner mit 128MB RAM und einer 10GB Festplatte empfiehlt sich die Konfiguration in Listing 1.

Zum Vergleich wollen wir jetzt noch einen Blick auf die Konfiguration eines Dual-CPU Linux-Servers mit ca. 1024MB RAM und einem Hardware Raid, das eine Festplattenkapazität von ungefähr 100GB bereitstellt, werfen (siehe Listing 2).

Da InnoDB Verzeichnisse nicht selbst anlegt, sollten Sie sicherstellen, dass alle Verzeichnisse, die in der Konfiguration angegeben werden, existieren bzw. diese mit den entsprechenden Rechten erstellen.

Nach einem Neustart des MySQL Servers wird InnoDB die Datendateien anlegen, was je nach Rechnerleistung einige Zeit in Anspruch nimmt. Um den Status der Datendatei-Erstellung zu verfolgen, empfiehlt es sich beim ersten Start

mit InnoDB, den MySQL-Daemon unter Linux nicht über den *safe-mysqld*-Wrapper oder als Windows-Dienst zu starten, sondern über die Konsole. Sobald MySQL auf der Konsole *InnoDB started* und *mysqld: ready for connections* ausgibt, ist InnoDB erfolgreich installiert und einsatzbereit.

Erste Schritte mit InnoDB

Läuft der MySQL-Daemon, können Sie sich jetzt wie gewohnt mit dem MySQL-Client verbinden und Ihre erste InnoDB Tabelle erstellen. Dafür muss bei der Tabellenerstellung nur der *TYPE=INNODB* gesetzt werden. Sollten Sie nicht jedes Mal den *TYPE* angeben, sondern standardmäßig InnoDB Tabellen erstellen wollen, ändern Sie in der *my.cnf* den Standardtabellentyp auf *default-table-type=innodb*.

```
CREATE TABLE test (a INT NOT NULL, b INT, PRIMARY KEY (a))
                                                TYPE=INNODB;
```

Zu beachten ist, dass MySQL System-Tabellen, wie *user* oder *host*, zwingend im MyISAM-Format vorliegen müssen und unter keinen Umständen in das InnoDB-Format konvertiert werden dürfen. Um bestehende Tabellen in das InnoDB-Format zu konvertieren, verwenden Sie:

```
ALTER TABLE ihretabelle TYPE=INNODB;
```

Probleme können auftreten, wenn die ursprüngliche Tabelle zum Beispiel einen „Fulltextindex“ beinhaltet, der zurzeit noch nicht von InnoDB unterstützt wird, da eine Konvertierung in diesem Fall zu Datenverlust führen kann. Allgemein ist anzumerken, dass InnoDB zwar stabil läuft, jedoch nicht an die Stabilität des langjährig getesteten MyISAM Tabellentyps reichen kann. Die Unterschiede zwischen MyISAM und InnoDB werden wir später genauer untersuchen.

Transaktionen

Häufigstes Anwendungsbeispiel für Transaktionen ist mit Sicherheit die klassische Überweisung einer Bank. Man stelle sich folgendes Szenario vor: Kunde A überweist 100 Euro an Kunden B. Auch im einfachsten Fall sind für eine Kontobewegung dieser Art einige Datenbankanfragen notwendig:

- Lese Kontostand von Kunde A.
- Subtrahiere 100 Euro von Kontostand und speichere neuen Kontostand.
- Lese Kontostand von Kunde B.
- Addiere 100 Euro und speichere neuen Kontostand.

Sollte jetzt zum Beispiel die Verbindung zum Datenbankserver nach der zweiten Abfrage abbrechen, hätte Kunde A 100 Euro weniger auf seinem Konto, Kunde B allerdings immer noch den gleichen Kontostand.

Wie lässt sich nun dieses Problem lösen? Sie werden es schon geahnt haben – durch Transaktionen. Das InnoDB Transaktionsmodell hat das Ziel, die besten Aspekte einer Multiversions-Datenbank mit denen des traditionellen Zweiphasen-Locking zu verbinden. Bei InnoDB geschieht die Sperrung auf Datensatzebene, was im Englischen als „Row-Level Locking“ bekannt ist. Abfragen werden, wie bei Oracle, standardmäßig als „nicht-sperrend“ behandelt.

InnoDB verwaltet alle Benutzeranweisungen als Transaktionen. Wenn *auto-commit* in MySQL verwendet wird, wird jede Anfrage als einzelne Transaktion gewertet. Ist dies nicht der Fall, hat ein Benutzer immer eine Transaktion offen und muss diese entweder mit *COMMIT* oder *ROLLBACK* schließen bzw. eine neue Transaktion beginnen. Mit *COMMIT* werden alle SQL-Anweisungen der aktuellen Transaktion permanent in die Datenbank übernommen und sind ab diesem Zeitpunkt für andere Benutzer sichtbar. Mit der Anweisung *ROLLBACK* werden alle Änderungen der aktuellen Transaktion zurückgesetzt. Zum Beispiel könnte eine Transaktion wie folgt aussehen:

```
set autocommit=0;

SELECT * FROM beispiel WHERE feld ="wert" FOR UPDATE;
UPDATE beispiel SET feld ="neuerwert" WHERE feld ="wert";

COMMIT;
```

Listing 1

```
[mysqld]
#...
#
innodb_data_home_dir = c:\ibdata
#                                   Sie sollten die Größe der Datendateien
#                                   so wählen, dass genug Platz für Ihre Daten
#                                   und Indizes ist. Es ist zur Zeit noch nicht möglich
#                                   die Größe existierender Dateien zu ändern
innodb_data_file_path = ibdata1:2000M;ibdata2:2000M
#                                   Der Buffer-Pool sollte in etwa 60-80% Ihres
#                                   Arbeitsspeichers belegen
set-variable = innodb_buffer_pool_size=70M
set-variable = innodb_additional_mem_pool_size=10M
innodb_log_group_home_dir = c:\iblogs
#                                   innodb_log_arch_dir und innodb_log_group_home_dir
#                                   müssen identisch sein
innodb_log_arch_dir = c:\iblogs
innodb_log_archive=0
set-variable = innodb_log_files_in_group=3
#                                   Die Logdatei Größe sollte in etwa
#                                   der des Buffer-Pools entsprechen
set-variable = innodb_log_file_size=70M
set-variable = innodb_log_buffer_size=8M
#                                   Sollten Sie auf Daten, die jünger als eine Sekunde sind,
#                                   verzichten können, setzen Sie "innodb_flush_log_at_trx_commit=0"
innodb_flush_log_at_trx_commit=1
set-variable = innodb_file_io_threads=4
set-variable = innodb_lock_wait_timeout=50
```

Anzeige

Direkt nach *COMMIT* wird automatisch eine neue Transaktion begonnen. Es ist auch durchaus möglich, dass zwei Überweisungen gleichzeitig auf das gleiche Konto zugreifen und zum Beispiel den Kontostand lesen, um dann später einmal 100 Euro zu addieren und einmal 100 Euro abzuziehen. Problem ist jetzt, dass beide mit dem gleichen alten Kontostand arbeiten. Entweder hat nun der Kunde 100 Euro zu wenig oder 100 Euro zu viel auf seinem Konto. Für dieses Problem gibt es bei InnoDB verschiedene Methoden, einen gelesenen Datensatz zu sperren, das „Locking". Die wichtigsten sollen an dieser Stelle kurz erwähnt werden:

- *SELECT ... FROM ...* : Kein Locking. Schnappschuss der Datenbank wird gelesen
- *SELECT ... FROM ... LOCK IN SHARE MODE*: Alle Ergebnisdatensätze werden für *UPDATE* und *DELETE* Anweisungen anderer Benutzer gesperrt.
- *SELECT ... FROM ... FOR UPDATE*: Alle Ergebnisdatensätze werden exklusiv gesperrt, d.h. für andere Benutzer sind weder *UPDATE*, *SELECT* noch *DELETE* Anweisungen zugelassen.

SELECT ... FROM .. LOCK IN SHARE MODE sollte man auch verwenden, wenn man die aktuellsten Daten verwenden möchte.

In diesem Fall wird intern mit dem *SELECT* so lange gewartet, bis alle offenen Transaktionen, die auf die ausgewählten Daten zugreifen, abgeschlossen sind.

InnoDB erkennt automatisch Blockaden innerhalb von Transaktionen und behebt diese, indem die betroffene Transaktion per *ROLLBACK* auf ihren Ausgangszustand zurückgeführt wird.

Listing 2

```
[mysqld]
#...
#
innodb_data_home_dir=/var/lib/mysql/
#                                   Sie sollten die Größe der Datendateien
#                                   so wählen, dass genug Platz für Ihre Daten
#                                   und Indizes ist. Es ist zur Zeit noch nicht möglich,
#                                   die Größe existierender Dateien zu ändern
#
innodb_data_file_path=ibdata1:2000M;ibdata2:2000M;ibdata3:2000M;ibdata4:2000M;ibdata5:2000M;ibdata6:2000M
#
#                                   Der Buffer-Pool sollte in etwa 60-80% Ihres
#                                   Arbeitsspeichers belegen
set-variable=innodb_buffer_pool_size=400M
set-variable=innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir=/var/lib/mysql/
#                                   innodb_log_arch_dir und innodb_log_group_home_dir
#                                   müssen identisch sein
innodb_log_arch_dir=/var/lib/mysql/
innodb_log_archive=0
set-variable=innodb_log_files_in_group=3
#                                   Die Logdatei Größe sollte in etwa
#                                   der des Buffer-Pools entsprechen
set-variable=innodb_log_file_size=400M
set-variable=innodb_log_buffer_size=8M
#                                   Sollten Sie auf Daten die jünger als eine Sekunde sind
#                                   verzichten können, setzen Sie "innodb_flush_log_at_
                                    trx_commit=0"
innodb_flush_log_at_trx_commit=1
set-variable=innodb_file_io_threads=4
set-variable=innodb_lock_wait_timeout=50
```

Referentielle Integrität

Der Einsatz von „Referentieller Integrität", auch bekannt unter dem englischen Namen „Foreign Key Constraints", lässt sich am einfachsten anhand eines Beispiels erläutern. Mit Hilfe von Foreign Keys kann man verhindern, dass ein Kunde in Tabelle A gelöscht wird, obwohl in Tabelle B Rechnungen für diesen Kunden vorhanden sind.

```
CREATE TABLE kunde(id INT NOT NULL, PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE rechng(id INT, kunde_id INT, INDEX kund_ind
(kunde_id), FOREIGN KEY (kunde_id) REFERENCES
kunde(id)) TYPE=INNODB;
```

Zu beachten ist, dass sowohl für den Foreign Key als auch den Referenz-Key ein Index existieren müssen, da sonst MySQL bei der Tabellenerstellung eine Fehlermeldung liefert und die Tabelle nicht erstellen kann.

Performance

Leider ist es aufgrund der Oracle Lizenzbestimmungen nicht möglich, Benchmarks bzw. Performancevergleiche mit anderen Datenbanksystemen zu veröffentlichen. Es sei nur so viel gesagt: InnoDB kann, was Performance angeht, ohne weiteres mit Oracle mithalten. Der Vergleich von InnoDB und MyISAM ist aber durchaus erlaubt. Getestet wurden einfache *SELECT* und *INSERT* Anweisungen sowie komplexe *SELECT* Anweisungen, die mehrere *JOINS* beinhalten. Eigene Benchmarks sowie Benchmarks von Heikki Tuuri zeigen:

- Einfacher *SELECT*: InnoDB ist 20 Prozent schneller als MyISAM
- Einfacher *INSERT* : InnoDB ist hier ca. 60 Prozent schneller als MyISAM
- Komplexer *SELECT*: Hier ist InnoDB erstaunliche 600 Prozent schneller als MyISAM

Für Windows und Linux weichen diese Werte kaum ab. Ausschlaggebend für eine gute Performance von InnoDB ist die richtige Konfiguration. Hier lohnt es sich, durch Trial-and-Error die für das jeweilige System optimale Konfiguration herauszufinden. Hierbei lassen sich die oben

genannten Beispielkonfigurationen als Grundlage verwenden.

Nachteile von InnoDB

Die fehlende Unterstützung von Volltext-Indizes ist wohl der derzeit größte Nachteil von InnoDB. Allerdings wird laut Heikki Tuuri, dem finnischen Entwickler, dieses Feature in einer der nächsten veröffentlichen Versionen von InnoDB implementiert sein. Viele werden sich fragen, warum InnoDB dann nicht den MySQL Standardtabellentyp vollständig ersetzt. Hierfür gibt es wohl zwei Gründe. Zum einen benötigt InnoDB auf Grund der Multiversions-Kontrolle im Transaktionsmodell und des Row-Level Locking erheblich mehr Speicherplatz. Außerdem ist der InnoDB Tabellentyp bisher bei weitem nicht so genau auf Fehler getestet worden wie MyISAM, schon alleine, weil InnoDB erheblich jünger als MyISAM ist. In der Praxis hat sich InnoDB als stabil und sehr leistungsfähig gezeigt. Unter anderem wird InnoDB produktiv bei Lycos Europe (http://shopping.lycos.de) bzw. Pangora.com eingesetzt.

