In diesem Kapitel:

- *URL's, URI's und URN's*
- *HTML und SGML*
- *HTTP*
- *MIME*
- *CGI*
- *Applets und Sicherheit*

3 Grundlegende Web Konzepte

3.1. Einführung

Sie werden in den folgenden Kapiteln sehen, zu was Java einsetzbar ist, was mit Java gemacht werden kann. Sehr häufig werden wir die Programme nicht als Applets programmieren. Aber sehr viele Web Anwendungen setzen Java in Form von Applets ein oder im Rahmen dynamischer Webseiten als Java Server Pages oder als Servlets. Applets müssen spezielle Sicherheitsbestimmungen beachten. Seit Java 2 (JDK1.2+) sind Sie in der Lage die Sicherheitsrichtlinien auf Ihrem Rechner sehr detailliert zu spezifizieren. Damit wurde es möglich die sogenannte Sandbox zu verlassen und komplexere interaktive Applikation in Web Applikationen einzubauen.

Das Hypertext Transfer Protokoll (HTTP) ist das Standardprotokoll für die Kommunikation eines Web Clients mit einem Web Server. HTTP selbst basiert wesentlich auf zwei andern Protokollen: MIME, Multipurpose Internet Mail Extensions und HTML, Hypertext Markup Language. MIME bietet die Möglichkeit unterschiedliche Datenformate mittels 7-bit ASCII Verbindungen zu übertragen. Der Empfänger wird auch über den übertragenen Datentyp informiert, so dass diese Daten auch passend dargestellt werden können. Wie aus dem Namen hervorgeht wurde MIME ursprünglich für den Mailbetrieb entworfen. Aber heute wird MIME viel breiter eingesetzt. HTML ist ein einfacher Standard für die Beschreibung von Dokumenten auf der semantischen Ebene. Da auch Verknüpfungen spezifiziert werden können erhielt die Sprache den Zusatz "Hypertext". Die Verbindung der unterschiedlichen Dokumentebausteine geschieht mittels Tags, die URLs, Unified Resource Locators, enthalten. Mittels URLs können Dateilokationen eindeutig spezifiziert werden. Sie benötigen vertiefte Kenntnisse dieser Grundbausteine HTTP, HTML und URL um sinnvolle Netzwerkprogramme entwickeln zu können.

3.2. URLs, URIs und URNs

Ein Uniform Resource Identifier URI beschreibt die eindeutige Lokation einer *Ressource* im Internet. Eine Resource ist in der Regel eine Datei, kann aber auch eine email Adresse, eine Meldung oder sonst was sein. Es gibt zwei Typen von URIs : URLs (Uniform Resource Locator) und URNs (Uniform Resource Names). Eine URL ist ein Zeiger auf eine Ressource im Internet an einer bestimmten Lokation, beispielsweise http://www.switch.ch oder ftp://sunsite.switch-cnlab.ch . Bei URN handelt es sich eher um ein Konzept als um eine konkrete Lösung. URNs sind Zeiger auf Ressourcen im Web, allerdings ohne auf eine spezifische Lokation zu verweisen. Was heisst dies konkret? Falls wir die URN einer Seite kennen kann der Internet Provider diese Seite auf einen andern Rechner verschieben, auf eine andere www-Adresse ohne dass dies auf die URN irgend einen Einfluss hat. URNs wurden in den RFCs RFC1738 und RFC1808 beschrieben und ist auch recht einfach zu verstehen. Aber es gibt meines Wissens keine Software, die URNs unterstützt.

Ein URL spezifiziert das Protokoll mit dem auf die Ressource zugegriffen wird, beispielsweise FTP, HTTP, MAIL... zusätzlich zur Lokation der Ressource. Ein typischer URL sieht somit folgendermassen aus:

protocol://hostname[:port]/path/filename#section

Zum Teil bezeichnet man das Protokoll auch als Schema. Typischer Protokolle sind

file	eine Datei auf der lokalen Festplatte
ftp	ein FTP (Filetransfer) Server
http	ein HTTP (Web) Server
gopher	ein Gopher Server
news	ein News (Usenet Usegroups) Server
telnet	eine Telnet-Verbindung
ldap	Zugriff auf ein LDAP Verzeichnis

In Java können Sie auch eigene Protokolle mit diesem Format definieren

Der *hostname* der URL beschreibt den Namen des Servers, auf dem die Ressource zur Verfügung steht, beispielsweise www.switch.ch .Es kann sich allerdings auch um eine IP Adresse handeln, beispielsweise 180.35.22.23. Falls Sie eine Newsgruppe angeben handelt es sich bei *hostname* um den Namen der Newsgruppe.

Der *port* ist optional. Die Angabe eines Ports ist immer dann zwingend, wenn nicht der Default Port eingesetzt wird, also beispielsweise 8080 statt 80 bei einem HTTP Server.

Der *path* zeigt zu einem spezifischen Verzeichnis auf dem angegebenen Server. Zu beachten ist, dass dabei relativ zum Root des Servers adressiert wird, nicht relativ zum Ursprung des Dateisystems. Das gesamte Dateisystem des Servers wird dem Client jedoch nicht gezeigt. Der Pfad wird auch als Dokument-Root bezeichnet. Beispielsweise könnte eine HTML Seite den URL Pfad ~joller/PVS/html besitzen; auf dem Server entspricht dieses Verzeichnis möglicherweise dem Verzeichnis c:\wroot\joller\PVS\html .

