CGI-Programmierung mit Perl

CGI-Programmierung mit Perl

2. Auflage

Scott Guelich, Shishir Gundavaram & Gunther Birznieks

Deutsche Übersetzung von
Jørgen Lang

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
Tel.: 0221/9731600
Fax: 0221/9731608
E-Mail: kommentar@oreilly.de

1. Auflage 1996
2. Auflage 2001

Die Originalausgabe erschien 2000 unter dem Titel
CGI Programming with Perl, 2nd Edition im Verlag O'Reilly & Associates, Inc.

Die Deutsche Bibliothek - CIP - Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist
bei der Deutschen Bibliothek erhältlich.

Übersetzung und deutsche Bearbeitung: Jørgen Lang
Lektorat: Michael Gerth, Köln
Korrektorat: Friederike Daenecke, Oliver Mosler, Martin Vorländer, Gerald Richter
Satz: Tung Huynh, reemers publishing services GmbH, Krefeld
Umschlaggestaltung: Edie Freedman & Risa Graziano, Boston
Produktion: Geesche Kieckbusch, Köln
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Kempten; www.Koeselbuch.de

ISBN 3-89721-167-X

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt

Inhalt

Vorwort

Als 1996 die 1. Auflage von *CGI-Programmierung im World Wide Web* herauskam, sah das Web noch ganz anders aus. Es gab gerade einmal 100.000 Webserver. Der aktuelle Netscape-Browser war der Navigator 2.0 (der erste Browser mit JavaScript™-Fähigkeiten). Java™ war nicht einmal ein Jahr alt und wurde hauptsächlich zur Programmierung von Applets benutzt.

1996 war CGI die einzige stabile und gut verständliche Methode, dynamische Inhalte für das Web zu erzeugen. Dennoch wurde es nur auf wenigen Sites mit der ganzen Bandbreite seiner Möglichkeiten eingesetzt. 1996 schrieb Shishir Gundavaram:

> Heutzutage erwarten die Benutzer individuelle Antworten auf bestimmte Fragen. Die Zeiten, in denen die Leute mit einer einzigen von der EDV-Abteilung erstellten Liste zufrieden waren, sind vorbei. Statt dessen möchte jeder der Vertriebsmitarbeiter, Manager und Ingenieur spezielle Fragen stellen und aktuelle Antworten erhalten. Wenn das ein einziger Computer kann, warum dann nicht auch das Web?
>
> Genau das verspricht CGI. Sie können die Verkaufsstatistiken ganz nach Wunsch der Vertriebsmitarbeiter in wunderschönen (Torten-)Diagrammen ausgeben. Sie können Kunden die Eingabe von Schlüsselwörtern erlauben mit denen sie nach bestimmten Produktinformationen suchen können.

Was 1996 noch hochgesteckte Ziele waren ist, heute Tagesgeschäft. Was CGI versprochen hat, ist sicherlich eingelöst worden.

In diesem Buch geht es um mehr als nur um das Schreiben von CGI-Skripten. Es geht darum, für das Web zu programmieren. Obwohl wir uns hauptsächlich mit CGI-Programmen in Perl beschäftigen (daher auch die Änderung des Titels dieser Auflage), lassen sich viele der Konzepte, die wir hier beschreiben, auf die Entwicklung anderer serverseitigen Web-Anwendungen übertragen. Auch, wenn Sie nur mit entfernt verwandten Technologien zu tun haben, lohnt sich die Auseinandersetzung mit CGI.

Was Sie in diesem Buch finden

In den letzten Jahren hat sich CGI stark verändert. Daher ist es nur angemessen, daß die Neuauflage diese Veränderungen widerspiegelt. Deshalb wurde dieses Buch auch in großen Teilen neu geschrieben. Neue Themenbereiche sind das Modul CGI.pm, die Verwendung von HTML-Templates, Sicherheitsaspekte, JavaScript, XML, Suchmaschinen, Vorschläge zum Programmierstil und kompatible, hochperformante Alternativen zu CGI. In der ersten Auflage bereits behandelte Themen, wie Zustandsverwaltung, E-Mail, dynamisch erzeugte Grafiken und relationale Datenbanken wurden ergänzt und auf den neuesten Stand gebracht. Schließlich haben wir der Vorstellung von CGI eine Einführung in die dem Web zugrundeliegende »Sprache«, das Hypertext Transfer Protocol (HTTP), vorangestellt. Das Verständnis von HTTP ist eine solide Grundlage, um CGI besser zu verstehen.

Trotz dieser Veränderungen hat sich das Ziel dieses Buches nicht geändert: Ihnen alles beizubringen, was Sie wissen müssen, um ein guter CGI-Entwickler zu werden. Dieses Buch ist keine Schritt-für-Schritt-Anleitung. Es ist kein Buch, das lediglich aus einer Handvoll Skripten besteht und erklärt, wie diese funktionieren. Es gibt viele solche CGI-Bücher. Diese Bücher können sicherlich hilfreich sein, besonders, wenn sie genau für die Aufgaben Beispiele enthalten, die Sie gerade zu lösen haben. Allerdings wird oft nur gezeigt, *wie* etwas funktioniert, ohne zu erklären, *warum*. Das Ziel dieses Buches ist es, die Grundlagen zu schaffen, die Sie brauchen, um CGI-Skripten für die verschiedensten Aufgaben zu entwickeln. Machen Sie sich aber keine Sorgen – wir werden uns trotzdem eine Menge Beispiele ansehen. Allerdings dienen unsere Beispiele eher dazu, die Diskussionen zu illustrieren, als andersherum.

Wir geben zu, daß dieses Buch etwas Unix-lastig ist. Perl und CGI sind ursprünglich auf Unix-Systemen (und für diese) entwickelt worden, diese sind auch heute noch die populärste Plattform für diese Aufgaben. Dennoch unterstützen eine Vielzahl anderer Systeme Perl und CGI, darunter Microsofts beliebte 32-bit Windows-Systeme: Windows 95, Windows 98, Windows NT und Windows 2000 (im folgenden *Win32-Systeme* genannt). Wir werden uns in diesem Buch auf Unix-basierte Systeme konzentrieren, werden aber auf die Dinge hinweisen, die Sie wissen müssen, um auch für andere Plattformen entwickeln zu können.

In unseren Beispielen benutzen wir den Apache-Webserver. Dafür gibt es mehrere Gründe: Er ist der momentan beliebteste Webserver, es gibt ihn für fast alle Betriebssysteme, er ist kostenlos, sein Quellcode ist frei verfügbar und er unterstützt Module (wie etwa *mod_perl* und *mod_fastcgi*), mit denen die Leistungsfähigkeit von Perl für Webentwicklungen erhöht werden kann.

Die Beispielcodes in diesem Buch und weitere Informationen finden Sie unter:

http://www.oreilly.de/catalog/cgi2ger/

Was Sie wissen sollten, bevor Sie dieses Buch lesen

Sie sollten sich bereits mit Perl auskennen. Während in der 1. Auflage von *CGI-Programmierung im World Wide Web* auch andere Programmiersprachen behandelt wurden, beschäftigt sich diese Ausgabe von *CGI-Programmierung mit Perl* ausschließlich mit Perl (wie der neue Titel bereits andeutet). Auch wenn CGI-Programmierung in vielen Sprachen möglich ist, hat sich Perl als die Sprache der Wahl herauskristallisiert.

Falls Sie noch nicht mit Perl vertraut sind, finden Sie eine ausgezeichnete Anleitung in *Einführung in Perl, 2. Auflage* von Randal Schwartz und Tom Christiansen (O'Reilly-Verlag). Wenn Sie die Grundlagen einmal beherrschen, empfehlen wir die Anschaffung des Buches *Programmieren mit Perl* (auch bekannt als das *Kamelbuch*) von Larry Wall, Tom Christiansen und Jon Orwant (O'Reilly-Verlag). Dieses Buch ist das Referenzwerk für Perl-Entwickler in aller Welt. Eine Liste weiterer Bücher und anderer Quellen zum Thema Perl finden Sie im Anhang A.

Wir werden viele Module aus dem CPAN, dem *Comprehensive Perl Archive Network,* behandeln. Eine Anleitung zum Herunterladen und Installieren von Modulen aus dem CPAN finden Sie im Anhang B.

Außerdem sollten Sie *perldoc* benutzen können. *perldoc* ist das Standardwerkzeug zum Betrachten und Durchsuchen der umfangreichen Dokumentation, die Perl standardmäßig beiliegt. Außerdem ist *perldoc* essentiell, um mehr über die CPAN-Module zu erfahren. Hinweise zur Benutzung von *perldoc* finden Sie ebenfalls im Anhang B.