Fazit

Wenn man Oracle-Features wie Transaktionen und Referentielle Integrität in seinen PHP-Anwendungen verwenden möchte, gleichzeitig auf Performance Wert legt und das Budget vergleichsweise klein ist, setzt man mit InnoDB sicherlich auf das richtige Produkt. Dennoch sollte man bei der Anwendungsentwicklung das Vorhandensein von InnoDB beim Kunden nicht voraussetzen, sondern explizit klären, da, wie erwähnt, InnoDB nicht der Standard MySQL Tabellentyp ist.

Frederik M. Kraus ist als PHP-Freelancer unter anderem für Lycos Europe tätig, engagiert sich in der deutschen PHP-Community und arbeitet an der Entwicklung von XPcm (www.xpcm.com) mit.

Anzeige

Punkt, Linie, Kreis

von Ewald Geschwinde und Hans-Jürgen Schönig

Visualisierung geometrischer Daten mit PostgreSQL und PHP

Viele der derzeit weitverbreiteten Datenbanksysteme sind gänzlich ungeeignet, um geometrische Daten effizient zu speichern und zu verarbeiten. PostgreSQL stellt ein ausgereiftes und durchdachtes System zur Bearbeitung geometrischer Daten zur Verfügung. PHP und PostgreSQL sind ein erprobtes Team zur Visualisierung geometrischer Daten – dieser Artikel gibt einen kurzen Einblick in diese interessante und faszinierende Materie.

Speziell bei geographischen Informationssystemen (GIS) waren kommerzielle Datenbanken lange Platzhirsche und Open Source hat erst langsam Fuß gefasst. Im Schatten dieser kommerziellen Systeme haben sich jedoch Open Source Produkte entwickelt, die als Basis für professionelle und hochverfügbare Systeme herangezogen werden können.

PostgreSQL ist eine unter der BSD Lizenz verfügbare Datenbank, die sich in den vergangenen Jahren entscheidend weiterentwickelt hat. Neben ausgereifter Unterstützung für Transaktionen, Triggers, Embedded Languages (PL/pgSQL, PL/Perl, PL/Tcl), Views und anderen essentiellen Features verfügt PostgreSQL auch über zahlreiche Datentypen zum effizienten Speichern von geometrischen Informationen wie Punkten, Linien, Kreisen oder Streckenzügen. Zur schnellen Abarbeitung von räumlichen Suchen wurden so genannte R-Bäume implementiert, die im Falle von geometrischen Daten signifikante Geschwindigkeitsgewinne gegenüber B+-Bäumen bringen.

Zur Visualisierung von geometrischen Daten kann PHPs Grafikschnittstelle verwendet werden, die stark an die GD Schnittstelle angelehnt ist. In Kombination mit PostgreSQL stellt PHP eine solide Basis zur dynamischen Generierung von Grafik dar.

Starten von PostgreSQL:

Nach der Installation der notwendigen Software auf Ihrem System kann PostgreSQL gestartet werden. Da die Installationsroutinen systemspezifisch sind, empfiehlt es sich, diese in den entsprechenden Manuals zu PHP und PostgreSQL nachzulesen. Zum Starten von PostgreSQL benutzen sie entweder *pg_ctl* oder die Init-Skripte in */etc/rc.d/init.d* (im Falle von Red Hat Linux).

Um PostgreSQL mit *pg_ctl* zu starten, müssen Sie sicherstellen, dass Sie bereits mittels *initdb* einen neuen Datenbankcluster erstellt haben. Im Falle von PostgreSQL ist ein Datenbankcluster kein Set von Rechnern, sondern eine Ansammlung von Datenbanken, die sich in einem bestimmten Verzeichnis befindet. Sofern noch kein Datenbankcluster angelegt ist, wird es *pg_ctl* nicht möglich sein, PostgreSQL zu starten.

Im nächsten Beispiel starten wir PostgreSQL mittels Init-Skript:

```
[root@notebook /root]# /etc/rc.d/init.d/postgresql start
Starting postgresql service:                [ OK ]
```

Sofern kein Fehler aufgetreten ist, ist PostgreSQL nun aktiv. Im nächsten Schritt können sie mittels *createdb* eine neue Datenbank anlegen:

```
[postgres@notebook postgres]$ createdb linuxdb
CREATE DATABASE
```

Anlegen von Tabellen

Um sich zur Datenbank zu verbinden, kann ein Frontend namens *psql* benutzt werden. Für Informationen über psql empfiehlt es sich, die Hilfetexte der Software zu lesen, die mit *psql --help* sehr leicht abzufragen sind.

Versuchen wir das Frontend zu starten:

```
bash-2.04$ psql linuxdb
Welcome to psql, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help on internal slash commands
      \g or terminate with semicolon to execute query
      \q to quit

linuxdb=#
```

PostgreSQL unterstützt ein Set von geometrischen Datentypen: *point* (Punkte bestehend aus x- und y-Koordinate),

box (Rechteck, das durch zwei gegenüberliegende Punkte definiert wird), *circle* (ein durch Mittelpunkt und Radius definierter Kreis), *line* (Linie bestehend aus einem Anfangs- und einem Endpunkt), *lseg* (Liniensegment) und *path* (Streckenzug bestehend aus einer Sequenz von Punkten).

Wie in fast jeder auf SQL basierenden Datenbank, können neue Tabellen mittels *CREATE TABLE* angelegt werden. Im nächsten Beispiel wird eine Tabelle namens *mybox* angelegt, die zum Speichern von Rechtecken dient:

```
linuxdb=# CREATE TABLE mybox (data box);
CREATE
```

Sofern es keine Problem mit Userrechten gibt und kein Syntaxfehler auftritt, wurde die Tabelle erfolgreich angelegt. Im nächsten Schritt werden zwei Datensätze in die Tabelle eingefügt:

```
linuxdb=# INSERT INTO mybox VALUES ('(1,1),(5,3)');
INSERT 10033820 1
linuxdb=# INSERT INTO mybox VALUES ('(4,3),(6,6)');
INSERT 10033821 1
```

Wenn kein Fehler aufgetreten ist, enthält die Tabelle nun zwei Werte, die einfach abgefragt werden können:

```
linuxdb=# SELECT * FROM mybox;
  data
-------
(5,3),(1,1)
(6,6),(4,3)
(2 rows)
```

Ein Rechteck definiert sich durch zwei gegenüberliegende Punkte.

PostgreSQL unterstützt eine Reihe von Operatoren, die jedoch nur für geometrische Datentypen definiert sind. Diese Operatoren sind Kernfunktionen von PostgreSQL und erlauben effiziente Berechnungen direkt innerhalb der Datenbank. Hier ist ein kleines Beispiel:

```
linuxdb=# SELECT * FROM mybox WHERE data && '(1,1),
                                        (2,2)'::box;
  data
-------
(5,3),(1,1)
(1 row)
```

Listing 1

```
<?php

# Anlegen eines Bildes und Allozieren von Farben
header („Content-type: image/png“);
$im = @ImageCreate (300, 300)
 or die („Kann Bild nicht anlegen“);

$background_color = ImageColorAllocate
                                ($im, 255, 255, 255);
$rectcolor = ImageColorAllocate ($im, 200, 20, 100);
$linecolor = ImageColorAllocate ($im, 0, 0, 0);

# Verbinden zur Datenbank
$dbh = pg_connect(“dbname=linuxdb user=postgres“);
if (!$dbh)
{
 exit (“ein Fehler ist aufgetreten<br>“);
}

# Selektieren von Rechtecken
$sql = “SELECT data FROM mybox“;
$result = pg_exec($dbh, $sql);
$rows = pg_numrows($result);

# Durchgehen des Ergebnisses
for ($i = 0; $i < $rows; $i++)
{
 $data = pg_fetch_array ($result, $i);
 $mybox = new box($data[„data“]);
 imagefilledrectangle($im, $mybox->x1,
                                $mybox->y1,
  $mybox->x2, $mybox->y2, $rectcolor);
}

# Selektieren von Rechtecken
$sql = „SELECT data FROM mylseg“;
$result = pg_exec($dbh, $sql);
$rows = pg_numrows($result);

# Durchgehen des Ergebnisses
for ($i = 0; $i < $rows; $i++)
{
 $data = pg_fetch_array ($result, $i);
 $mylseg = new lseg($data[„data“]);
 ImageLine($im, $mylseg->x1, $mylseg->y1,
  $mylseg->x2, $mylseg->y2, $linecolor);
}

ImagePng ($im);

# Klasse zum Parsen von Liniensegmenten
class lseg
{
 var $x1;
 var $y1;
 var $x2;
 var $y2;

 #Konstruktor
 function lseg($string)
 {
  $faktor = 20;
  $string = ereg_replace(“\[*\(*\)*\]*“, “”, $string);
  $tmp = explode(“,“, $string);
  $this->x1 = $tmp[0] * $faktor; $this->y1 =
                                $tmp[1] * $faktor;
  $this->x2 = $tmp[2] * $faktor; $this->y2 =
                                $tmp[3] * $faktor;
 }
}

# Klasse für Parsing von Rechtecken
class box
{
 var $x1;
 var $y1;
 var $x2;
 var $y2;

 #Konstruktor
 function box($string)
 {
  $faktor = 20;
  $string = ereg_replace(“\(*\)*\s*“, “”, $string);
  $tmp = explode(“,“, $string);
  $this->x1 = $tmp[0] * $faktor; $this->y1 =
                                $tmp[1] * $faktor;
  $this->x2 = $tmp[2] * $faktor; $this->y2 =
                                $tmp[3] * $faktor;

  if ($this->x1 > $this->x2)
  {
   $xtmp = $this->x1;
   $this->x1 = $this->x2;
   $this->x2 = $xtmp;

   $xtmp = $this->y1;
   $this->y1 = $this->y2;
   $this->y2 = $xtmp;
  }
 }
}

?>
```

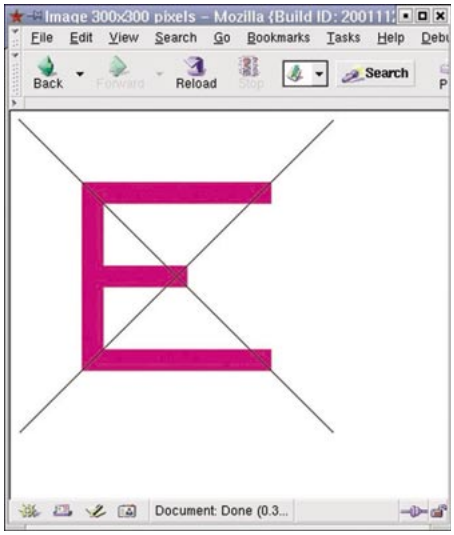


Abb. 1: Visualisierung unserer Beispielsdatenbank

Der *&&* Operator sucht alle Datensätze, die das Rechteck *(1,1), (2,2)* enthalten. In diesem Beispiel wird genau ein Datensatz zurückgegeben. Im nächsten Beispiel wird der Operator <-> verwendet, um den Abstand zwischen den Rechtecken in der Tabelle und dem Rechteck *(1,1), (2,2)* herauszufinden.

```
linuxdb=# SELECT data <-> '(1,1),(2,2)'::box FROM mybox;
  ?column?
------------
1.58113883008419
4.60977222864644
(2 rows)
```

Um zu prüfen, ob ein Rechteck rechts von einem anderen Rechteck ist, kann man den >> Operator benutzen:

```
linuxdb=# SELECT data, data >> '(1,1),(2,2)'::box FROM mybox;
  data       | ?column?
-------------+----------
(5,3),(1,1)  | f
(6,6),(4,3)  | t
(2 rows)
```

Eines der beiden Rechtecke ist rechts des Rechteckes, das in der Abfrage definiert wurde.

PostgreSQL unterstützt eine Menge weiterer Operatoren, die jedoch den Rahmen dieses Artikels sprengen würden. Wenn sie eine komplette Liste aller definierten Operatoren benötigen, empfehlen wir das „PostgreSQL Developer's Handbook" von SAMS Publishing sowie die offiziellen Dokumentationen, die unter www.postgresql.org zu finden sind.

Arbeiten mit PHP – ein Beispiel

PHP kann sehr einfach mit PostgreSQLs geometrischen Datentypen umgehen. Im Wesentlichen ist für jeden Datentypen ein Konstruktor zu implementieren, der den von PostgreSQL retournierten String in ein für Sie angenehmes und sinnvolles Format konvertiert. In nächsten Beispiel sehen Sie ein Programm, das Linien und Rechtecke darstellen kann. Bevor wir uns jedoch den Code dieses Beispiels ansehen, löschen wir die sich in der Datenbank befindlichen Daten und fügen neue Daten ein:

```
linuxdb=# DELETE FROM mybox;
DELETE 2
```

Zwei Datensätze wurden erfolgreich gelöscht. Um größere Mengen an Daten einzufügen, empfiehlt es sich, statt *INSERT* den Befehl *COPY* zu verwenden, da in diesem Fall Transaktions- und Parsingoverhead gespart wird. Weiter empfehlen wir, die Daten in einem externen File zu speichern und dann weiter via Frontend einzulesen; hier das File mit den Daten:

```
CREATE TABLE „mylseg" (
    „data" lseg
);

COPY „mybox" FROM stdin;
(12,4),(3,3)
(4,12),(3,3)
(12,12),(3,11)
(8,8),(3,7)
\.

COPY „mylseg" FROM stdin;
[(0,0),(15,15)]
[(0,15),(15,0)]
\.
```

Am Beginn des SQL Codes wird eine neue Tabelle angelegt, um Liniensegmente zu speichern. In weiterer Folge werden die Daten via *COPY* in die verschiedenen Tabellen eingefügt.

```
[postgres@notebook geo]$ psql linuxdb < data2.sql
CREATE
```

Sollte es zu keinem Fehler gekommen sein, existieren jetzt zwei Tabellen, die Daten enthalten. In nächsten Schritt können wir ein PHP Skript schreiben, das die Daten in der Datenbank visualisiert (siehe Listing 1).

Nach Ausgeben des Headers wird ein Bild mit der Auflösung 300 x 300 angelegt. Sollte das Bild nicht angelegt werden können, bricht PHP die Ausführung des Programms ab. Im nächsten Schritt werden drei Farben alloziert und eine Verbindung zur Datenbank aufgebaut, die in diesem Beispiel *linuxdb* heißt. Für dieses Beispiel nehmen wir an, dass die Datenbank auf der lokalen Maschine läuft und der User namens *postgres* über alle notwendigen Rechte verfügt.

Nach erfolgreicher Authentifizierung werden alle Werte aus *mybox* selektiert und in einer Schleife Datensatz für Datensatz durchgearbeitet. Ziel ist es, alle sich in der Datenbank befindlichen Objekte in der Grafik darzustellen. Für jeden Datensatz wird ein Objekt der Klasse *box* angelegt. Der Konstruktor dieses Objektes dient dazu, den von PostgreSQL zurückgegebenen String zu parsen und in ein weiterverarbeitbares Format zu konvertieren. In diesem Fall werden vier Koordinaten extrahiert und mit einem Faktor multipliziert. Mit Hilfe dieses Faktors können die Daten in der Datenbank auf die gewünschte Größe skaliert werden. In weiterer Folge prüft PHP noch, ob die x-Koordinate des ersten Punktes größer als die des zweiten Punktes ist. Wenn die Bedingung erfüllt ist, werden die Punkte vertauscht – in diesem Fall ist das notwendig, weil PHP das Rechteck sonst nicht anzeigt.

Nach dem Abarbeiten der Rechtecke werden Linien aus der Datenbank extrahiert und ebenfalls mittels Konstruktor geparst sowie zur Szenerie hinzugefügt. Jedem der zur Grafik hinzugefügten Objekte wird eine Farbe zugewiesen. In diesem Beispiel ist die Farbe fix vorgegeben – es ist jedoch auch möglich, diese aus der Datenbank zu extrahieren. Zur Verarbeitung von Farben ist es möglich, einen eigenen Datentyp zu definieren, was den Rahmen dieses Artikels sprengen würde. Beispiele für eigene Datentypen finden Sie in den offiziellen Dokumentationen zu PostgreSQL (Beispiel für komplexe Zah-

len) sowie im „PostgreSQL Developer's Handbook" von Sams Publishing (Beispiel für Datentyp *color*; Infos zum Buch finden sich unter www.postgresql.at).

Nachdem alle Objekte zur Grafik hinzugefügt worden sind, wird die Grafik an den Webserver gesendet.

Conclusio

Je nach Art der Anwendung, die Sie implementieren, sind noch eine Reihe zusätzlicher Features einzubauen. Für jeden Datentyp ist ein Konstruktor oder irgendeine andere Funktion, die das Parsing übernimmt, zu implementieren. Auf diese Weise ist es ohne großen Aufwand möglich, geometrische Daten schnell und effizient zu verarbeiten.

Durch die internen Strukturen von PostgreSQL und das Set von Operatoren, das definiert ist, ist es auch möglich, einfache geometrische Berechnungen direkt mit Hilfe der Datenbank durchzuführen.

Hans-Jürgen Schönig ist Autor von Büchern über PostgreSQL (www.postgresql.at). Seine Firma Cybertec Geschwinde & Schönig OEG bietet seit Jahren Support und Schulungen für PostgreSQL und Unix Datenbanken.

Literatur & Links

Offizielle Seite von PostgreSQL: www.postgresql.org/

Kommerzielle Supporter für PostgreSQL: www.at.postgresql.org/users-lounge/commercial-support.html

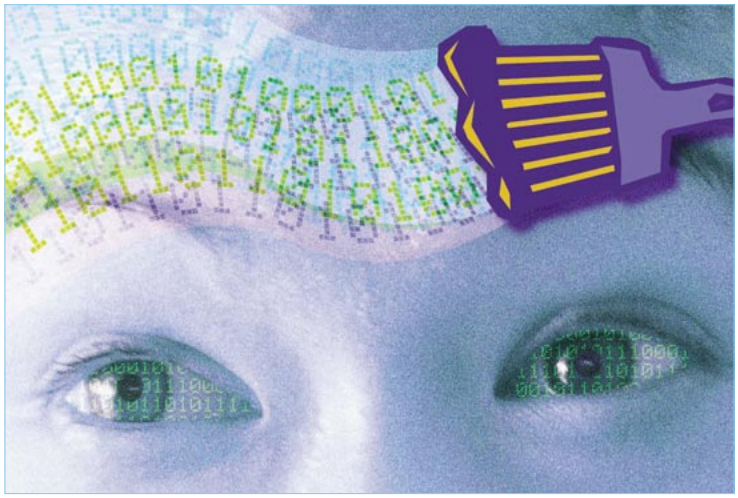
Offizielle Seite von PHP: www.php.net

Die offiziellen Dokumentationen zu PostgreSQL: www.postgresql.org/idocs

„PostgreSQL Developer's Handbook", von Ewald Geschwinde und Hans-Jürgen Schönig, SAMS, 2001

„PostgreSQL: Introduction and Concepts", von Bruce Momjian, Addison Wesley, 2000

Anzeige



Nichts leichter als das ...?

von Stefan Priebsch

PHP datenbankunabhängig programmieren

Es gibt heute kaum eine ernstzunehmende Datenbank, die von PHP nicht unterstützt wird. Eine einheitliche Datenbankschnittstelle wird allerdings oft schmerzlich vermisst. Hier lernen Sie, wie man Anwendungen erstellt, die mit unterschiedlichen Datenbanken zusammenarbeiten, und an welche Grenzen man dabei stößt.

Es klingt auf den ersten Blick so einfach und verlockend: im PHP-Code werden Funktionsaufrufe, die mit *mysql_* beginnen, nach Bedarf durch solche ersetzt, die mit *odbc_* oder *pg_* beginnen, und schon arbeitet die Anwendung mit einer anderen Datenbank zusammen. Also kapselt man von vornherein alle Datenbankfunktionen in Schalenfunktionen oder in einer Klasse, ersetzt diese jeweils nur an einer Stelle und die Anwendung ist auf eine andere Datenbank portiert. Leider ist es nicht so einfach. Die Unterschiede zwischen den einzelnen Datenbanken sind nämlich viel gravierender, als es auf den ersten Blick erscheinen mag. Wer bei MySQL oder PostgreSQL gerne mit assoziativen Arrays arbeitet, wird in ODBC vergeblich eine Funktion suchen, die assoziative Arrays liefert. MySQL-Nutzer, die ein *blob* einfach per *INSERT* in die Datenbank schreiben, werden unter Oracle mit Schrecken feststellen, dass zunächst eine Art Platzhalter erzeugt werden muss, um ein *blob* in die Datenbank zu schreiben. Einmal ganz abgesehen davon, dass jede Datenbank schon beim Verbindungsaufbau einen individuellen Parametersatz verlangt. Oracle verlangt einen „Datenbanknamen“, der SQL Server verlangt einen „Servernamen“, ODBC einen „DSN-String“ und PostgreSQL gar einen „Connect-String“.

Haben die PHP-Entwickler also geschlampt, weil sie keine einheitliche Datenbankschnittstelle definiert haben? Mit Sicherheit nicht. Datenbankanbindungen sind nicht Bestandteil des Sprachkerns von PHP, sondern in einzelnen Erweiterungen realisiert. Deshalb benötigt man beim Übersetzen von PHP Schalter wie *—with-mysql* oder muss, wenn man PHP nicht selbst übersetzt, zur Laufzeit Erweiterungen in der *php.ini* aktivieren (abgesehen von der Unterstützung für MySQL und ODBC, die in den meisten PHP-Binaries standardmäßig enthalten und damit aktiviert sind). Da beim Entwurf von PHP noch nicht feststand, welche Datenbankanbindungen überhaupt einmal benötigt, geschweige denn realisiert werden würden, wird klar, dass allein deshalb zu diesem Zeitpunkt schwerlich eine allgemeingültige Datenbankschnittstelle definiert werden konnte.

Der Entwickler muss also damit leben, jede Datenbank über spezifische PHP-Funktionen anzusprechen. Als Alternative hierzu bietet sich der Einsatz einer Datenbankabstraktionschicht an. Hier werden die unterschiedlichen Funktionen wie oben beschrieben gekapselt und der Anwendung selbst einheitliche Datenbankfunktionen angeboten. Bekannte Vertreter sind PHPLIB, Metabase, PHP DB, AbstractDB, ADODB, DAL oder PEAR::DB. Eine Abstraktionsschicht macht eine Anwendung aber noch lange nicht datenbankunabhängig.

SQL ist nicht gleich SQL

Alle gängigen Datenbanken unterstützen mehr oder weniger den SQL92-Standard, jede einzelne bietet aber auch proprietäre Erweiterungen an, die dem Entwickler das Leben leichter machen sollen. Bekannte

Erweiterungen sind etwa *LIMIT* in MySQL und automatisch vergebene Sequenznummern, die von verschiedenen Datenbanken unterstützt werden. Mit *LIMIT* kann man bei MySQL die Anzahl der Ergebnisdatensätze für eine Abfrage einschränken. Richtig eingesetzt kann man damit in einer Anwendung viel Performance gewinnen, da nicht unnötig große Abfrageergebnisse verarbeitet werden müssen. Andere Datenbanken haben dieses Feature nicht oder nur eingeschränkt. So kann man im SQL-Server ein Abfrageergebnis nur mit *TOP* auf die ersten Datensätze einschränken. Bei Oracle sucht man ein solches Feature zunächst vergeblich.

Proprietäre Erweiterungen sind ein zweischneidiges Schwert. Verwendet man die Erweiterungen, kommt man zwar schneller zum Ziel, muss die Anwendung portiert werden, unterstützt die andere Datenbank die verwendeten Erweiterungen im Allgemeinen nicht. Mancher Entwickler fasst proprietäre Erweiterungen deshalb als eine Form der Kundenbindung an Datenbankhersteller auf. Tatsächlich müssen wir den Datenbankherstellern aber auch dankbar sein, dass an verschiedenen Stellen Erweiterungen des SQL92-Standards vorgenommen wurden: Der Standard von 1992, mittlerweile schon etwas in die Tage gekommen, sieht keine Unterstützung der wichtigen *blobs* (binary large objects) vor. *blobs* benötigt man, um gif-, jpeg- oder andere (besonders größere) binäre Datenmengen in der Datenbank zu speichern. Auch zur Speicherung von großen Textmengen, wenn *varchar* nicht mehr ausreicht, braucht man *blobs*.