Wie dieses Buch aufgebaut ist

Kapitel 1 enthält eine allgemeine Einführung in CGI, einen Überblick über die historische Entwicklung von CGI, eine Anleitung zur Webserverkonfiguration und ein Beispiel-CGI-Skript.

Die Kapitel 2 bis 4 behandeln die Grundlagen von CGI. Wir beginnen mit einem Überblick über HTTP und schauen uns dann an, wie CGI darauf aufbaut. Schließlich beschäftigen wir uns mit HTML-Formularen, der üblichen Methode, Informationen an CGI-Skripte zu übergeben.

Kapitel 5 und 6 zeigen populäre Module, die beim Schreiben von CGI-Skripten hilfreich sind. Außerdem vergleichen wir verschiedene Strategien, dynamische HTML-Ausgaben zu erzeugen.

In Kapitel 7 geht es darum, wie Sie JavaScript einsetzen können, um Ihre CGI-Skripten zu verbessern.

In den Kapiteln 8 bis 13 stellen wir Lösungen von Problem vor, die beim Entwickeln von CGI-Skripten häufig auftreten. Hier geht es um Themen wie Sicherheit im Netz, die Persistenz von Daten und die Zustandsverwaltung über mehrere Seitenaufrufe. Außerdem zeigen wir Ihnen, wie Sie mit Perl E-Mails verschicken, Ihren Benutzern das Durchsuchen Ihrer Website ermöglichen, und erklären Ihnen das Erzeugen dynamischer Grafiken.

Kapitel 14 zeigt, wie man mit Hilfe von Middleware und XML CGI-Schnittstellen zu anderen Informationsservern entwickeln kann.

Die Kapitel 15 bis 17 zeigen, wie man bessere CGI-Skripten schreibt. Es wird beschrieben, wie das Debuggen von CGI-Skripten funktioniert. Es werden Richtlinien zum Schreiben guten Codes vorgestellt und gezeigt, wie man die Performanz von CGI-Skripten steigern kann.

Dieses Buch beinhaltet außerdem zwei Anhänge, die weiterführende Quellen zum Thema CGI sowie eine Anleitung zur Benutzung des CPAN enthalten.

In diesem Buch verwendete Konventionen

`Nichtproportionalschrift`
: wird für HTTP-Header, Statuscodes, MIME-Inhaltstypen, Direktiven in Konfigurationsdateien, Arrays, Operatoren, Variablennamen (außer solchen in Beispielen) und zeichenorientierte Computerausgaben verwendet.

Kursive Schrift
: wird für Dateinamen, Pfadangaben, Namen von Newsgruppen, Internetadressen (URLs), E-Mail-Adressen, neueingeführte Begriffe, Kommandos, Optionen/Switches, Namen von Programmen, Namen von Subroutinen, Funktionen, Methoden und Host-Namen verwendet.

GROSSBUCHSTABEN
: werden für Umgebungsvariablen, HTML-Attribute und HTML-Tags (innerhalb spitzer Klammern <>) verwendet.

Danksagungen

Nachdem ich selbst die Erfahrung gemacht habe, an einem Buch zu arbeiten, werde ich Listen mit Danksagungen nicht mehr so lesen können wie bisher. Ein Buch erfordert eine Menge Arbeit von vielen verschiedenen Leuten. Freunde und Familie tragen weit mehr zum Gelingen bei, als ich es mir je vorgestellt habe.

Ich möchte mich bei meinen Freunden und meiner Familie bedanken. Sie waren nicht nur sehr verständnisvoll, als dieses Buch einen Großteil meiner Zeit und Energie von ihnen abgezogen hat, sondern haben auch niemals aufgehört zu fragen, wie es voran-

geht, und haben geduldig zugehört, wann immer ich mich über meinen Mangel an Freizeit beklagt habe. Ich danke auch den freundlichen Menschen bei Printers Inc. in Mountain View, wo der größte Teil dieses Buches zwischen diversen Tassen Kaffee und Tee geschrieben wurde.

Ich danke Brad Ashmore bei Hewlett Packard vielmals, der es mir ermöglicht hat, in Teilzeit zu arbeiten, während ich mit meiner Arbeit, dem Buch und meiner Gesundheit jonglierte. Mein Dank geht auch an Baskar Srinivasan, Natasha Fattedad und Anh Hoang, die sich um das Liegengebliebene gekümmert haben. Ich danke außerdem allen Mitarbeitern bei HP für Ihr Verständnis, als ich es nicht mehr geschafft habe, alle Bälle in der Luft zu halten.

Danken möchte ich allen bei O'Reilly. Ein großes Dankeschön geht an Linda Mui, die dieses Buch bis zur Fertigstellung »behütet« hat. Sie war immer da, um Fragen zu beantworten und hat mir immer genau die richtige Mischung aus Ermunterung und vorsichtiger Kritik zukommen lassen. Dank auch an Rob Romano für die Illustrationen und an Christien Shangraw für die Koordination.

Shishir Gundavaram verdient meinen Dank sowohl für neues Material als auch dafür, daß er die erste Ausgabe dieses Buches geschrieben hat, die so viele von uns gelesen und benutzt haben.

Ein großes Dankeschön geht an die Korrektoren und an alle, die mir mit Ihren Rückmeldungen geholfen haben. Gunter Birznieks hat nicht nur Kapitel zu diesem Buch beigetragen, sondern es auch einer gründlichen Prüfung unterzogen. Nat Torkington hat es ebenfalls sehr detailliert geprüft. Des weiteren haben mir Linda Mui, Andy Oram, Dan Beimborn, Sam Tregar, Paula Ferguson und Jon Orwant mit ihren Rückmeldungen geholfen.

Mein Dank geht nicht zuletzt an die Gemeinschaft der Open Source-Entwickler, die viele Stunden gearbeitet haben, um die Programme und Module zu schreiben, die in diesem Buch besprochen werden. Ohne ihre Arbeit wäre das Web nicht das, was es heute ist.

– Scott Guelich
Juli 2000

Am Enstehungsprozeß eines Buches sind viele Leute beteiligt. Besonders die Zusammenarbeit mit den beiden sehr talentierten Autoren Scott Guelich und Shishir Gundavaram, die ich vorher nur »virtuell« kannte, war eine großartige Erfahrung. Zusätzlich möchte ich Andy Oram und Linda Mui danken. Von beiden habe ich während der Arbeit an diesem Buch viel gelernt.

Ich danke Lincoln Stein vielmals, der mir vorgeschlagen hat, mit Andy Kontakt aufzunehmen. Außerdem herzlichen Dank an all die anderen bei O'Reilly, die mitgeholfen haben, dieses Buch wirklich werden zu lassen. Ein Buch ist immer ein Projekt, an der viele Personen beteiligt sind.

Ich möchte mich außerdem bei der Gemeinschaft der Open-Source-Entwickler bedanken, dafür, daß sie es notwendig gemacht haben, dieses Buch neu zu schreiben. Wenn ich mir anschaue, wie viele Verbesserungen an Perl und an den anderen Webtechnologien vorgenommen wurden, wie viele Module diese Menschen geschrieben haben und wie sich die zugrundeliegenden globalen Infrastruktur des Web verändert hat, bin ich immer wieder erstaunt darüber, was alles erreicht worden ist.

Schließlich möchte ich mich noch bei Organisationen, wie der Electronic Frontier Foundation (*http://www.eff.org/*) bedanken, daß sie sich in diesen Zeiten zunehmender gesetzlicher Einschränkungen bemühen, Web und Internet so frei wie möglich zu erhalten. Ideale wie die ihren, haben einen Raum entstehen lassen, durch den Ideen und Informationen frei in jede Gegend der Erde gelangen können.

– Gunther Birznieks
Juli 2000

Danksagungen der 1. Auflage

Ich möchte mich bei Dyung Le bedanken, nicht nur, weil er die Idee zu diesem Buch hatte, sondern auch, weil er mir die Möglichkeit gegeben hat, unmittelbar nach der High School Software zu entwickeln. Darüber hinaus danke ich Rita Horsey, die mir ebenfalls viel beigebracht hat und mir in der ersten Schreibphase eine Internetverbindung zur Verfügung gestellt hat.

Natürlich möchte ich auch meiner Familie danken. Sie haben sich, während ich dieses Buch geschrieben habe, nicht nur mit meinen bizarren Arbeitszeiten abgefunden, sondern waren auch immer da, wenn ich sie brauchte. Ohne ihre Unterstützung wäre dieses Buch niemals möglich gewesen.