Aber sogar bei den im SQL92-Standard definierten Datentypen ist es mit der Unterstützung nicht weit her: von den bekannteren Datenbanken scheint lediglich PostgreSQL alle im Standard definierten Datentypen zu unterstützen. Kaum eine Datenbank unterstützt den Typ *boolean* und der SQL-Server und Sybase kennen kein *date*. Es empfiehlt sich in der Praxis, sich auf die wenigen Datentypen zu beschränken, die von allen gängigen Datenbanken unterstützt werden [2]. Das sind *char*, *int*, *float* und *varchar*, was für die meisten Anwendungen ausreichend sein sollte. Datumsangaben werden dann zwar nicht direkt unterstützt, aber für „handelsübliche“ Datumsangaben zwischen 1904 und 2038 kann man einen *unix timestamp* [3] als *int* in der Datenbank speichern.

Weniger ist mehr

Was für die Datentypen gilt, bestätigt sich erst recht beim Erstellen von SQL-Statements. Hier müsste für jedes einzelne Statement geprüft werden, ob es tatsächlich SQL92-konform und so geschrieben ist, dass jede Datenbank damit zurechtkommt. Das ist zeitaufwändig und in der Praxis kaum machbar; als Alternative können für jede Datenbank individuelle SQL-Statements erzeugt und beispielsweise in einer Klasse pro Datenbankhersteller gekapselt werden. Fehlende Datenbankfunktionalität kann dann durch in PHP programmierte Funktionalität ersetzt werden.

Anstatt SQL-Befehle mit proprietären Erweiterungen zu verwenden, wie zum Beispiel *SELECT * FROM test LIMIT 0, 10* für MySQL, kann eine Abstraktionsschicht Methoden bereitstellen, mit denen vorab für den nächsten auszuführenden SQL-Befehl Einschränkungen gemacht werden können. Abhängig vom Datenbanktyp kann dann der SQL-Befehl – im Beispiel wäre das *SELECT * FROM test* – vor der Ausführung dann von der Abstraktionsschicht verändert werden. Für MySQL würde man lediglich *LIMIT 0, 10* anhängen, für den SQL-Server würde man *TOP 10* anhängen. Für kompliziertere Fälle, wenn etwa nicht die „obersten“ Datensätze gewünscht sind oder für andere Datenbanken, ist freilich etwas mehr PHP-Code notwendig. PEAR::DB implementiert *LIMIT* für Oracle dadurch, dass der ursprüngliche SQL-Befehl durch eine verschachtelte Anfrage ersetzt wird, die genau die gewünschten Ergebniszeilen liefert. Man sieht an diesem Beispiel, dass selbst „einfache“ SQL-Befehle eine größere Lawine auslösen können, wenn eine Anwendung datenbankunabhängig sein soll.

Ein anderer typischer Problemfall sind automatisch von der Datenbank vergebene Sequenznummern. Diese werden oft verwendet, um Datensätze eindeutig zu identifizieren. Die eindeutige Vergabe von Datenbank-Sequenznummern auf PHP-Ebene ist nämlich alles andere als trivial zu lösen, besonders, wenn mehrere Clients bzw. Skripte gleichzeitig auf eine Datenbank zugreifen.

Während die meisten Datenbanken Sequenznummern unterstützen, zeigen sich große Unterschiede im Detail. Während MySQL alles automatisch erledigt und sogar eine Funktion *mysql_insert_id* bereitstellt, mit der die zuletzt erzeugte Sequenznummer abgefragt werden kann, muss unter Oracle schon ein Trigger herhalten, um die Sequenz zu erzeugen, und ein spezieller *SELECT*-Befehl zur Abfrage der Sequenznummer. Wirklich datenbankunabhängig mit Sequenznummern zu arbeiten, ist relativ aufwändig.

Alternative ODBC

Ironischerweise kommt ein guter Ansatz zur Datenbankabstraktion, nämlich ODBC, von Microsoft, denen man ja sonst eher monopolistische Züge zuschreibt. ODBC vereinheitlicht unterschiedliche Datenbanken auf Treiberebene und ist ein Industriestandard, der auf gängigen Systemen und für gängige Datenbanken verfügbar ist. Tatsächlich ist mit ODBC bereits ein großer Schritt in Richtung Datenbankunabhängigkeit getan, zum Beispiel durch den vereinheitlichten Verbindungsaufbau mit DSNs. Der Preis für die Unabhängigkeit ist wieder einmal geringere Performance und Funktionalität, da ODBC eine weitere Abstraktionsschicht zwischen Datenbank und Anwendung einzieht. ODBC definiert eigene Datentypen, darunter auch *blobs*, sodass einige der bereits beschriebenen Probleme aus der Welt geschafft sind, wenn man ODBC verwendet. Allerdings ist ODBC auch nicht gleich ODBC: es gibt drei verschiedene Level, also muss der Entwickler im Einzelnen prüfen, welchen Level der ODBC-Treiber und der ODBC-Client (in diesem Fall PHP) unterstützen.

Doch auch mit ODBC bleibt die Verantwortung für sorgfältig geschriebene SQL-Befehle beim Entwickler. Manche Datenbanken akzeptieren Typinkonsistenzen klaglos, andere sind eher empfind-

lich: MySQL akzeptiert *INSERT INTO test (name, salary) VALUES ('Fred', '100')*, selbst wenn *salary* eine *int*-Spalte ist, denn MySQL führt eine implizite Typumwandlung durch. Andere Datenbanken, etwa der SQL-Server, erwarten eine typkonsistente Abfrage *INSERT INTO test (name, salary) VALUES ('Fred', 100)* – also ohne Anführungszeichen. Vor solchen Feinheiten schützt auch ODBC nicht.

Besondere Vorsicht ist mit reservierten Schlüsselworten geboten. Keine Tabelle oder Spalte darf *SELECT* heißen. Wer könnte auch aus einer Abfrage *SELECT SELECT FROM SELECT* schlau werden? Da unter Umständen im Vorfeld nicht bekannt ist, welche Begriffe in der später eingesetzten Datenbank reservierte Schlüsselworte sind, sollten alle Tabellen- und Spaltennamen mit einem anwendungsspezifischen Präfix beginnen, etwa *sp_*. So lassen sich Datenbanktabellen auch in einer „fremden" Datenbank ablegen, was durchaus von Nutzen sein kann, wenn der Anwender nicht für jede Anwendung eine eigene Datenbankinstanz anlegen kann oder will.

Ein klassischer Anfängerfehler bei SQL-Befehlen sind Zeichenketten, die Hochkommata enthalten. Da in SQL-Befehlen Zeichenketten mit Hochkommata umschlossen werden, gibt es Probleme, wenn die Zeichenkette selbst ein Hochkomma enthält. Das folgende Beispiel wird auch gerne verwendet, um eine der klassischen Sicherheitslücken aufzuzeigen: *INSERT INTO test (name, salary) VALUES ('$Name', $Salary);* wird mit *$Name="'; DELETE FROM test;"* und *$Salary = 0* expandiert zu: *INSERT INTO test (name, salary) VALUES ('''; DELETE FROM test;', 0);*. Die Datenbank würde die erste Abfrage als fehlerhaft zurückweisen und dann die syntaktisch korrekte Abfrage *DELETE FROM test;* ausführen. Um solche Probleme zu vermeiden, gibt es in PHP die Funktionen *addslashes*, die vor „gefährlichen" Zeichen jeweils einen Backslash einfügt. Im Beispiel sollte also *addslashes ($Name)* in den SQL-Befehl eingefügt werden. Die Datenbank erkennt dann, dass die „gefährlichen" Zeichen Bestandteil der Zeichenkette sind und diese nicht abschließen. Unglücklicherweise müssen Hochkommata laut SQL-Standard als doppelte Hochkommata gekennzeichnet („escaped") werden. Während MySQL oder PostgreSQL mit *addslashes* gesicherte Zeichenketten akzeptieren, pocht Oracle auf SQL-Standard-konforme Befehle und erwartet doppelte Hochkommata.

Manche Betriebssysteme und manche Datenbanken sind Case-sensitiv, das bedeutet, zwischen Groß- und Kleinschreibung wird unterschieden. Unix ist ein Beispiel für ein Case-sensitives Betriebssystem, während Windows Case-insensitiv ist. Unter Windows kann man also Groß- und Kleinschreibung in Pfaden, Dateinamen, Datenbanknamen und Tabellennamen bedenkenlos mischen. Doch Vorsicht: wenn Datenbanken wie MySQL auf beiden Betriebssystemen laufen, ist Ärger vorprogrammiert, wenn man von Windows kommt und unsauber gearbeitet hat. *SELECT * FROM test* ist eben nicht immer dasselbse wie *SELECT * FROM TEST*. Es ist daher dringend anzuraten, für jedes Projekt eine „Rechtschreib-Richtline" festzulegen und sich konsequent daran zu halten.

Fazit

Jeder Ansatz, Datenbankunabhängigkeit zu schaffen, kostet Performance und schränkt die verwendbare Funktionalität der Datenbank ein. Datenbankunabhängigkeit wirkt sich auf die gesamte Anwendung aus und bedeutet mitunter erheblich größeren Testaufwand, da allein die Testkonfigurationen mit verschiedenen Datenbanken sehr aufwändig zu erstellen sind. Der Entwickler sollte deshalb im Einzelfall prüfen, ob eine Anwendung tatsächlich datenbankunabhängig sein muss. Für eine Webserver-Backend-Datenbank, die etwa Log-, Session- oder Authentisierungsinformationen speichert, reicht das ohnehin meist vorhandene MySQL aus. Hier ist Performance wichtiger als Portabilität, und da MySQL als Webserver-Backend lizenzfrei und für gängige Plattformen verfügbar ist, fällt die Entscheidung dafür nicht allzu schwer.

Anders bei einer Produktionsdatenbank: die Datenmengen werden mit der Zeit sehr umfangreich, mitunter sind komplexe Tabellenstrukturen und -beziehungen notwendig und nicht zuletzt spielt die vorhandene Systemumgebung beim Kunden eine wichtige Rolle. Hier verlocken „ausgewachsene" Datenbanken wie Oracle mit Unmengen von Features und Möglichkeiten. Ein Großteil der Anwendungslogik kann mit datenbanknahem Code in PL-SQL oder Java realisiert werden, wenn man die nötige Erfahrung mitbringt. Eine solche Lösung ist performant, aber eben stark von der Datenbank abhängig.

Eine hundertprozentig datenbankunabhängige Anwendung zu erstellen, die auch noch mit der „Datenbank von morgen" zusammenarbeitet, wird nie gelingen. Eine Anwendung so zu erstellen, dass sie mit verschiedenen zuvor bekannten Datenbanken zusammenarbeitet, ist mit etwas Aufwand und Kreativität durchaus möglich und sinnvoll.

Dipl. Inform. Stefan Priebsch ist Gründer und Geschäftsführer der e-novative GmbH in München und beschäftigt sich mit innovativen IT-Konzepten und Lösungen, beispielsweise einem Application Server in PHP. Er ist erreichbar unter stefan.priebsch@e-novative.de.

Literatur

[1] C. J. Date: „An Introduction to Database Systems, 7th ed", Addison Wesley Publishing, 1999
Umfassende, theoretisch fundierte Einführung in Datenbanken

[2] www.mysql.com/information/crash-me.php
Feature-Übersicht einzelner Datenbanken und Live-Test

[3] www.heise.de/newsticker/data/hag-15.03.01-002
Zeitdarstellung in Unix-Sekunden

[4] www.phpbuilder.com/columns/banahan20010720.php3
Sehr guter Artikel von Mike Banahan über die Erfahrungen mit Cross-Plattform, Cross Database-Entwicklung in PHP

Daumenkino

von Sebastian Nohn

SiteThumb – Ein System zur Generierung von Websitepreviews

Mitte 2001, als die NeT&Trade GmbH die erste Version ihres hauseigenen Content-Management-Systems in Betrieb nahm, standen wir vor dem Problem, Vorschaubilder der erzeugten Seite erzeugen zu müssen. Nach einigen ernüchternden Experimenten mit HTML2PS und einigen anderen Tools entschlossen wir uns, das Problem mit Hilfe der COM-Funktionen [1] und einem realen Browser anzugehen. Nach den ersten Tests stand dann fest, dass diese Lösung mehr als zufriedenstellend funktionierte und wir schufen auf den erarbeiteten Grundlagen ein eigenständiges Produkt Namens SiteThumb.

Die COM-Klasse

Neben den „normalen" Funktionen stehen die verschiedenen COM-Funktionen auch in einer Klasse zur Verfügung, die dem Programmierer einiges an Tipparbeit ersparen und zudem eine objektorientierte Herangehensweise bei der Programmierung von COM-Objekten zulassen. Erlaubte Parameter sind der Name des COM-Objektes, das aufgerufen werden soll, der DCOM-Server sowie die Codepage, die verwendet werden soll, um PHP-Zeichen in Unicode-Zeichen zu wandeln und diese auch wieder zurückzuverwandeln.

Ein Codebeispiel für die Nutzung der COM-Klasse finden Sie in Listing 1.

COM-Objekte und deren Eigenschaften herausfinden

Leider liefern die wenigstens Programme eine Dokumentation ihrer COM-Objekte und deren Eigenschaften. Microsoft hat die eigenen Produkte im Microsoft Developer Network [2] dokumentiert, für Fälle, in denen das nicht zutrifft, sollte man sich einen COM-Browser besorgen. Die meisten integrierten Entwicklungsumgebungen für Windows bringen einen solchen mit sich (MS Visual C++ zum Beispiel). Es gibt jedoch auch einige freie Standalone-Tools, die zudem meist noch einen erweiterten Funktionsumfang haben. Als ein Beispiel wäre der ScriptingSpy [3] zu nennen.

Auslieferung und Verwaltung der Daten

Als Datenbanksystem wählten wir MySQL [4], da es schnell und einfach zu installieren und administrieren ist und zudem Auto-Increment-Felder vorhanden sind, die uns die Arbeit um einiges erleichtern. Die Datenbank selbst besteht lediglich aus einer Tabelle mit drei Feldern: einer eindeutigen numerischen ID, der dazugehörigen URL sowie einem Unix-Timestamp, in dem die letzte Aktualisierung festgehalten wird.

Fordert ein Webbrowser nun einen Screenshot an, so wird dem serverseitigen PHP-Skript die URL übergeben. Also zum Beispiel *http://preview.nettrade.de/preview/bild.php?url=http://www.entwickler.com/*. Das PHP-Skript schaut nun in der Datenbank nach, ob diese URL bereits bekannt ist. Wenn nicht, wird diese in die Datenbank geschrieben, der Timestamp wird auf *NULL* gesetzt. Für den Fall, dass die URL bekannt ist, schickt das Skript die entsprechende Datei aus dem Dateisystem an den Browser. Schlägt dieses fehl, wird der Timestamp ebenfalls geleert.

Der Webserver

Die ersten Tests mit dem Apache-Webserver verliefen positiv. Im Wirkbetrieb zeigten sich jedoch sehr schnell die Limitierungen des Apache, der für jede Anfrage einen eigenen Kindprozess startet. Die Lösung bestand im thttpd [5], der bereits im Zusammenhang mit der IRCG-Technologie [6] seine Leistungsfähigkeit unter Beweis stellen konnte. Dadurch konnte die Load bei 100 pro Sekunde ausgelieferten Dateien

Listing 1

```
<?php
        // Instanziiere Word
        $word = new COM(„word.application");
    // Bringe Word in den Vordergrund
        $word->Visible = 1;
        // Ein leeres Dokument oeffnen
        $word->Documents->Add();
        // Text in das Dokument schreiben
        $word->Selection->TypeText(„Das ist ein Test");
        // Das Dokument speichern
        $word->Documents[1]-
>SaveAs(„eintest.doc");
        // Word schliessen
        $word->Quit();
            // Das Objekt freigeben;
            $word->Release();
            $word=null;
?>
```

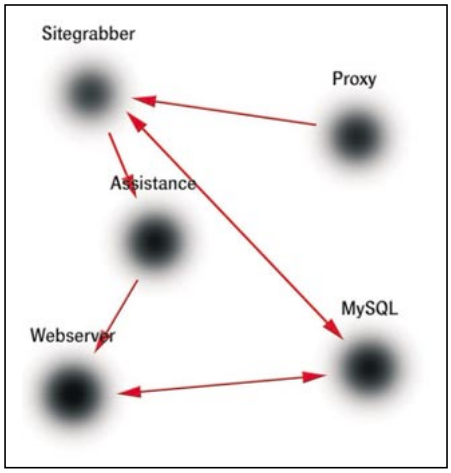


Abb. 1: Schematischer Aufbau des Systems

pro Sekunde von über 5 auf unter 0,2 gedrückt werden.

Viele kleine Dateien im System ablegen

Das Sitethumb-System erzeugt sehr schnell hundertausende von kleinen Dateien zwischen 5 und 30 Kbyte. Zunächst speicherten wir diese in einer flachen Verzeichnisstruktur, doch ab etwa 1000 gespeicherten Dateien traten große Performanceeinbrüche auf. Die Ursache war schnell gefunden, sie liegt in der Organisation des Linux(file)systems. Dieses hat ab 1024 Einträgen in einem Verzeichnis einen erhöhten Verwaltungsaufwand. Die Lösung für dieses Problem ist recht simpel: Die Dateinamen werden in Tripel zerlegt, das letzte Tripel bildet den Dateinamen, die Tripel davor stellen jeweils ein Verzeichnis dar:

```
000/000/001.jpg
[...]
000/000/999.jpg
000/001/000.jpg
000/001/001.jpg
[...]
```

So werden die Limitierungen des Dateisystems zwar nicht optimal ausgenutzt, der Programmcode wird jedoch simpler und ist somit schneller auszuführen. Eine Milliarde speicherbare Bilder sollten zunächst auch reichen. Sollte eines Tages diese Grenze gesprengt werden (was nicht anzunehmen ist), lässt sich dieses System jedoch recht einfach skalieren. Nach diversen Benchmarks stellte sich zudem XFS von SGI als das optimale Dateisystem für unsere Zwecke heraus. Bisher wird dieses zwar noch nicht mit dem Kernel ausgeliefert, auf der Open-Source-Seite von SGI [7] findet sich jedoch immer ein Patch für die aktuellen Kernelversionen.

Die Sitegrabber

Die Sitegrabber, also die Rechner, die die Screenshots der Seiten erstellen, waren zunächst allesamt ausgediente Intel-Workstations vom Pentium 90 bis zum Pentium 233. Diese gelangten jedoch schnell an ihre Grenzen, da pro Rechner durchschnittlich nur zwischen vier und fünf Screenshots pro Minute geschossen werden konnten. Inzwischen übernimmt ein Rack aus 10 1HE-Computern mit aktueller Ausstattung diese Aufgaben. Die Rechner ziehen die Seiten über einen selbst programmierten Proxy-Server, der aktive Skriptkomponenten und Banner herausfiltert, um ein Überfluten der Maschinen mit Popups zu verhindern. Nachdem der Ladestatus einer Seite positiv überprüft wurde, ruft das PHP-Skript ein Kommandozeilentool zum Grabben von Screenshots auf. Dieses speichert das Bild in einem Temporärverzeichnis, die Datei wird per FTP auf einen Linux-Rechner verschoben, anschließend im lokalen Dateisystem der Windows-Maschine gelöscht, und der nächste Job wird aus der Datenbank gezogen. Der Linux-Rechner übernimmt die Datei, skaliert sie mit Hilfe von NetPBM [8] in verschiedene Größen und wandelt die Bilder in das JPEG-Format. Anschließend werden sie mit Hilfe von SCP [9] auf den Webserver verschoben.

Einsatzmöglichkeiten und Lizensierungsoptionen

Der Haupteinsatzzweck von SiteThumb sind zurzeit im weitesten Sinne Webverzeichnisse, wobei die größten Datensätze durch Suchmaschinen zustande kommen, sodass hier ein Einsatz weniger sinnvoll erscheint und lediglich Demonstrationszwecken dient. Momentan gibt es drei Lizensierungsmodelle für die Technologie: Beim ersten wird die vorhandene Technik von NeT&Trade komplett genutzt, beim zweiten Modell werden lediglich Sitegrabber von NeT&Trade genutzt, die generierten Screenshots werden über FTP oder auf einer CD-ROM ausgeliefert. Das dritte Lizensierungsmodell sieht eine Unterbringung der gesamten Technologie in den Räumen des Kunden vor. Hierbei handelt es sich um ein abgeschlossenes Rack, das an die Schnittstellen des Kunden angepasst wird und in regelmäßigen Abständen von NeT&Trade aktualisiert und gewartet wird.

Resümee

Im Nachhinein hat sich herausgestellt, dass die Technologie auf Windows-Seite mehr Pflege erforderte als zu Beginn kalkuliert, da das praktizierte „Powersurfen“ nach und nach so viele Internet-Explorer-Leichen im Hauptspeicher zurücklässt, dass eine tägliche Kontrolle der Rechner nötig ist. Die Datenbank enthält zur Zeit etwa 150.000 URLs, etwa 1,4 GByte Bildmaterial wird vorgehalten. Es hat sich in Klickanalysen gezeigt, dass Seiten, von denen Screenshots vorliegen, häufiger angeklickt werden. Dies stellt eine von vielen Möglichkeiten dar, die Kosten für die Nutzung des Dienstes unmittelbar zu refinanzieren, aber auch die höhere User-Return-Rate durch eine erhöhte Nutzerbefriedigung ist nicht zu unterschätzen. So konnte die Seite City-Review.de nach Einführung des SiteThumb-Systems 20 Prozent mehr wiederkommende Besucher verzeichnen.

Sebastian Nohn, Jahrgang 1979, beschäftigt sich seit seinem zehnten Lebensjahr mit Computern. Seit 1996 plant und programmiert er Internetauftritte zunächst in Cold Fusion, seit 1999 in PHP. 1997 gründete er die dreaMEDIA GbR, die noch 1997 in die NeT&Trade GmbH aufging, und realisierte dort unter anderem die Portale jura.de und phpCrawler.de. Zurzeit ist Sebastian Nohn bei der Deutschen Telekom AG beschäftigt.

Links

[1] www.phpcrawler.de/com
[2] msdn.microsoft.com/
[3] www.wininfo.de/
[4] www.mysql.org/
[5] www.acme.com/thttpd/
[6] www.schumann.cx/ircg/
[7] oss.sgi.com/
[8] www.netpbm.org/
[9] www.openssh.org/

Appetizer

von Alexander Meis

Das Kochcommunity-Portal chefkoch.de

Die Idee zu chefkoch.de entstand Ende 1998 aus dem einfachen Gedanken, eine Rezeptdatenbank online zu publizieren und damit eine alltagstaugliche Anwendung für den im Internet bisher wenig beachteten Bereich „Rund ums Kochen" zu schaffen. Diese Plattform wurde von der Firma Pixelhouse Gbr ins Leben gerufen und bis heute betrieben und weiterentwickelt. In der über dreijährigen Entwicklungszeit hat sich chefkoch.de von einer einfachen Rezeptsammlung zu einer der größten Kochcommunitys im deutschsprachigen Internet gemausert.

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

Derzeit verzeichnet chefkoch.de ca. 2.5. Millionen Seitenabrufe pro Monat, hat ca. 15.000 registrierte Benutzer sowie etwa 20.000 Newsletterabonnenten. Die Datenbank beinhaltet über 3.000 einzigartige Rezepte, unzählige Tipps und Tricks sowie Informationen rund ums Kochen, den Haushalt und die Küche. Das Herzstück der Community ist das sehr gut besuchte Forum, in dem sich Kochbegeisterte aus der ganzen Welt begegnen und austauschen. Hier findet jedermann, vom Anfänger bis zum Kochprofi, Hilfe bei Problemen und Fragen oder einfach Unterhaltung. Dieser Artikel gibt Ihnen einen Einblick in die Entwicklung von chefkoch.de – insbesondere in die Lösungsansätze und Strategien, die in der aktuellen Version 3.0 verfolgt werden.

Chefkoch und Open Source

Bei der Entwicklung von chefkoch.de kommt ausschließlich freie Software aus dem Linuxbereich zur Anwendung, um die Kosten der Projektumsetzung möglichst gering zu halten, nicht von einem Softwarehersteller abhängig zu werden und eine größtmögliche Stabilität, Flexibilität und Anpassungsfähigkeit der eingesetzten Produkte zu erreichen. Linux als Betriebsystem, der Webserver Apache, PHP und die freie Datenbank MySQL bilden die Grundsäulen der für chefkoch.de entwickelten Community. PHP ist hierbei die optimal geeignete Skriptsprache für die Kommunikation mit der MySQL Datenbank, erledigt alle nötigen Arbeiten im Frontend- und Backendbereich und ermöglicht eine sehr einfache, schnelle Implementation komplexer und unterschiedlichster Aufgabenstellungen.

Derzeit laufen Datenbank und Webserver von chefkoch.de auf Red Hat Linux 6.2 bzw. 7.2. mit Apache 1.3.20, PHP 4.0.6 und MySQL 3.23.32. Als Hardware kommen Intel-basierte Webserver der Firma Blueware.de zum Einsatz.