Ein Dank an alle Korrektoren und an alle anderen Menschen, die mich mit Anregungen versorgt haben: Jeffrey Friedl (der König der regulären Ausdrücke), Andreas König (der Vater von MakeMaker), Marc Hedlund (der Begründer der CGI-FAQ), Tom Christiansen (der UNIX-Wizard), Jon Backstrom, Joseph Radin, Paul DuBois und an alle bei O'Reilly, Norman Walsh, Paula Ferguson, Ellie Cutler, Tanya Herlick, Frank Willison, Andy Oram, Linda Mui und Tim O'Reilly.

Zuletzt noch ein Dank an all meine Freunde hier und an meine Familie und Verwandten in Indien, ganz besonders an meine Großeltern.

Ich hoffe, Sie finden dieses Buch nützlich.

– Shishir Gundavaram
März 1996

1

Einführung

Wie das übrige Internet hat sich auch die Anwendung des *Common Gateway Interface* (*CGI)* in sehr kurzer Zeit sehr weit entwickelt. Noch vor ein paar Jahren waren CGI-Skripten eine Neuheit, allerdings waren sie noch nicht besonders praktikabel. Hauptsächlich von Hobby-Programmierern geschrieben, handelte es sich meist um Zugriffszähler und Gästebücher. Heutzutage werden CGI-Skripten von professionellen Web-Entwicklern geschrieben und tragen viel zu der Logik bei, die die riesige Struktur des Internet im Innern beherrscht.

Geschichtliches

Trotz der Aufmerksamkeit, die das Internet heutzutage bekommt, ist es doch nicht neu. Tatsächlich ist der Vorgänger des heutigen Internet schon dreißig Jahre alt. Das Internet begann seine Existenz als ARPAnet. Dieses wurde vom US-Verteidigungsministerium zur Erforschung von Rechnernetzwerken finanziert. Nach und nach wuchs das Internet während der ersten 25 Jahre, um dann zu einer plötzlichen Blüte zu gelangen.

Im Internet gab es schon immer eine Reihe von verschiedenen Protokollen zum Informationsaustausch. Als allerdings Webbrowser wie NCSA Mosaic und später der Netscape Navigator aufkamen, sorgten sie für ein explosives Wachstum. Allein während der letzten sechs Jahre ist die Zahl der im Web betriebenen Rechner von unter tausend auf über zehn Millionen angewachsen. Inzwischen verstehen viele Leute unter dem Begriff Internet nur noch das Web. Andere Protokolle für z.B. E-Mail, FTP, (IRC-)Chat oder News werden sicherlich auch weiterhin beliebt bleiben, sind aber im Vergleich zum Web nur noch zweitrangig. Dies liegt daran, daß inzwischen viele Leute über Websites auf diese Dienste zugreifen.

Das Web ist natürlich nicht das erste Medium, über das Informationen veröffentlicht und ausgetauscht werden konnten. Dennoch gibt es im Web einen entscheidenden Unterschied, der sein explosives Wachstum ausgelöst hat. Wir würden Ihnen jetzt gern erzählen, daß CGI der einzige Grund ist, warum das Web über Protokolle wie FTP und Gopher hinausgewachsen ist. Das würde aber nicht der Wahrheit entsprechen. Der

wahre Grund, warum das Web so beliebt wurde, liegt vermutlich darin, daß Bilder dabei waren. Das Web wurde entwickelt, um die verschiedensten Medientypen darzustellen. Browser konnten schon sehr früh eingebettete Bilder darstellen. HTML ermöglichte schon recht bald eine rudimentäre Layoutkontrolle, so daß Informationen leichter zu präsentieren und zu lesen waren. Diese Kontrolle wuchs immer weiter, nachdem Netscape seinen Browsern mit jeder Version mehr Fähigkeiten mitgab, um neue HTML-Erweiterungen darzustellen.

Dadurch wuchs das Web nach und nach zu einer Ansammlung privater Homepages und Websites, die eine Fülle verschiedenster Informationen enthielten. Dennoch wußte niemand, insbesondere auch bei Firmen, so recht, was damit nun anzufangen war. 1995 hörte man in vielen Unternehmen noch die Auffassung: »Na klar, das Internet ist toll, aber wie viele Leute haben denn tatsächlich schon online Geld verdient?« Wie schnell sich die Dinge doch ändern.

Wie CGI heute benutzt wird

Mittlerweile ist der E-Commerce den Kinderschuhen entwachsen, und Firmen, die ihr Geld ausschließlich online verdienen (sog. dot-coms), tauchen überall auf. Diese Entwicklung beruht auf einigen grundlegenden Technologien, und CGI ist sicherlich eine der wichtigsten. Ressourcen, die sich zwischen zwei Aufrufen nicht verändern, wie zum Beispiel eine HTML-Datei oder eine Grafik werden *statische* Ressourcen genannt. Durch CGI kann das Web etwas *tun,* anstatt nur eine Sammlung statischer Ressourcen zu sein. Eine *dynamische* Ressource ist eine, die sich bei jedem Aufruf abhängig von einer beliebigen Zahl von Bedingungen verändern kann, wie etwa je nach Benutzereingaben oder der Identität eines Benutzers. Hierbei kann auch die Datenquelle variabel sein, beispielsweise eine Datenbank. Dadurch, daß CGI dynamische Inhalte unterstützt, können Benutzer aus aller Welt mit einer standardisierten Clientsoftware, dem Webbrowser, von völlig verschiedenen Rechnerplattformen aus auf eine Online-Anwendung zugreifen.

Es ist schwierig aufzuzählen, was man mit CGI alles machen kann, weil CGI so viel kann. Wenn Sie auf einer Website eine Suche durchführen, ist es höchstwahrscheinlich ein CGI-Programm, das Ihre Informationen verarbeitet. Wenn Sie im Web ein Formular ausfüllen, bearbeitet vermutlich ein CGI-Programm Ihre Eingaben. Wenn Sie online etwas kaufen, ist es vermutlich ein CGI-Programm, das Ihre Kreditkartendaten überprüft und über die Transaktion Buch führt. Wenn Sie sich online ein Diagramm anschauen, ist es sehr gut möglich, daß ein CGI-Programm dieses Diagramm erzeugt hat. Natürlich haben sich in den vergangenen Jahren noch andere Techniken entwikkelt, um diese Aufgaben zu erledigen. Wir werden uns gleich ein paar anschauen. Dennoch bleibt CGI die beliebteste Methode, um diese und andere Aufgaben zu erledigen.

Einführung in CGI

CGI kann soviel, weil es so schlicht ist. CGI ist eine sehr einfach gehaltene Schnittstelle. Es stellt gerade so viel Funktionalität zur Verfügung, wie der Webserver benötigt, um externen Prozessen zu gestatten, Webseiten zu erstellen. Erhält ein Webserver eine Anfrage nach einer statischen Webseite, sucht er das entsprechende HTML-Dokument in seinem Dateisystem und sendet es an den Browser zurück. Übergibt man dem Webserver eine Anfrage nach einem CGI-Skript, führt er das Skript als eigenen Prozeß (z.B. ein externes Programm) aus. Der Server übergibt diesem Prozeß einige Parameter und sammelt dessen Ausgabe wieder ein. Diese Daten werden dann an den Client zurückgegeben, als wären die Daten aus einer statischen Datei gekommen (siehe Abbildung 1-1).

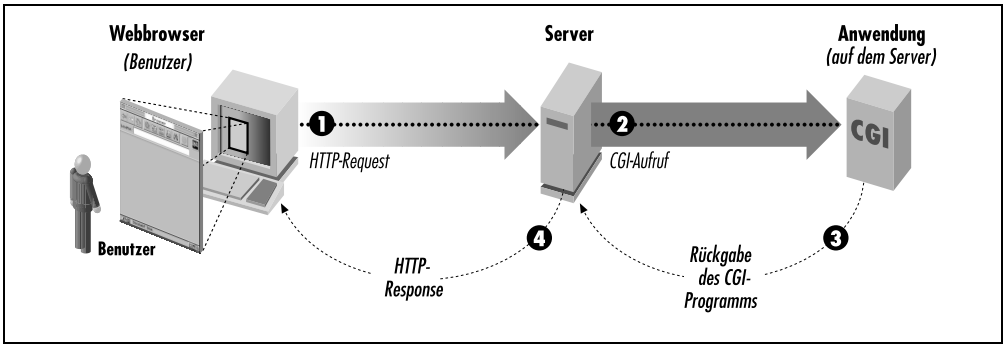

Abbildung 1-1: Ausführung eines CGI-Programms

Wie aber funktioniert diese Schnittstelle genau? Wir werden den Rest dieses Buches damit zubringen, diese Frage detaillierter zu beantworten. Lassen Sie uns aber zuerst einmal einen Blick auf die Grundlagen werfen.