PHPlib

Aus den Erfahrungen mit den bisherigen Versionen von chefkoch.de wurde die Version 3.0 technisch, optisch sowie funktionell einem grundlegendem Facelifting unterzogen und zu einem Comunity Portal ausgebaut. Die wichtigsten zu erreichenden Eigenschaften ließen sich einfach definieren und ergaben folgende Zielsetzung:

- Einfache Pflegbarkeit der Seiten und Inhalte
- Trennung von Programmcode und Darstellungscode
- Optimale Antwortzeiten der Server
- Globale Einstellungen der Serverparameter in allen Programmteilen
- Gute Erweiterungsmöglichkeiten

Weil das Rad an dieser Stelle nicht neu erfunden werden sollte, benutzt chefkoch.de eines der bekanntesten Frameworks im PHP Bereich, PHPlib. Die PHPlib stellte alle für die Version 3.0 nötigen Tools und Klassen zur Verfügung, welche die Realisierung einer Webanwendung erleichtern.

Die Templateklasse ermöglicht eine vollständige Trennung von Programm-

und Darstellungscode. Dies bietet den Vorteil, dass der Programmierer unabhängig vom Designer an der Funktionalität der Seite entwickeln kann, während der Designer am Layout der kompletten HTML Webseite arbeitet und diese dabei so gestaltet, wie sie später einmal aussehen soll, ohne blockweise arbeiten zu müssen. Auch entfällt die lästige und aufwändige Implementation des HTML Codes in den PHP Programmcode. Da das Design und Layout in Templates, einfache HTML Dateien, die mit speziellen Platzhaltern versehen sind, gespeichert ist, kann der Webdesigner auch später Designänderungen vornehmen, indem er einfach die Templates, die von einem Skript verwendet werden, bearbeitet oder austauscht. Die PHPlib Templateklasse bietet die Möglichkeit, einfache Platzhalter für Variablen oder Blöcke innerhalb der Templates zu definieren. Der Platzhalter einer einfachen Variable besteht aus einem frei definierbaren Namen, der in *{}* eingefasst wird. Ein Block besteht aus einem HTML Kommentar, der *BEGIN* am Anfang und *END* am Ende enthält, gefolgt von einem frei definierbaren Blocknamen:

```
...
<!— BEGIN Meinblock —>
<a href="{meinlink}">{meinlinkname}</a>
<!— END Meinblock —>
...
```

Die Verwendung von Templates erfolgt im Skript über die Initialisierung der Templateklasse. Danach werden das Template ausgewählt, der Block initialisiert und die Variablen gefüllt. Am Ende wird das Template durch den Templateparser bearbeitet und danach ausgegeben:

```
...
$t = new Template("$template_path");
$t->set_file("Meinblock", "dastemplate.html");
$seite->set_var("meinlink", "http://www.chefkoch.de");
$seite->set_var("meinlinkname", "Chefkoch.de");
$t->parse("out","Meinblock");
$t->p("out");
...
```

Dies ist nur ein einfaches Beispiel. Die Templateklasse kann auch mehre Templates gleichzeitig verwenden, die ineinander geparst werden können. Auch können Blöcke weitere Unterblöcke enthalten, die ineinander verschachtelt sein können oder je nach Anforderung zu unterschiedlichen Zeiten/Zuständen ausgegeben werden.

Die Datenbankklasse der PHPlib ermöglicht einen vereinfachten Zugriff auf Datenbanken in der Form, dass einmal geschriebene SQL Statements auf mehreren verschiedenen Datenbanken ausgeführt werden können, ohne dass Anpassungen an den Statements erfolgen müssen (vorausgesetzt, die Datenbank unterstützt alle in einem SQL Statement verwendeten Funktionen). Lediglich die Datenbankklasse, die von der PHPlib benutzt wird, muss getauscht werden, was sehr einfach an einer zentralen Stelle in der PHPlib Konfiguration möglich ist. Datenbankklassen für die gängigsten Datenbanken wie MySQL oder Oracle sind bereits in der PHPlib enthalten.

Die wichtigsten Klassen der PHPlib, die auf chefkoch.de Verwendung finden, sind die Klassen *Session*, *Authentification* und *User*.

Die Klasse *Session* dient dazu, jedem Benutzer beim Betreten von chefkoch.de eine eindeutige SessionID zuzuweisen, die ihn während seines gesamten Aufenthalts identifiziert. Auch können userspezifische Variablen in der Session gespeichert werden. Diese Variablen können nach der Speicherung zu jedem beliebigen Zeitpunkt wieder aus der Session gelesen werden. Bei chefkoch.de wird dieses Verfahren verwendet, um z.B. die Suchbegriffe, nach denen ein Benutzer auf den Rezeptseiten oder im Forum sucht, zu speichern, um Sie dem User zu einem späteren Zeitpunkt bei einer erneuten Suche wieder anzuzeigen.

Da die SessionID während des gesamten Aufenthalts auf chefkoch.de nicht verloren gehen darf, wird Sie in einem Cookie clientseitig gespeichert. Chefkoch.de setzt daher im Moment die Benutzung von Cookies voraus. Die Benutzung von Cookies ist zur Arbeit mit der Session nicht zwingend erforderlich. Die SessionID kann auch an die URL angehängt und per *POST* oder *GET* über die Webseite mitgenommen werden. Die Verwendung von *POST* oder *GET* setzt dann voraus, dass die SessionID in jedem Link oder Formular auf der gesamten Webseite enthalten ist. Die PHPlib Session stellt aber einen Mechanismus zur Verfügung, der es dem Programmierer einfach ermöglicht, die SessionID an jeden Link anzuhängen. Wichtig für chefkoch.de ist, dass die Verwaltung der Sessions in der Datenbank und nicht im Filesystem erfolgt. Chefkoch.de besteht aus mehreren physikalischen Webservern, die zu einem Webcluster zusammengeschaltet sind. Wechselt ein Benutzer nun beim Surfen von einem Server zum anderen, so würden die Sessiondaten des Benutzers verloren gehen, wenn diese im Filesystem der Server und nicht in einer zentralen Datenbank gespeichert würden.

Eine Grundvoraussetzung für das Erstellen einer Webcommunity ist die Schaffung einer Account-User-Verwaltung. Chefkoch.de bedient sich hierbei der Klassen *Authentification* und *User* der PHPlib. Die Klassen ermöglichen die Erstellung von Usern, das Vergeben von Userleveln, wie z.B. User, Editor oder Admin, sowie die Steuerung der Seiteninhalte und Funktionen je nach vergebenem Userlevel. Prinzipiell ist es jedem, auch den Surfern, die keinen Account auf chefkoch.de haben, möglich, die meisten Inhalte der Webseite anzuschauen. Möchte der Surfer jedoch eine Frage im Forum stellen oder einen Tipp oder ein Rezept veröffentlichen, muss er über einen Account verfügen.

Diesen Account kann der User sich jederzeit unter Verwendung eines frei wählbaren Usernames und einer funktionierenden eMail-Adresse erstellen. Bei der Erstellung werden die Daten vom User über ein Webformular an den Server übertragen und in die Datenbank eingetragen. Hierbei wird ein Passwort für den Zugang generiert, welches per Mail an den User versendet wird. Anfangs war es so, dass der User auch sein Passwort beim Anlegen des Accounts frei wählen konnte. Dies führte dazu, dass einige Individuen sich veranlasst sahen, den Betrieb des chefkoch.de durch meist wenig sinnvolle bis die anderen User beleidigende Postings im Forum zu stören. Um dem einen Riegel vorzuschieben, wurde das Registrierungssystem auf eine automatische Generierung der Passworte umgestellt, wodurch

Anzeige

nun zumindest die Richtigkeit der eMail-Adresse gesichert ist, was wiederum in den meisten Fällen eine Identitätsfeststellung des Users ermöglicht. Der User kann, nachdem er sich das erste Mal auf chefkoch.de eingeloggt hat, das Passwort selbständig in sein Wunschpasswort ändern.

Um festzustellen, was der Surfer auf den einzelnen Seiten darf, wird auf jeder Seite untersucht, welche SessionID gerade benutzt wird. Da beim Login in der Session gespeichert wird, ob es sich um einen angemeldeten User handelt, kann anhand der Userrechte (Permission) herausgefunden werden, was der User auf der Webseite darf und welche Steuerelemente somit für ihn sichtbar werden. So sieht z.B. ein „Editor" im Forum die Option, Beiträge von Usern zu löschen, die einem „User" verborgen bleibt. Die Vergabe der Permissions an die registrierten Benutzer ermöglicht es somit, die unterschiedlichsten Funktionen direkt in die Webseiten zu integrieren, wodurch sich administrative Aufgaben direkt aus der Webseite lösen lassen und eine Programmierung eigener Frontends zur Administration der Community mit dem dazugehörigen Aufwand entfällt.

Insgesamt lässt sich sagen, dass die Entwicklung von chefkoch.de durch den Einsatz der PHPlib extrem vereinfacht wurde. Auch lassen sich die verwendeten Klassen sehr einfach erweitern, wodurch sie an neue Anforderungen angepasst werden können. Einer Weiterentwicklung und einem Ausbau der Funktionalitäten des chefkoch.de steht somit nichts im Wege.

MySQL

Chefkoch.de setzt als Datenbank MySQL ein, weil es in der Zusammenarbeit mit einer Webanwendung eine hervorragende Performance bietet. Auch ist die Unterstützung von MySQL in allen PHP Versionen sowie in der PHPlib optimal. Ein Nachteil von MySQL ist, dass es keine serverseitige Möglichkeit des Datenbankclustering bietet. Bei der Planung des neuen chefkoch.de wurden daher die Inhaltsbereiche direkt in vier Teile aufgesplittet und in einzelne Datenbanken verteilt, um die Möglichkeit der Lastverteilung auf mehrere physikalische Server zu schaffen. Es gibt die Datenbanken *Userverwaltung*, *Forum*, *Rezepte* und *Tippscout*, deren Dateninhalt jeweils völlig unabhängig von den anderen Datenbanken ist.

Die Forumsdatenbank machte bei der Umsetzung die größten Probleme, weil eine Suche implementiert werden musste, die in allen Beiträgen nach mehreren Begriffen suchen sollte. Das Forum setzt sich im Grundgerüst aus drei Tabellen zusammen. Hier sehen Sie die einfache Beschreibung der Tabellen, eine genaue Beschreibung der Tabellen finden Sie auf der CD.

- Tabelle *Forum*: ForumID, ForumName, ForumBeschreibung
- Tabelle *Forumthreads*: ForumThreadID, ForumID, ForumThreadName, ForumThreadPoster
- Tabelle *Forummessages*: ForumMessageID, ForumID, ForumThreadID, ForumMessagePoster, ForumMessage

Beim Surfen durch das Forum wird auf der Foren-Übersichtsseite die Tabelle *Forum* ausgelesen und dem User angezeigt. Hat sich der User für ein Forum entschieden, klickt er auf einen Link und bekommt die Threads des jeweiligen Forums angezeigt. Hierzu werden die zur Anzeige benötigten Daten zu der entsprechenden, mit dem Click übergebenen ForumID aus der Tabelle *Forumthreads* nach Datum sortiert geholt und ausgegeben. Sollten mehr als 30 Threads in einem Forum vorhanden sein, so werden die Threads 1-30 angezeigt. Durch eine dynamisch aufgebaute Navigationsleiste ist auch der Rest der Threads einsehbar. Möchte der User nun einen Thread lesen, so klickt er auf den entsprechenden Link und erhält eine Übersichtsseite mit allen Messages, die zu diesem Thread gepostet wurden. Die Messages werden anhand der ForumThreadID und der ForumID in der Tabelle *Forummessages* ermittelt.
Dieses System ist auch bei vielen Datensätzen schnell, weil die Datenbank während der Navigation und Anzeige der Daten immer über die Indizes der Tabelle sucht, was im Allgemeinen recht performant ist. Anders sieht es hingegen bei der Suche aus. Um eine Suche zu realisieren, ist es notwendig, das Feld *ForumMessage* der Tabelle *Forummessages* zu durchsuchen. Die hierbei verwendete Anfrage könnte z.B so aussehen:

```
select ForumMessageID,ForumMessages from Forummessages where ForumMessage like '%reis%' and ForumMessage like '%honig%'
```

Diese Anfrage ist zwar mit PHP recht einfach zu generieren, jedoch ist die Antwort des Datenbankserver aufgrund der Verwendung der *like* und % Operatoren nicht besonders schnell. Bei chefkoch.de enthält die Tabelle *Forummessages* derzeit ca. 90.000 Datensätze, auch muss der User nach mehr als zwei Begriffen suchen können. Eine Suche mit *like* und % über die aktuelle Datenbank mit fünf Begriffen, die wie im Beispiel oben mit *and* verknüpft sind, ergibt eine Antwortzeit von ca. 5-10 Sekunden, die absolut unzumutbar ist.
Um dieses Problem zu umgehen, arbeitet chefkoch.de mit einem Schlagwortindex für das Durchsuchen der Foren. Für die Erstellung des Wortindexes muss zuerst jeglicher Foreninhalt verschlagwortet und daraus ein Verzeichnis (Index) erstellt werden. Zur Datenhaltung des Indexes dienen die Tabellen:

- Tabelle *Words*: WordID, Word
- Tabelle *Wordsindocuments*: WordID,ForenID,ThreadID,MessageID
- Tabelle *Badwords*: BadwordID, Badword

Zum Aufbau des Indexes dient ein Programm, das einmal pro Nacht automatisch per *Cron* ausgeführt wird und den Index neu aufbaut. Am Anfang des Skripts werden die Tabellen *Words* und *Wordsindocuments* geleert. Danach werden alle nicht im Index erwünschten Worte (Badwords) aus einer Tabelle geholt und in ein Array geschrieben:

```
$dbb->query("select * from Badwords");
$i=0;
While($dbb->next_record()){
        $badwordarr[$i]=$dbb->f('Badword');
        $i++;
}
```

Dieses Array dient dazu, alle Badwords aus dem Inhalt des Indexes zu filtern. Bad-

words sind z.B. „der“, „die“, „das“ oder auch Schimpfworte oder andere gesperrte Begriffe, die nicht in den Index gelangen sollen.
Nach dem Erstellen des Badwordarrays werden die Inhalte aus der Tabelle *Forummessages* geholt:

```
$db->query("select ForumMessage,ForumMessageID,
        ForumID,ForumThreadID from Forummessages");
```

Danach wird in einer *while* Schleife der Inhalt der *ForumMessages* an eine Skriptvariable übergeben, in Kleinbuchstaben umgewandelt und unerwünschte Zeichen werden aus dem String entfernt. Das Ergebnis ist ein String, der alle Worte, getrennt durch das Zeichen „-” enthält.

```
$string = strtolower($db->f('ForumMessage'));
$string = preg_replace("[[^a-zA-Z]]", ' ', $string);
$string = eregi_replace(" ", '-', $string);
```

Aus dem so generierten Sting wird nun wieder ein Array aufgebaut:

```
$pieces = explode("-", $string);
$pieces_size = sizeof($pieces);
```

Dieses Array wird anschließend durchlaufen, wobei die Elemente, die nicht im Badwordarray vorhanden sind, in einen Array namens *nobadword* übernommen werden. Der Vergleich wird mittels der Funktion *in_array()* von PHP erledigt. Die Funktion *in_array()* nimmt dem Programmierer einiges an Arbeit ab.

```
while ($pieces_size > $i){

    $word = $pieces[$i];
    if(!$word==''){
            if (!in_array ($word, $badwordarr)
            && !in_array ($word, $nobadword)){
                array_push($nobadword, $word); }
        }
        $i++;
    }
```

Danach erhalten wir ein Array mit allen gültigen Suchworten, die im Folgenden in die Suchworttabelle einfügt werden. Dabei ist zu prüfen, ob das Suchwort bereits in der Tabelle vorhanden ist. Ist das Suchwort in der Tabelle *Words* nicht enthalten wird es eingefügt, danach werden WordID, ForumID, ThreadID, MessageID in die Tabelle *Wordsindocuments* eingefügt und somit die Verknüpfung von Wort/WortID zum Forumsinhalt hergestellt. Der Code hierzu ist auf der CD zu finden. Sollte das Suchwort in der Tabelle *Words* vorhanden sein, so werden nur die Einträge in der *Wordsindocuments* Tabelle vorgenommen.

Die Suche über das Forum wird mittels eines HTML Formulars, in das der User seine Suchbegriffe durch Leerzeichen getrennt eingibt, gestartet. Nachdem der User durch Drücken des *SUBMIT* Buttons seine Anfrage abgesendet hat, wird der Suchstring auf der Suchseite zuerst auf seine formale Richtigkeit untersucht. Dabei werden alle unzulässigen Zeichen aus dem Suchstring gelöscht und durch ein Leerzeichen ersetzt. Unzulässige Zeichen sind z.B. Hochkomma, Pipezeichen oder Anführungszeichen, welche die SQL Abfragen stören können. Diese Zeichen können mit der PHP Funktion *preg_replace()* entfernt bzw. umgewandelt werden. Sind diese Zeichen entfernt, so wird der String der Suchbegriffe wieder in ein Array aufgeteilt. Danach wird das Array mit einer *while* Schleife durchlaufen. Bei jedem Durchlauf wird eine WordID zu einem der gefundenen Suchbegriffe geholt.

```
$dbf->query("select WordID,Word from Words where Word =
'$suche_arr[$i]'");
```

Verbesserungswürdig wäre an dieser Stelle vielleicht, dass sich alle WordIDs zu den Suchworten auch mit einem erweiterten SQL Statement holen ließen. Dies würde die Zahl der an die Datenbank gestellten Anfragen etwas reduzieren.
Nachdem wir die WordID haben, wird sie einem Array, das die WortIDs der gefundenen Suchbegriffe enthält, angefügt. Darüber hinaus wird der Suchstring neu mit gültigen Suchbegriffen zusammengebaut, die dann später dem User noch einmal angezeigt werden.

```
if($wid!=''){
                    $suchwortids[$j]= $wid;
                    $Suche = $Suche . $dbf->f('Word')
." ";
                    $j++;
}
```

Im Anschluss daran wird aus dem Array mit den gefundenen WortIDs der SQL String zusammengesetzt, der dann aus der Tabelle *Wordsindocuments* die Dokumente ermittelt, auf die alle unsere Suchbegriffe passen. Der generierte SQL String, der zur Abfrage benutzt wird, sieht z.B. so aus:

```
select straight_join t0.MessageID,t0.ThreadID,t0.ForumID
        from Wordsindocuments t0,Wordsindocuments
        t1,Wordsindocuments t2 where t0.WordID =
        '340'and t0.MessageID=t1.MessageID and t1.
        WordID='645' and t1.MessageID=t2. MessageID
                                and t2.WordID='780'
```

Dieses SQL Statement gibt eine Liste aller nötigen IDs zurück, die gebraucht werden, um die zur Anzeige benötigten Informationen aus den Tabellen *Forum*, *Forumthreads*, *Forummessages* zu holen und anzuzeigen. An dieser Stelle ist festzustellen, dass MySQL im direkten Vergleich mit anderen Datenbanken mehrere Queries benötigt, um an die gleichen Daten zu kommen, für die andere Datenbanken nur ein Query benötigen, da in der MySQL Version 3.x keine Subselects möglich sind. (MySQL plant jedoch die Implementierung von Subselects in einer der neuen 4.1 Versionen). Gleichzeitig ist MySQL aber bei diesen einfachen Queries recht schnell, sodass eine Suche nach den vier Begriffen „reis fleisch butter mehl“ aus den ca. 90.000 Messages im Forum in weniger als 0.8 Sekunden 32 heraussucht, in denen alle Suchbegriffe zu finden sind. Man bedenke, dass MySQL und PHP die Daten aus 170.000 Worten in der Tabelle *Words* sowie aus ca. 2,9 Millionen Datensätzen in der Tabelle *Wordsindocuments* heraussucht.

Der Cluster

Kurz nach dem Start der Version 3.0 wurde deutlich, dass der Webserver das Nadelöhr in dem neuen System war. Der Webserver, der mit der alten chefkoch.de Version einen Load von unter 2 lief und somit recht ordentliche Antwortzeiten generierte, ereichte nun im Normalbetrieb einen Load von 12-20 mit den dazu gehörigen grausigen Wartezeiten beim Seitenaufbau. Die Erhöhung des Rechenbedarfs des Webservers ist wohl durch die

nun zu 100 Prozent dynamischen, auf Templates basierenden Seiten zu erklären. Des Weiteren führte der hohe Load des Webservers dazu, dass Anfragen an die Datenbank langsamer verarbeitet wurden und trieb somit auch den Load des bisher beinah schlafenden Datenbankservers in die Höhe, was wiederum dazu führte, dass der Load des Webservers weiter anstieg. Das schlechte Laufzeitverhalten des Webservers sorgte also dafür, dass das gesamte System instabil arbeitete und zeitweise völlig zum Stillstand kam. Zwar hatten vorherigen Belastungstests mit Apache AB ergeben, das die Leistung des Systems ausreichen sollte, der Realbetrieb zeigte jedoch anderes. Die Bereitstellung einer Clustering Lösung, welche die Webserver leistungsfähiger machen würde, war somit unumgänglich.

Die einfachste – und eine sehr gebräuchliche Lösung in diesem Bereicht – stellt die Einrichtung von RoundRobin DNS dar, welche die Anfragen auf unterschiedliche physikalische Webserver verteilt und somit die Last mehr oder weniger gleich auf die einzelnen beteiligten Systeme aufteilt. RoundRobin DNS hat jedoch auch einige Nachteile, die bei chefkoch.de nicht in Kauf genommen werden sollten. Durch das Verteilen von DNS Anfragen auf unterschiedliche IPs/Rechner können diese bei Ausfall eines der beteiligten Rechner ins Leere laufen. Des Weiteren ist keine Priorisierung der Anfragen auf die einzelnen Systeme möglich, sodass die Hardware der am Clustering beteiligten Systeme möglichst gleichwertige Leistung haben sollte, damit alle Anfragen gleich schnell bedient werden. Unterschiedlich starke Systeme führen im Fall von RoundRobin DNS zu unterschiedlich hoch belasteten Servern, was nicht zu kontrollieren ist und zu ungleichen Antwortzeiten führt. Ein weiterer Nachteil ist, dass Logginginformationen nicht zentral auflaufen, sondern auf jedem der angeschlossenen Systeme separat geschrieben werden und nur über eine Zusatzsoftware, z.B. Spread, wieder zusammengefasst werden können. Die Lösung für chefkoch.de brachte *mod_backhand*, eine Clusteringsoftware die als Modul in den Apache geladen wird.

mod_backhand stellt zwei Methoden des Clustering zur Verfügung. Zum einen Proxying (default), zum anderen Redirecting. Beim Proxying werden alle Anfragen an den Cluster vom „Frontendserver" angenommen, an die Clusterserver durchgereicht, dort bearbeitet und an den Frontendserver zurückgeliefert, der die angeforderten Daten dann an den Client übermittelt. Beim Redirecting werden die Anfragen vom Frontendserver auf einen der Clusterserver umgeleitet, welcher dann selbst die Kommunikation mit dem Client übernimmt und seine Daten direkt an den Client sendet. Da das Redirecting aber zu einer Aufteilung der Logfiles führt und auch noch einige andere Nachteile hat, kommt auf chefkoch.de Proxying zum Einsatz.

Die Installation von *mod_backhand* ist denkbar einfach. Vor der Kompilierung des Apache wird *mod_backhand* mittels eines mitgelieferten Installationskripts in den Apache Sourcetree eingepatcht. Danach kann Apache wie gewohnt mit den gewünschten Parametern kompiliert werden. *mod_backhand* kann als Modul oder fest mit in den Apache einkompiliert werden. Bei chefkoch.de wird es als Modul geladen:

```
#Apache httpd.conf Modules
LoadModule backhand_module  libexec/mod_backhand.so
AddModule mod_backhand.c
```

mod_backhand benutzt zur Kommunikation der Server untereinander wahlweise Broadcast oder Multicast. Chefkoch.de verwendet die Broadcast Methode, da Broadcast in jedem IP-Netz zur Verfügung steht, wohingegen Multicast nur auf multicastfähigen Systemen betrieben werden kann. Die Konfiguration im Apache Configfile ist sehr einfach gehalten. Der Apache Parameter *MulticastStats* in der Form *[<IP ADDR>] <BROADCAST ADDR>:<PORT>* veranlasst *mod_backhand* dazu, seine Existenz und Runtime-Informationen per Broadcast an andere *mod_backhand* Server preiszugeben. *AcceptStats* im Format *<a.b.c.d>[/<mask>]* legt fest, in welchem IP-Kreis auf Runtime-Informationen der am Cluster beteiligten Server gelauscht werden soll.

Werden die beiden im Beispiel verwendeten Apaches nun gestartet, verständigen sie sich untereinander, tauschen die fürs Clustering notwendigen Systeminformationen aus und schließen sich zu einem arbeitsfähigen Cluster zusammen. Bei Ausfall eines Clusterservers wird er automatisch binnen weniger Sekunden aus der internen Serverliste des Clusters genommen, sodass keine Anfragen mehr an ihn geleitet werden. Dies hat den Vorteil das man alle am Cluster beteiligten Server außer den „Frontendserver" nach belieben ein- oder abschalten kann, was z.B. Wartungsarbeiten an den Servern deutlich erleichtert, da man sich keine Gedanken um fehlgeleitete oder nichtbeantwortete Clientanfragen machen muss.