CGI-Skripten und andere dynamische Ressourcen werden vom Browser auf die gleiche Art und Weise angefordert, wie alle anderen Ressourcen im Web. Hierfür schickt der Browser eine gemäß dem *Hypertext Transport Protocol* (*HTTP)* formatierte Nachricht an den Webserver. Wir werden HTTP in Kapitel 2 genauer kennenlernen. Ein *HTTP-Request* enthält einen *Universal Resource Locator* (*URL)*. Anhand des URL kann der Webserver erkennen, welche Ressource zurückgegeben werden soll. In der Regel teilen sich CGI-Skripten ein gemeinsames Verzeichnis, wie zum Beispiel */cgi* oder eine Dateiendung, wie etwa *.cgi*. Erkennt nun der Webserver, daß es sich um einen Request für ein CGI-Skript handelt, so wird dieses Skript ausgeführt.

Nehmen wir einmal an, Sie wollten den URL *http://www.mikesmechanics.com/cgi/willkommen.cgi* besuchen. Beispiel 1-1 zeigt einen Beispiel-HTTP-Request, wie ihn der Webbrowser schicken könnte.

Beispiel 1-1: Beispiel für einen HTTP-Request

```
GET /cgi/willkommen.cgi HTTP/1.1
Host: www.mikesmechanics.com
```

Der GET-Request weist den Server an, die Ressource */cgi/willkommen.cgi* zurückzuliefern. Ist der Server so konfiguriert, daß alle Dateien im Verzeichnisbaum */cgi* als CGI-Skripten erkannt werden, so wird das Skript *willkommen.cgi* ausgeführt, anstatt den Inhalt der Datei direkt an den Browser zurückzuschicken.

CGI-Programme erhalten ihre Eingaben über die Standardeingabe (STDIN) und aus Umgebungsvariablen. Diese Variablen enthalten Informationen über den entfernten Rechner (remote host), den Benutzer, die übergebenen Formulardaten (sofern vorhanden) usw. Außerdem werden hier Informationen über den Namen des Servers, das verwendete Kommunikationsprotokoll sowie den Namen der verwendeten Serversoftware gespeichert. In Kapitel 3*, Das Common Gateway Interface*, werden wir uns die Umgebungsvariablen genauer anschauen.

Läuft das CGI-Programm, gibt es seine Ausgaben über die Standardausgabe (STDOUT) zurück. In Perl ist das ganz einfach, da alles, was Sie per *print* ausgeben, standardmäßig an STDOUT weitergereicht wird. CGI-Programme können sowohl ganze HTML-Dokumente zurückliefern als auch einen URL, auf den der Request weitergeleitet werden soll. Hierfür geben CGI-Skripten eine gemäß dem HTTP formatierte Zeile zurück, die der Webserver entsprechend interpretieren kann. Die Verwendung von HTTP-Headern werden wir im folgenden Kapitel eingehend behandeln. Mit dem folgenden Header signalisiert unser Beispielskript die Ausgabe von HTML-Code:

```
Content-type: text/html
```

CGI-Skripten können, wenn nötig, zusätzliche Header-Zeilen zurückliefern. Mit der Ausgabe einer leeren Zeile signalisiert das Skript das Ende des Header-Blocks. Falls ein Dokument ausgegeben werden soll, folgt nun der Inhalt dieses Dokuments.

Der Webserver übernimmt jetzt die Ausgabe des Skripts, fügt seine eigenen HTTP-Header hinzu und schickt alles zusammen an den Browser des Benutzers zurück, von dem der Request kam. Beispiel 1-2 zeigt, wie ein solcher HTTP-Response aussehen kann:

Beispiel 1-2: Beispiel für einen HTTP-Response

```
HTTP/1.1 200 OK
Date: Sat, 18 Mar 2000 20:35:35 GMT
Server: Apache/1.3.9 (Unix)
Last-Modified: Wed, 20 May 1998 14:59:42 GMT
ETag: "74916-656-3562efde"
Content-Length: 852
Content-Type: text/html

<HTML>
<HEAD>
  <TITLE>Willkommen bei der Datenbank von Mike's Mechanics</TITLE>
</HEAD>
```

Beispiel 1-2: Beispiel für einen HTTP-Response (Fortsetzung)

```
<BODY BGCOLOR="#ffffff">
  <IMG SRC="/images/mike.jpg" ALT="Mike's Mechanics">
  <P>Willkommen, Benutzer von user14.my-isp.net. Hier finden Sie eine Liste
    von Mechanikern aus dem ganzen Land und den Dienstleistungen, auf die
    diese spezialisiert sind. Diese Liste wurde von unseren Benutzern
    zusammengetragen.</P>
  <P>Worauf warten Sie noch? Klicken Sie <A HREF="/cgi/list.cgi">hier</A></P>
  <HR>
  <P>Die aktuelle Zeit auf unserem Server ist: Sat Mar 18 20:35:35 2000.</P>
  <P>Falls Sie mit dieser Site Probleme haben sollten oder uns
    Verbesserungsvorschl&auml;ge mitteilen wollen, schicken Sie eine E-Mail an
    <A HREF="mailto:webmaster@mikesmechanics.com">webmaster@mikesmechanics.com
    </A>.</P>
</BODY>
</HTML>
```

Der Header enthält die Bezeichnung des verwendeten Kommunikationsprotokolls, das Datum und die Uhrzeit der Serverantwort, den Namen und die Versionsnummer der verwendeten Serversoftware, die Zeit der letzten Änderung des Dokuments, Anweisungen, die für das Caching wichtig sind, die Länge des Nachrichten-Body der Antwort und den verwendeten Dokumententyp. In unserem Fall handelt es sich um ein HTML-formatiertes Textdokument. Header wie diese werden bei jeder Serverantwort zurückgeliefert. Wir werden uns diese Header im folgenden Kapitel genauer anschauen. Bitte beachten Sie, daß in den Headern keinerlei Hinweis darauf mitgeliefert wird, ob es sich hierbei um eine statische oder eine von einem CGI-Skript erzeugte, dynamische HTML-Datei handelt. So soll das auch sein. Der Browser hat eine Ressource angefordert, und diese Ressource bekommt er auch. Hierbei ist es dem Browser egal, wo das Dokument herkommt und wie es vom Webserver generiert wurde.

Mit CGI können Sie Ausgaben erzeugen, die für den Endbenutzer nicht anders aussehen als andere Ressourcen im Web. Das bedeutet, daß Sie per CGI all das dynamisch erzeugen können, was der Webserver sich auch aus einer Datei holen kann. Das kann alles mögliche sein, beispielsweise HTML-Dokumente, Text- und PDF-Dateien, selbst Bilder wie PNGs oder GIFs. Wie Sie dynamische Grafiken erzeugen können, zeigen wir in Kapitel 13*, Dynamische Grafiken*.

Beispiel-CGI

Lassen Sie uns nun ein Beispiel für eine CGI-Anwendung in Perl anschauen, die uns die dynamische Ausgabe erzeugt, die wir in Beispiel 1-2 gerade gesehen haben. Das Programm (siehe Beispiel 1-3) ermittelt, von wo aus der Benutzer eine Verbindung zu unserem Server aufbaut, und erzeugt dann ein einfaches HTML-Dokument, welches diese Information zusammen mit der aktuellen Zeit enthält. In den folgenden Kapiteln werden wir sehen, wie die verschiedenen CGI-Module dazu benutzt werden können, solch eine Anwendung noch einfacher zu entwickeln. Im Moment wollen wir jedoch bei einem schlichten Beispiel bleiben.

Beispiel 1-3: willkommen.cgi

```
#!/usr/bin/perl -wT

use strict;

my $zeit        = localtime;
my $remote_id   = $ENV{REMOTE_HOST} || $ENV{REMOTE_ADDR};
my $admin_email = $ENV{SERVER_ADMIN};

print "Content-type: text/html\n\n";

print <<ENDE_DER_SEITE;
<HTML>
<HEAD>
  <TITLE>Willkommen bei der Datenbank von Mike's Mechanics</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff">
  <IMG SRC="/images/mike.jpg" ALT="Mike's Mechanics">
  <P>Willkommen, Benutzer von $remote_id. Hier finden Sie eine Liste
    von Mechanikern aus dem ganzen Land und den Dienstleistungen, auf die
    diese spezialisiert sind. Diese Liste wurde von unseren Benutzern
    zusammengetragen.</P>
  <P>Worauf warten Sie noch? Klicken Sie <A HREF="/cgi/list.cgi">hier</A></P>
  <HR>
  <P>Die aktuelle Zeit auf unserem Server ist: $zeit.</P>
  <P>Falls Sie mit dieser Site Probleme haben sollten oder uns
    Verbesserungsvorschl&auml;ge mitteilen wollen, schicken Sie eine E-Mail an
    <A HREF="mailto:$admin_email">$admin_email</A>.</P>
</BODY>
</HTML>
ENDE_DER_SEITE
```

Dieses Programm ist verhältnismäßig einfach gehalten. Es werden sechs Anweisungen benutzt, auch wenn die letzte sich über mehrere Zeilen erstreckt. Lassen Sie uns einmal sehen, wie das funktioniert. Da dies unser erstes Skript ist und es kurz ist, wollen wir es Zeile für Zeile durchgehen. Wie wir im *Vorwort* bereits gesagt haben, sollten Sie schon mit Perl vertraut sein. Falls Sie keine Perl-Kenntnisse besitzen oder diese schon etwas eingerostet sind, sollten Sie eine Perl-Referenz zur Hand haben, um das eine oder andere beim Lesen dieses Buches nachschlagen zu können. Wir empfehlen Ihnen *Programmieren mit Perl* von Larry Wall, Tom Christiansen und Jon Orwant (O'Reilly). Dieses Buch gilt als das Standardwerk und enthält unter anderem eine hilfreiche alphabetische Liste der in Perl eingebauten Funktionen.

Die erste Zeile sieht aus wie bei den meisten Perl-Skripten. Sie teilt dem Server mit, das Programm im Pfad */usr/bin/perl* zu benutzen, um das Skript zu interpretieren und auszuführen. Die am Ende der Zeile benutzten Flags *-wT* weisen Perl an, zusätzliche Warnungen einzuschalten und das Taint-Checking zu benutzen. Die zusätzlichen Warnun-

gen helfen uns, Probleme zu finden, die normalerweise keine Syntaxfehler erzeugen. Dieses Flag ist zwar optional, hat sich aber als sehr hilfreich bei der Fehlersuche und -vermeidung erwiesen. Taint-Checks sollten Sie nicht als optional ansehen. Wenn Sie nicht gerade sehr gefährlich leben wollen, sollten Sie dieses Feature bei allen CGI-Skripten benutzen. Taint-Checks werden wir in Kapitel 8*, Sicherheit*, genauer betrachten.

Die Anweisung *use strict* teilt Perl mit, strenge Regeln für die Benutzung von Variablen, Subroutinen und Referenzen anzuwenden. Wenn Sie dieses Kommando bisher nicht benutzt haben, dann sollten Sie es sich für CGI-Skripten angewöhnen. Wie die Warnungen hilft diese Anweisung Ihnen, Fehler zu vermeiden, die normalerweise keine Meldungen generieren würden. Außerdem bringt das *strict*-Pragma Sie dazu, sich einen guten Programmierstil anzugewöhnen, da Sie bei dessen Verwendung Variablen deklarieren müssen und die Anzahl globaler Variablen gering halten. Dadurch schreiben Sie Code, der sich leichter pflegen läßt. Wie wir in Kapitel 17*, Effizienz und Optimierung*, sehen werden, ist das *strict*-Pragma für die Verwendung von FastCGI und *mod_perl* obligatorisch. Wenn Sie daran denken, in der Zukunft eines dieser Verfahren zu benutzen, sollten Sie jetzt damit anfangen, *strict* zu verwenden.

Jetzt können wir mit der richtigen Arbeit beginnen. Zuerst deklarieren wir drei Variablen. Der ersten Variablen, `$zeit`, weisen wir einen String zu, der das aktuelle Datum und die Uhrzeit enthält. Die zweite Variable, `$remote_id`, wird auf den Wert gesetzt, mit dem sich der entfernte Rechner identifiziert. Diesen Wert lesen wir aus den Umgebungsvariablen REMOTE_HOST oder REMOTE_ADDR. Wie bereits gesagt wurde, erhalten CGI-Skripten ihre Informationen aus den Umgebungsvariablen und der Standardeingabe (STDIN). REMOTE_HOST enthält den vollständigen Domainnamen des entfernten Rechners. Dies klappt allerdings nur dann, wenn der Webserver dafür eingerichtet ist, zu einer IP-Adresse den Hostnamen zu bestimmen (»reverse domain name lookups«). Ansonsten ist diese Umgebungsvariable leer. Falls das der Fall ist, benutzen wir die Umgebungsvariable REMOTE_ADDR, die die IP-Adresse des entfernten Rechners enthält. Der letzten Variablen, `$admin_email`, weisen wir den Inhalt der Umgebungsvariablen SERVER_ADMIN zu. SERVER_ADMIN enthält die E-Mail-Adresse des für unseren Server zuständigen Administrators, wie sie in den Konfigurationsdateien des Servers eingetragen ist. Wir benutzen hier nur ein paar der Umgebungsvariablen, die CGI-Skripten zur Verfügung stehen. Diese und die übrigen Umgebungsvariablen werden wir in Kapitel 3*, Das Common Gateway Interface*, detailliert behandeln.

Wie wir bereits gesehen haben, muß das CGI-Skript zum Ausgeben eines neuen Dokuments zuerst einen entsprechenden Header ausgeben, der definiert, welches Dokumentenformat verwendet wird. Nach der obligatorischen Leerzeile, die das Ende der Header anzeigt, folgt das eigentliche Dokument.

Anstatt jede Zeile des Dokuments mit einem *print* separat auszugeben, benutzen wir ein »Here«-Dokument, mit dessen Hilfe wir einen Textblock als Ganzes ausgeben können. Dieses Feature von Perl ist Ihnen vielleicht nicht so vertraut, wenn Sie sich nicht mit der Shell-Programmierung auskennen. Diese Anweisung teilt Perl mit, alle nachfol-

genden Zeilen bis zur Marke `ENDE_DER_SEITE` (die allein auf einer Zeile stehen muß) auszugeben, als hätte dieser Text bei einem normalen *print* in doppelten Anführungszeichen gestanden (d.h. mit Auswertung der Variablen), wobei doppelte Anführungszeichen selbst nicht gesondert behandelt (mit Escape-Zeichen geschützt) werden müssen. »Here«-Dokumente ersparen uns eine Menge zusätzlicher Tipperei und tragen deutlich zur Lesbarkeit unserer Programme bei. Trotzdem gibt es noch bessere Methoden, um HTML auszugeben, wie wir in Kapitel 5*, CGI.pm*, und Kapitel 6*, HTML-Templates*, sehen werden.

Danach wird das Programm beendet. Der Webserver fügt nun zusätzliche HTTP-Header hinzu und schickt die Antwort an den Client zurück (siehe Beispiel 1-2). Dies war nur ein einfaches Beispiel für ein CGI-Skript. Falls Sie jetzt noch Fragen dazu haben oder bestimmte Details nicht genau verstanden haben, machen Sie sich bitte keine Sorgen. Die Vielzahl an Verweisen auf die weiterführenden Kapitel zeigt, daß wir den Rest dieses Buches damit verbringen werden, diese Fragen zu beantworten und Unklarheiten zu beseitigen.

Wie man CGI-Skripten aufruft

Genau wie HTML-Dokumente und andere Ressourcen im Web haben CGI-Skripten ihre eigenen URLs. Ist der Webserver entsprechend konfiguriert, werden bestimmte virtuelle Verzeichnisse (d.h. solche, die Teil einer URL sind), wie z.B. */cgi-bin*, */cgi*, */scripts* etc., auf die CGI-Skripten abgebildet. Sowohl der URL-Pfad als auch der dazugehörige lokale Verzeichnisname können in der Serverkonfiguration eingestellt werden. Wie das beim Apache-Webserver gemacht wird, zeigen wir im Abschnitt »Konfigurieren von CGI-Skripten« etwas weiter unten.

Bei Unix-basierten Systemen wird zwischen ausführbaren und nichtausführbaren Dateien unterschieden. CGI-Skripten müssen ausführbar sein. Um zum Beispiel die Datei *mein_skript.cgi* ausführbar zu machen, müssen Sie auf der Kommandozeile folgendes Kommando eingeben:

```
chmod 0755 mein_skript.cgi
```

Es ist ein häufiger Fehler, diesen Schritt zu vergessen. Auf anderen Systemen müssen Sie eventuell andere Schritte unternehmen, um die Skripten ausführen zu können. In der Dokumentation zu Ihrem Webserver sollten Sie die nötigen Informationen dazu finden.