Im Gegensatz zu anderen Clustering Methoden clustert *mod_backhand* nicht den ganzen Server, sondern der User kann festlegen, welche Verzeichnisse des Servers zum Clustering herangezogen werden sollen. Dadurch ist es z.B. möglich, für www.chefkoch.de den Cluster zu verwenden, während alle Anfragen für die Domain bilder.chefkoch.de, die auf dem gleichen Server gehostet wird nicht ins Clustering mit einbezogen werden. Die Einstellung welche Inhalte geclustert werden, erfolgt über die Directory Einstellungen von Apache. Hierbei gibt es die Möglichkeit, die Last nach Last des Servers (*byLoad*), Alter (*byAge*), oder per Random (*byRandom*) zu verteilen (es gibt noch einige andere, die hier unerwähnt bleiben). Das Clustering *byAge* und *byLoad* sind nach Aussage des

Listing 1

Beispiele:

Server1:

```
<IfModule mod_backhand.c>
MulticastStats 213.70.6.16 213.70.6.255:4445
AcceptStats 213.70.6.0/24
UnixSocketDir /usr/local/apache/backhand
BackhandConnectionPools Off
</IfModule>
```

Server2:

```
...
MulticastStats 213.70.6.14 213.70.6.255:4445
AcceptStats 213.70.6.0/24
...
</IfModule>
```

Programmierers von *mod_backhand* die am besten geeigneten für Anwendungen wie chefkoch.de und kommen daher auch hier zur Anwendung. Der directory Eintrag auf chefkoch.de ist so gestaltet, dass alle Anfragen an www.chefkoch.de auf die beteiligten Server umgeleitet werden. Hier ist der Apache Eintrag der aktuellen Config von chefkoch.de:

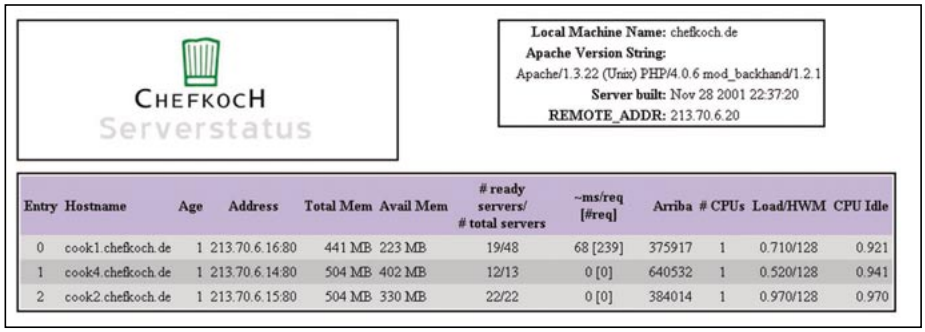
```
<Directory /www/httpd/chefkoch.de/html/>
#...
Backhand byAge 3
Backhand byLoad
#...
</Directory>
```

In manchen Fällen ist es erwünscht, das Anfragen auf Unterverzeichnisse des Webservers aus dem Cluster herausgenommen und nur vom Führungsserver beantwortet werden. Dies lässt sich recht komfortabel über den Location Eintrag von Apache lösen. In diesem Beispiel wird *www.chefkoch.de/testdir/* nicht geclustert, sondern nur vom Führungsserver beantwortet:

```
<Location „/testdir/“>
  Backhand off
</Location>
```

Damit der Admin eines Clusters nicht im Dunkeln über den Zustand seiner Systeme gelassen wird, bringt *mod_backhand* eine Statusübersichtsseite mit, auf welcher einige nützliche Systeminformationen angezeigt werden. Diese Übersichtsseite kann mit wenigen C++ Kenntnissen vom Layout her angepasst werden.

Durch die Aufteilung der Anfragen unter Beachtung der Serverbelastung (Clustering byLoad) der angeschlossenen Systeme ist es möglich, jede beliebige Hardware als Clusterserver zu verwenden. Hier kann von dem alten P 200 bis hin zu modernen Doppelprozessor-Maschinen jeder Rechner eingesetzt werden. Ist der Cluster dann richtig konfiguriert, werden alle Server mit einer ähnlich schnellen Antwortzeit arbeiten, weil *mod_backhand* die Anfragen auf alle beim Clustering beteiligten Maschinen verteilt und so den Load auf allen Rechnern nahezu gleich hält. Bei chefkoch.de kommen derzeit ein PIII 550 ein PIII 650 und ein PIII 933 mit jeweils 512 MB RAM zum Einsatz. Alle Rechner laufen im Normalbetrieb mit einem Load von ca. 1, was eine derzeit recht ordentliche Performance garantiert und den Usern ein performantes Surfen ermöglicht.



Entry	Hostname	Age	Address	Total Mem	Avail Mem	# ready servers/ # total servers	~ms/req [#req]	Arriba	# CPUs	Load/HWM	CPU Idle
0	cook1.chefkoch.de	1	213.70.6.16:80	441 MB	223 MB	19/48	68 [239]	375917	1	0.710/128	0.921
1	cook4.chefkoch.de	1	213.70.6.14:80	504 MB	402 MB	12/13	0 [0]	640532	1	0.520/128	0.941
2	cook2.chefkoch.de	1	213.70.6.15:80	504 MB	330 MB	22/22	0 [0]	384014	1	0.970/128	0.970

Abb. 1: *mod_backhand* Statusübersicht, angepasst an chefkoch.de.

CVS als Entwicklungs- und Updatesystem.

In einem Servercluster müssen die hinterlegten Informationen immer möglichst gleich auf allen angeschlossenen Systemen sein. Um zu gewährleisten, dass alle Systeme den gleichen Filebestand haben, benutzt chefkoch.de CVS, ein System für die Verwaltung von Programmentwicklungen. Mittels CVS ist es zum einen möglich, einen ständigen Überblick über die laufenden und vergangenen Softwareversionen zu haben, zum anderen können sich hier beliebig viele Entwickler oder auch die Server eine aktuelle Version der chefkoch.de Daten holen. Die Entwicklungsarbeit an PHP, HTML sowie den zum Layout gehörenden Bildern wird auf einem Development Server geleistet. Dieser Server ist nicht an den Cluster angeschlossen, wodurch Entwicklungen an den PHP-Skripten unabhängig vom laufenden System erledigt und in Ruhe getestet werden können. Ist eine Entwicklung abgeschlossen, so wird sie in den CVS Server eingespielt, der alle Entwicklungsstände verwaltet. Die Clusterserver aktualisieren ihre lokale Version von chefkoch.de in regelmäßigen Abständen per Cronjob am CVS Server und enthalten eine aktuelle lauffähige Kopie der chefkoch.de Skripte, Templates und Bilder.

Als Resümee der nun mehrjährigen Entwicklungszeit bis zur Version 3.0 von chefkoch.de lässt sich sagen, dass die Site ohne die Verwendung der Freien Software wie Apache, PHP und MySQL nicht mit so reichhaltigen Funktionen ausgestattet wäre, wie sie es heute ist. Die Verwendung von kommerzieller Software, die den gleichen Funktionsumfang bietet wie der in chefkoch.de implementierte, hätte wahrscheinlich mehrere 10.000 € gekostet. Fraglich ist, ob das System damit genau so stabil betrieben und flexibel entwickelt worden wäre. Auch ist die Unabhängigkeit von der Lizenz- und Updatepolitik der Softwarehersteller ein nicht zu unterschätzender Vorteil, insbesondere im Hinblick auf Sicherheitslücken in der Software, die bei freier Software in der Regel umgehend nach dem Bekanntwerden mit einem neuen Release beseitigt werden.

An dieser Stelle möchte ich allen Entwicklern danken, die zur Entwicklung freier Software beitragen und im Allgemeinen einen sehr guten Support bieten, der bei kommerziellen Produkten in dieser Form nur selten zu finden ist. Mein besonderer Dank geht an die PHP Community im IRC Channel php.de, welche immer ein offenes Ohr für Fragen hat.

Alexander Meis ist unter am@chefkoch. zu erreichen.

Links

Apache: www.apache.org/

mod_backhand: www.backhand.org

Spread: www.backhand.org/mod_log_spread/

PHP: www.php.net

MySQL: www.mysql.org

CVS: www.cvshome.org/

PHP Magazin Inserentenverzeichnis & Vorschau

Die nächste Ausgabe des PHP Magazins erscheint am 29.05. 2002!

Vorschau

Content Management mit PHP

Ausführliche Übersicht über mehr als 30 PHP-basierende Content-Management-Systeme.

PDF-Erzeugung mit PC4P und PDFLIB

Wie Sie einfach dynamische PDFs generieren.

Die Website phpug.de

als Dreh- und Angelpunkt der deutschsprachigen PHP-Usergruppen. Erfahren Sie alles über die Gründung von phpug.de und die Usergruppen.

Mit XML-RPC erfolgreich arbeiten.

Warum Sie XML-RPC brauchen und wozu es gut ist.

Außerdem Tutorials für PHP-Einsteiger, Rezensionen, Interviews, Development-Artikel für Profi-Entwickler, Tipps & Tricks rund um Datenbanken sowie Vorstellung professioneller PHP-Lösungen.

PHP Magazin Impressum

Herausgeber:
Software & Support Verlag GmbH

Anschrift der Redaktion:
PHP Magazin
Software & Support Verlag GmbH
Kennedyallee 87
60596 Frankfurt am Main
Tel.: (069) 63 00 89 - 0
Fax: (069) 63 00 89 - 89
eMail: redaktion@phpmag.de
WWW: www.phpmag.de

Chefredaktion: Björn Schotte
eMail: bjoern@phpmag.de
Redaktion: Dr. Nadja Rosmann, Dr. Stefan Wunderlich
Redaktionsassistenz: Mark Hohe-Dorst, Silke Martin, Meike Wulf
Herstellungsleitung: Jens Mainz
Layout, Titel:
Dominique Bergmann, Tobias Friedberg, Melanie Hahn, Jens Mainz, Maria Rudi

Autoren dieser Ausgabe:
Alexander Aulbach, Till Gerken, Ewald Geschwinde, Andre Gildemeister, Dr. Volker M. Göbbels, Markus Hasenbein, Hartmut Holzgraefe, Frederik M. Kraus, Alexander Meis, Sebastian Mordziol, Sebastian Nohn, Stefan Priebsch, Dr. Nadja Rosmann, Gerd Schaufelberger, Stephan Schmidt, Hans-Jürgen Schönig, Thorsten Suckow-Homberg, Tobias Wassermann

Pressevertrieb:
IPV Inland Presse Vertrieb GmbH
Postfach 10 32 46
20022 Hamburg

Anzeigen
MARKETING PROJEKT 2000 GMBH
Hochstr. 3
86453 Dasing
Tel. 08205 / 96 233
Fax 08205 / 96 23 - 45
eMail: info@mp2000.de

Aboservice
Software & Support Verlag GmbH
Tel. 069 / 6300 89 0
Fax 069 / 6300 89 89
eMail: abo@phpmag.de

Abonnementpreise

Inland	4 Ausgaben	34,50 €
Ausland	4 Ausgaben	44,50 €
Student Inland	4 Ausgaben	27,60 €
Student Ausland	4 Ausgaben	37,60 €

Einzelverkaufspreis

Deutschland	9,80 €
Österreich	10,20 €
Luxemburg	11,25 €
Schweiz	19,20 sFr

Erscheinungsweise: vierteljährlich

Druck: PVA, Landau

© 2002 für alle Beiträge.
Alle Rechte, auch für Übersetzungen, sind vorbehalten. Reproduktionen jeglicher Art (Fotokopie, Nachdruck, Mikrofilm oder Erfassung auf elektronischen Datenträgern) nur mit schriftlicher Genehmigung des Verlags. Jegliche Software auf der CD-Rom unterliegt den Bestimmungen des Herstellers oder der zuständigen Organisation. Eine Haftung für die Richtigkeit der Veröffentlichungen kann trotz Prüfung durch die Redaktion vom Herausgeber nicht übernommen werden. Honorierte Artikel gehen in das Verfügungsrecht des Verlags über. Mit der Übergabe der Manuskripte, Abbildungen eventueller Quellcodes an den Verlag erteilt der Verfasser dem Herausgeber das Exklusivitätsrecht zur Veröffentlichung. Für unverlangt eingesandte Manuskripte, Abbildungen oder Quellcodes keine Gewähr.

Alle Markennamen sind in der Regel eingetragene Warenzeichen der entsprechenden Hersteller oder Organisationen.

Anzeige

Anzeige