Alternative Technologien

Wie schon der Titel unseres Buches zeigt, geht es hier um CGI-Programme in Perl. Manche Leute sind sich nicht im klaren darüber, worin der Unterschied zwischen beidem besteht. Perl ist eine Programmiersprache, während CGI eine Schnittstelle ist, über die Programme die Anfragen eines Webservers bearbeiten können. Für beides gibt es Alter-

nativen. Dynamische Requests können mit verschiedenen Methoden bearbeitet werden, und CGI-Programme können in einer Vielzahl von Programmiersprachen geschrieben werden.

Warum Perl?

Obwohl CGI-Anwendungen in fast jeder Programmiersprache geschrieben werden können, gehören CGI und Perl für viele Entwickler einfach zusammen. Der erste Webmaster von Sun Microsystems, Hassan Schroeder, hat einmal gesagt: »Perl ist das Tesaband des Internet«. Perl ist die am weitesten verbreitete Sprache, die für die CGI-Programmierung benutzt wird. Dafür gibt es eine Menge guter Gründe:

- Perl ist einfach zu lernen, da es Elemente der verschiedensten Programmiersprachen (wie z.B. C) in sich vereint und dem Benutzer beim Auftreten von Fehlern mit präzisen und detaillierten Meldungen hilft, diese zu finden.
- Perl erlaubt ein schnelles Entwickeln, da es interpretiert wird. Es ist nicht notwendig, den Quellcode vor dem Ausführen zu kompilieren.
- Perl ist leicht zu portieren und steht für viele Plattformen zur Verfügung.
- Perl hat sehr mächtige Funktionen zum Bearbeiten von Zeichenketten. Reguläre Ausdrücke und Ersetzungsmechanismen sind schon in die Sprache eingebaut.
- Perl kann mit binären Daten genauso leicht umgehen wie mit Text.
- Perl hat keine strikten Variablentypen. Zahlen, Zeichenketten (Strings) und Boolesche Werte sind einfache Skalare.
- Perl macht das Zusammenarbeiten mit externen Programmen sehr leicht und bringt seine eigenen Dateisystem-Funktionen mit.
- Im CPAN finden sich zahllose Open-Source-Module für Perl. So gibt es beispielsweise Module für das Erzeugen dynamischer Grafiken, Module, die Schnittstellen zu Internetservern bieten, und solche, mit denen man auf Datenbanken zugreifen kann. Weitere Informationen zum CPAN finden Sie in Anhang B, *Perl-Module*.

Außerdem ist Perl schnell. Streng gesehen ist Perl keine reine Interpretersprache. Nachdem Perl den Quellcode aus einer Datei gelesen hat, wird dieser Code in Opcodes übersetzt und dann ausgeführt. Da Kompilierung und Ausführung normalerweise zusammen ablaufen, sehen Sie sie im allgemeinen nicht als voneinander getrennte Schritte. Perl liest den Quellcode, kompiliert ihn, führt ihn aus und beendet dann den Prozeß. Dieser Vorgang wird bei jeder erneuten Ausführung eines Perl-Skripts, auch bei CGI-Skripten, wiederholt. Perl ist effizient genug, um dieses Verfahren bei allen Websites schnell genug durchzuführen, ausgenommen diejenigen mit einem sehr hohen Besucheraufkommen. Unter Windows-Systemen kann die Effizienz etwas leiden, da das Erzeugen neuer Prozesse auf diesen Systemen zu einer höheren Systembelastung als unter Unix führen kann.

Alternativen zu CGI

Während der letzten Jahre sind einige Alternativen zu CGI aufgetaucht. Sie alle verfolgen letztendlich das gleiche Ziel wie CGI: Requests zu beantworten und per HTTP dynamische Inhalte bereitzustellen. Die meisten versuchen, den größten Nachteil von CGI zu vermeiden, daß nämlich bei jedem Aufruf eines Skripts ein neuer Prozeß erzeugt wird. Andere Technologien wiederum unterscheiden nicht mehr so sehr zwischen HTML-Seiten und Programmcode, indem sie diesen Code direkt in die HTML-Seiten einbetten. Wir werden diese Idee in Kapitel 6 weiterverfolgen. Im folgenden finden Sie eine Liste der wichtigsten Alternativen zu CGI:

ASP

Active Server Pages, ASP, sind ursprünglich eine Entwicklung von Microsoft für den IIS, die Webserver-Software von Microsoft. Mittlerweile ist ASP aber auch für andere Server erhältlich. Der ASP-Interpreter ist in den Webserver einkompiliert, so daß kein zusätzlicher Prozeß erzeugt werden muß. ASP-Anweisungen können direkt in den HTML-Seiten stehen. Es müssen keine separaten Programme geschrieben werden. In Kapitel 6 stellen wir Module vor, mit denen Sie ähnliche Dinge mit CGI realisieren können. ASP unterstützt verschiedene Programmiersprachen, wobei die beliebteste hier Visual Basic ist. Allerdings wird auch JavaScript unterstützt, und ActiveState bietet eine Windows-Variante von Perl an, die mit ASP benutzt werden kann. Des weiteren gibt es das Perl-Modul Apache::ASP, das eine ASP-Unterstützung für *mod_perl* bietet.

PHP

PHP ist eine Perl-ähnliche Programmiersprache. Der PHP-Interpreter wird, wie bei ASP, in den Webserver eingebettet. Mit PHP können Sie Programmcode in HTML-Seiten einbauen. PHP wird zum Beispiel vom Apache-Webserver unterstützt.

ColdFusion

ColdFusion von Allaire unterscheidet stärker zwischen Code und HTML-Seiten als PHP. Bei ColdFusion können HTML-Seiten zusätzliche Tags enthalten, über die dann Funktionen von ColdFusion aufgerufen werden. Neben den bereits vorhandenen Standardfunktionen von ColdFusion können Entwickler ihre eigenen Kontrollelemente und Funktionen hinzufügen. Auch wenn ColdFusion ursprünglich für Windows entwickelt wurde, gibt es mittlerweile auch Versionen für verschiedene Unix-Ableger. Der ColdFusion-Interpreter wird ebenfalls in den Webserver integriert.

Java Servlets

Java Servlets sind eine Entwicklung der Firma Sun. Servlets sind CGI insofern ähnlich, als daß sie aus Code bestehen, der dann die Dokumente erzeugt. Da die Servlets in Java programmiert sind, müssen ihre Klassen zuerst auf dem Webserver kompiliert werden, bevor sie ausgeführt werden. Die Klassen werden beim Ausführen vom Server dynamisch geladen. Die Schnittstelle selbst verhält sich jedoch ganz anders als CGI. JavaServerPages (JSP) sind eine ASP-ähnliche Technik, mit der Java-Code in HTML-Seiten eingebettet werden kann.

FastCGI

FastCGI hält eine oder mehrere Instanzen von *perl* im Arbeitsspeicher. Diese werden nach dem Ausführen von Skripten nicht beendet, sondern laufen weiter. Eine spezielle Schnittstelle ermöglicht das Bearbeiten dynamischer Requests, die vom Webserver übergeben wurden. Der Hauptnachteil von CGI, das Erzeugen zusätzlicher Prozesse, wird bei FastGGI vermieden. FastCGI ist in großen Teilen zu CGI kompatibel und ist für eine Vielzahl von Webservern erhältlich. Wir werden FastCGI in Kapitel 17*, Effizienz und Optimierung*, genauer betrachten.

mod_perl

mod_perl ist ein Modul für den Apache-Webserver. Der Ansatz ist ähnlich wie bei FastCGI. Allerdings wird bei *mod_perl* der Perl-Interpreter in den Webserver einkompiliert. Das erhöht die Performance und gibt Perl-Code, der für *mod_perl* geschrieben ist, Zugriff auf die Interna des Apache. Wir werden *mod_perl* in Kapitel 17 detaillierter vorstellen.

Auch wenn eine Zunahme dieser alternativen Techniken zu verzeichnen ist, so bleibt CGI doch die beliebteste Methode, um dynamische Seiten zu erzeugen. CGI wird so schnell nicht verschwinden, auch wenn die Marketing-Sprüche einiger Mitbewerber gern etwas anderes behaupten. Selbst wenn Sie sich überlegen, eine alternative Methode einzusetzen, ist es hilfreich, sich mit CGI auszukennen. Weil es eine recht schlichte Schnittstelle ist, bekommen Sie beim Lernen von CGI viel über die Funktionsweise der dem Web zugrundeliegenden Techniken mit. Das erhöht wiederum das Verständnis für die Techniken, die auf diesem Fundament aufbauen. Dazu kommt, daß CGI universell verfügbar ist. Bei vielen Alternativen zu CGI müssen Sie erst bestimmte Softwarekomponenten zusätzlich zum Webserver installieren. CGI wird dagegen bei fast allen Webservern »gleich mitgeliefert«, und das wird auch in Zukunft so bleiben.

Den Webserver konfigurieren

Bevor Sie CGI-Skripten auf Ihrem Server ausführen können, müssen einige Einstellungen an der Konfiguration des Webservers vorgenommen werden. In diesem Buch benutzen wir für unsere Beispiele den Apache auf einer Unix-Plattform. Apache ist bei weitem die beliebteste Webserver-Software, die es gibt. Außerdem ist Apache Open Source Software und kostenlos erhältlich. Apache ist eine Weiterentwicklung des NCSA-Webservers. Viele Webserver, die aus dem NCSA-Server hervorgegangen sind, sind sich in der Konfiguration sehr ähnlich, wie z.B. die von iPlanet verkauften.

Wir gehen davon aus, daß Sie bereits Zugang zu einem funktionierenden Webserver haben. Wir werden also nicht darauf eingehen, wie man den Apache neu installiert und vorkonfiguriert. Eine ausführliche Darstellung dieses Themas würde den Rahmen dieses Buches sprengen. Eine gute Anleitung finden Sie beispielsweise in dem Buch *Apache* von Ben und Peter Laurie (O'Reilly).

Je nach System kann sich der Ort im Dateisystem, an dem der Apache installiert ist, unterscheiden. In unserem Buch werden wir den Standard-Installationspfad benutzen. Hierbei werden die nötigen Installationen unterhalb von */usr/local/apache* vorgenommen. Welche Unterverzeichnisse das im einzelnen sind, können Sie folgendermaßen herausfinden:

```
$ cd /usr/local/apache
$ ls -F
bin/  cgi-bin/  conf/  htdocs/  icons/  include/  libexec/  logs/  man/  proxy/
```

Je nachdem, wie der Apache bei der Installation konfiguriert wurde, kann es sein, daß einige Verzeichnisse, etwa *libexec* oder *proxy* fehlen. Das ist für uns aber nicht so schlimm. Bei einigen Unix- oder Unix-verwandten Systemen (z.B. bei einigen Linux-Distributionen) kann es sein, daß die Verzeichnisse über das System verteilt sind. Tabelle 1-1 zeigt, wie die Verzeichnisse unter RedHat Linux angelegt werden:

Tabelle 1-1: Alternative Pfade zu wichtigen Apache-Verzeichnissen

Standardinstallationspfad	Alternativer Pfad (RedHat Linux)
/usr/local/apache/cgi-bin	*/home/httpd/cgi-bin*
/usr/local/apache/htdocs	*/home/httpd/html*
/usr/local/apache/conf	*/etc/httpd/conf*
/usr/local/apache/logs	*/var/log/httpd*

Wenn dies der Fall ist, müssen Sie die Pfade in unseren Anleitungen entsprechend auf Ihre Situation übertragen. Wenn der Apache auf Ihrem System installiert ist, sich die entsprechenden Verzeichnisse aber nicht an einem der oben genannten Orte befinden, fragen Sie am besten Ihren Systemadministrator, oder schauen Sie in der Dokumentation für Ihr System nach.

Der Apache wird konfiguriert, indem die Dateien im *conf*-Verzeichnis modifiziert werden. Diese Dateien enthalten Anweisungen, die der Apache beim Programmstart ausliest. Ältere Versionen des Apache benutzen drei Dateien für die Konfiguration. Diese sind: *httpd.conf*, *srm.conf* und *access.conf*. In den neueren Versionen wird auf die Benutzung von *srm.conf* und *access.conf* verzichtet. Statt dessen befinden sich alle für die Konfigurierung nötigen Informationen nun in der Datei *httpd.conf*. Sicherheitslükken, die durch Unstimmigkeiten zwischen den verschiedenen Dateien entstehen könnten, werden so vermieden.

Dennoch benutzen viele Websites auch heute noch drei Konfigurationsdateien. Daher werden wir in diesem Buch jeweils die entsprechende Datei nennen, die geändert werden muß.

Damit der Apache die Änderungen auch übernimmt, müssen die Konfigurationsdateien neu eingelesen werden. Hierfür muß der Webserver nicht komplett neu gestartet werden, obwohl das auch funktioniert. Wenn auf Ihrem System das Kommando *apachectl*

installiert ist (Teil der Standardinstallation), können Sie den Apache mit folgendem Kommando anweisen, die Konfiguration neu einzulesen:

```
$ apachectl graceful
```

Hierfür benötigen Sie eventuell Superuser-(d.h. *root*-)Rechte.

Apache für die Ausführung von CGI-Skripten konfigurieren

Apache läßt sich recht einfach für die Ausführung von CGI-Skripten einrichten. Allerdings gibt es ein gutes und ein weniger gutes Verfahren, dies zu tun. Schauen wir uns zuerst das bessere Verfahren an. Hierbei liegen unsere CGI-Skripten in einem speziellen Verzeichnis:

Konfigurieren nach Verzeichnis

Die Direktive `ScriptAlias` weist den Webserver an, einen virtuellen Pfad (den im URL) einem lokalen Verzeichnis zuzuordnen und Dateien in diesem Verzeichnis als CGI-Skripten auszuführen.

Damit unser Webserver CGI-Skripten ausführen kann, tragen wir folgende Direktive in die Datei *httpd.conf* ein:

```
ScriptAlias          /cgi        /usr/local/apache/cgi-bin
```

Wenn nun ein Benutzer auf den URL

http://www.meinserver.de/cgi/meinskript.cgi

zugreift, wird das lokale Programm

```
/usr/local/apache/cgi-bin/meinskript.cgi
```

von unserem Server ausgeführt. Hierbei muß der Pfad *cgi* im URL nicht der gleiche sein wie der tatsächliche Name *cgi-bin* im Dateisystem. Ob Sie hierbei den virtuellen Pfad *cgi* auf das Verzeichnis *cgi-bin* abbilden oder andere Pfade benutzen, ist allein Ihre Entscheidung. Sie können auch mehrere Verzeichnisse für das Ausführen von CGI-Skripten einrichten. Hierfür fügen Sie einfach eine weitere `ScriptAlias`-Direktive in die Datei *httpd.conf* ein:

```
ScriptAlias          /cgi        /usr/local/apache/cgi-bin/
ScriptAlias          /cgi2       /usr/local/apache/cgi2-bin/
```

Das Verzeichnis, in dem Sie CGI-Skripten ablegen, muß sich außerhalb des Wurzelverzeichnisses des Webservers befinden.[1] Bei der Standardinstallation des Apache entspricht das Wurzelverzeichnis dem Verzeichnis *htdocs*. Sämtliche Dateien in diesem Verzeichnis können von außen eingesehen werden. Standardmäßig befindet sich das *cgi-bin*-Verzeichnis nicht unterhalb von *htdocs*, so daß ein Entfernen der `ScriptAlias`-Direktive den

1 Welches Verzeichnis das ist, wird mit Hilfe der `DocumentRoot`-Direktive in der Datei *httpd.conf* festgelegt. – Anm.d.Ü.

Zugriff auf die CGI-Skripten wirksam unterbindet. Hierfür gibt es einen guten Grund, der aber nicht allein darin besteht, zu verhindern, daß jemand versehentlich die `ScriptAlias`-Direktive entfernt.

Das folgende Beispiel zeigt, warum es gefährlich sein kann, das Verzeichnis für CGI-Skripten im Wurzelverzeichnis des Webservers anzulegen. Nehmen wir an, Sie wollten mehrere Verzeichnisse für CGI-Skripten haben, die unterhalb des Wurzelverzeichnisses liegen, beispielsweise für jede größere Anwendung eines. Stellen wir uns nun vor, Sie hätten einen Online-Shop für *Dinge*, für den die CGI-Skripten im Verzeichnis */usr/local/apache/htdocs/dinge/cgi* liegen sollen. In Ihrer *httpd.conf* müßten Sie dann folgenden Eintrag vornehmen:

```
ScriptAlias      /dinge-cgi    /usr/local/apache/htdocs/dinge/cgi
```

Ein Test mit dieser Konfiguration verliefe erfolgreich. Nun will Ihre Firma zusätzlich zu Dingen auch noch Sachen verkaufen. Also braucht der Shop einen allgemeineren Namen. Sie benennen also das Verzeichnis *dinge* in *shop* um. Sie passen die `ScriptAlias`-Direktive entsprechend an, ändern alle relevanten HTML-Links und legen einen symbolischen Link an, der von *dinge* auf *shop* zeigt, um die Benutzer nicht zu vergraulen, die ein Lesezeichen auf den alten Pfad haben. Ist doch sinnvoll, oder?

Leider hat uns dieser letzte Schritt gerade eine große Sicherheitslücke beschert. Das Problem besteht darin, daß es nun möglich ist, Ihre CGI-Skripten über zwei verschiedene URLs aufzurufen. Hätten Sie beispielsweise ein Skript mit dem Namen *einkauf.cgi*, so könnte es nun auf zwei Arten aufgerufen werden:

http://localhost/store-cgi/einkauf.cgi
http://localhost/widgets-cgi/einkauf.cgi

Für den ersten URL existiert eine `ScriptAlias`-Direktive, für den zweiten nicht. Ruft ein Benutzer das Skript nun über den zweiten URL auf, so bekommt er statt einer Webseite den Quellcode Ihres Skripts zu sehen. Wenn Sie Glück haben, informiert Sie jemand von diesem Problem. Wenn nicht, so könnte ein böswilliger Benutzer versuchen, anhand des Quellcodes Sicherheitslücken zu finden, um in Ihr System einzubrechen und noch wertvollere Informationen (wie etwa Datenbankpaßwörter oder Kreditkartennummern) herauszubekommen.

Jeder symbolische Link oberhalb Ihres CGI-Verzeichnisses kann dieses Sicherheitsproblem erzeugen.[2] Das Szenario, in dem ein Verzeichnis umbenannt und ein Link auf dessen alten Namen angelegt wird, ist nur ein mögliches Beispiel dafür, wie es ungewollt zu einer solchen Situation kommen kann. Wenn Sie Ihre CGI-Skripten außerhalb des Wurzelverzeichnisses des Webservers ablegen, brauchen Sie sich darum keine Sorgen mehr zu machen.

2 Eine weitere Lösungsmöglichkeit wäre, den Apache so zu konfigurieren, daß symbolische Links nicht umgesetzt werden. Dennoch sind symbolische Links generell ganz nützlich und standardmäßig eingeschaltet. Das Problem in dieser Situation ist nicht der Link selbst, sondern die Tatsache, daß Ihre Skripten an einem Ort liegen, der von außen eingesehen werden kann.

Vielleicht fragen Sie sich jetzt, warum das Offenbaren Ihres Quellcodes solch ein Problem sein soll. Vom Sicherheitsstandpunkt aus gesehen unterscheiden sich CGI-Skripten erheblich von anderen ausführbaren Dateien. Sie ermöglichen es entfernten anonymen Benutzern, Programme auf Ihrem System auszuführen. Daher sollte die Sicherheit auf jeden Fall sehr ernst genommen werden. Ihr Code muß absolut fehlerfrei sein, wenn Sie es potentiellen Angreifern erlauben wollen, diesen einzusehen. Das Prinzip der »security through obscurity« ist für sich genommen kein besonders guter Schutz. Kombiniert mit anderen Methoden kann es aber auch nicht schaden. Das Thema Sicherheit wird in Kapitel 8*, Sicherheit*, eingehend behandelt.

Konfiguration nach Dateiendung

Anstatt CGI-Skripten in einem gemeinsamen Verzeichnis zu halten, können Sie den Webserver auch so einstellen, daß Skripten an ihrer Dateiendung erkannt werden. Die Skripten können dann an beliebigen Stellen im Dokumentenbaum liegen. Sowohl aus Sicherheits- wie auch aus administrativen Gründen ist dies allerdings eine sehr schlechte Idee.

Vom administrativen Standpunkt aus gesehen, sollten Sie dieses Verfahren nicht anwenden, da sich Skripten, die in einem Verzeichnis liegen, leichter pflegen lassen. Je größer eine Website wird, um so schwieriger ist es, den Überblick über alle verwendeten Skripten zu behalten. Wenn Sie Ihre Skripten in einem Verzeichnis ablegen, sind diese leichter aufzufinden und motivieren gleichzeitig dazu, übersichtliche Lösungen für mehrere Problemstellungen zu entwickeln, anstatt mit einer Handvoll Wegwerf-Skripten zu hantieren. Sie können unterhalb Ihres */cgi*-Verzeichnisses Unterverzeichnisse anlegen, die die Organisation Ihrer Skripten erleichtern.

Die Konfigurierung per Dateiendung ist aus zwei Gründen unsicher. Erstens kann dadurch jeder, der HTML-Dateien bearbeiten darf, auch CGI-Skripten anlegen. Wie bereits gesagt, sollte man bei CGI-Skripten den Sicherheitsaspekt bedenken. Programmierneulinge sollten keine CGI-Skripten auf tatsächlich im Web laufenden Servern anlegen dürfen. Zweitens erhöht dieses Verfahren die Wahrscheinlichkeit, daß jemand Ihren Quellcode einsehen kann. Viele Texteditoren legen beim Bearbeiten einer Datei ein Backup dieser Datei an. Dieses Backup wird von manchen Editoren in Ihrem gegenwärtigen Arbeitsverzeichnis angelegt. Wenn Sie beispielsweise mit dem Programm *emacs* an der Datei *top_secret.cgi* arbeiten, so legt *emacs* eine Backup-Datei mit dem Namen *top_secret.cgi~* an. Wenn diese Datei auf Ihren im Web laufenden Server gelangt und jemand mit etwas Glück diese Datei aufruft, so wird der Webserver die Dateiendung nicht erkennen und statt der Ausgabe Ihres Skripts dessen Quellcode zurückliefern.

Bestenfalls wird Ihr Texteditor diese Dateien nach der Arbeit wieder löschen. Dennoch sollten Sie Dateien nicht direkt auf einem im Netz laufenden Server bearbeiten. Manchmal werden Dateien von Hand umbenannt. Angenommen, ein Entwickler möchte etwas an einer Datei ändern, aber eine Backup-Kopie dieser Datei behalten, so versieht er die Kopie mit der Endung *.bak*. Liegt die Backup-Datei nun in einem Verzeichnis, für

das eine `ScriptAlias`-Direktive existiert, so wird das Backup nicht angezeigt, sondern ausgeführt, wie jedes andere CGI-Skript in diesem Verzeichnis auch, was wesentlich sicherer ist.

Sollte Ihr Webserver also so konfiguriert sein, daß CGI-Skripten überall ausgeführt werden können, zeigen wir Ihnen hier, wie Sie das ändern können. Die folgende Zeile weist den Webserver an, jede Datei mit der Endung *.cgi* als CGI-Skript auszuführen:

```
AddHandler    cgi-script    .cgi
```

Sie können diese Zeile auskommentieren, indem Sie ihr, genau wie in Perl, ein `#` voranstellen. Nun behandelt der Apache Dateien mit der Endung *.cgi* als unbekannten Dateityp und gibt sie zurück, als entsprächen sie dem Standardmedientyp – in der Regel *text/plain*. Stellen Sie also sicher, daß Sie alle CGI-Skripten aus dem Wurzelverzeichnis entfernt haben, bevor Sie diese Direktive auskommentieren.

Sie können das Ausführen von CGIs auch für bestimmte Verzeichnisse unterbinden, indem Sie die `ExecCGI`-Option für das betreffende Verzeichnis entfernen. Die Zeile, in der diese Anweisung steht, sieht folgendermaßen aus:

```
<Directory "/usr/local/apache/htdocs">
  .
  .
  Options Indexes FollowSymLinks ExecCGI
  .
  .
</Directory>
```

Vermutlich befinden sich über und unter der Zeile, die die `Options`-Direktive enthält, noch eine Menge anderer Zeilen, und der genaue Wortlaut der `Options`-Direktive ist auf Ihrem System ein etwas anderer. Wenn Sie die Option `ExecCGI` entfernen, so wird der Apache Skripten in diesem Verzeichnis, in unserem Beispiel im Wurzelverzeichnis */usr/local/apache/htdocs*, nicht ausführen, selbst wenn das Ausführen über die Dateiendung eigentlich eingeschaltet ist. Statt dessen sieht der Benutzer die Meldung »Permission Denied«.

Nachdem unser Webserver nun richtig konfiguriert ist, können wir uns damit beschäftigen, CGI detaillierter zu betrachten. Im nächsten Kapitel beginnen wir damit, uns HTTP, die »Sprache des Web« und die Grundlage für CGI, anzuschauen.