Der *filename* zeigt auf die Datei in einem bestimmten Verzeichnis. Falls die Dateiangabe fehlt ist es dem Server überlassen, welche Datei er schickt. Oft ist dies dann durch einen Standardwert vorgegeben, etwa *index.html*.

Falls *section* angegeben wird, dann bezieht sich diese Angabe auf einen "Anker" auf einer bestimmten Seite, beispielsweise

```
<A NAME="sectionID010101">Kommentar:</A>
```

Ein Zugriff auf diese Sektion der Webseite geschieht mit:

http://www.meinServer.ch/demoseite#sectionID010101

3.2.1. Relative URLs

Eine URL teilt dem Webbrowser mit, welches Protokoll, welche Datei von welchem Server geholt werden soll. Wenn wir nun auf einem Server sind, benötigen wir viele dieser Informationen nicht mehr in gleichem Masse: es bietet sich an, die weiteren Adressen relativ zur ersten anzugeben. Dies hat auch den Vorteil, dass bei einem Verschieben der Dateien auf dem Server im Falle der relativen Adressierung lediglich eine Adresse geändert werden muss.

Daher spricht man in diesen Fällen von *relativen* URLs im Gegensatz zu einem vollständigen, *absoluten* URL.

Beispiel:
auf der Seite index.html bezieht sich die Angabe der Bilddatei im Verzeichnis /images, meinEgo.gif, entweder auf die absolute Lokation (a)) oder relativ (b))

a) in index.html

```
...
<a href="../images/meinEgo.gif">Das bin ich</a>
...
```

b) in index.html

```
...
<a href="http://www.meinServer.ch/images/meinEgo.gif">Das bin ich</a>
...
```

3.3. HTML, XML und SGML

HTML ist heute noch das Standardformat für die Darstellung von Informationen auf dem Web. HTML beschreibt die Semantik der Textdaten. Die Art der Beschreibung basiert auf einem Standard, der Standard Generalized Markup Language (SGML). HTML ist eine einfache Instanz der abstrakteren SGML. SGML wurde in den 70er Jahren von Departmenet of Defense entwickelt und wurde zum ISO Standard, ISO 8879:1986.

SGML und damit auch HTML basiert auf dem Konzept: "Design by Meaning". Sie spezifizieren nicht, dass Sie Text in 18-Punkte Schrift schreiben wollen. Es genügt wenn Sie dafür den Tag <h1>...</h1> (Header Level 1) verwenden. Kursiv wird entsprechend mit ... angegeben.

Verschachtelungen sind möglich, sollten aber in korrekter Reihenfolge geschehen. Allerdings akzeptieren die meisten Browser auch unkorrekte Verschachtelungen, auch Überlappung genannt:

korrekte Verschachtelung: `<i><b>Kursiv, Fett</b></i>`
falsche Verschachtelung: `<i><b>Kursiv, Fett</i></b>`

Parameter innerhalb eines Tags werden immer dann in Anführungszeichen verlangt, falls der Parameterwert Leerzeichen enthält.

HTML 1.0 wurde als genereller Internet Standard vom IETF akzeptiert. Diese Version beschreibt den grundsätzlichen Aufbau und einige wenige wichtige Tags.

HTML 2.0 wurde als Erweiterung von HTML 1.0 definiert, speziell durch Bilder und Formulare. Diese Teile wurden vorallem durch Marc Andreesen und Eric Bina am NCSA gefördert, also der Geburtsstätte von Netscape. RFC1866 beschreibt diesen Stand der Spezifikation.

HTML 3.0 fügte Tabellengestaltung, mathematische Formeln, Stylesheets und einige weitere Möglichkeiten hinzu. Es gibt aber keinen definierten HTML 3.0 Standard.

HTML 4.0 basiert auf HTML 3.2 erweitert die Möglichkeiten von HTML 3.x, speziell für die Beschreibung mathematischer Formeln.

Sie finden Hinweise zum aktuellen Stand der HTML Spezifikation auf dem Website von W3C, dem World Wide Web Consortium.

3.4. HTTP

HTTP, das Hypertext Transport Protocol ist das Standard Protokoll für die Kommunikation zwischen Browsern und Web Servern. HTTP ist ein sogenanntes *zustandsloses* Protokoll, mit dem spezifiziert wird, wie die Kommunikation zwischen Client und Server ablaufen soll. Das heisst, das HTTP beschreibt wie Daten vom Server verlangt werden, wie der Server die Daten aufbereitet und wie der Server mit dem Client kommuniziert. HTTP verwendet TCP / IP als Basis.

HTTP 1.0 ist der definierte Standard. Das Protokoll verwendet MIME, um die Daten zu kodieren. Grundsätzlich verläuft eine Kommunikation als Sequenz der folgenden vier Schritte:

1) *Verbindungsaufbau*
   Der Client baut die Verbindung, eine TCP Verbindung, zum Server auf, zu Port 80, falls kein anderer Port definiert wurde.
2) *Absetzen der Anfrage*
   Der Client sendet eine Meldung an den Server und verlangt eine Seite an einer spezifischen URL. Das Format der Anfrage sieht folgendermassen aus:

   ```
   GET /index.html HTTP/1.0  <cr><lf><cr><lf>
   ```

   GET ist ein Schlüsselwort. /index.html ist eine relative URL zu einer Datei auf dem Server. Die volle Adresse mit Server und Protokoll ist in diesem Falle nicht nötig, weil die Anfrage ja bereits beim Server ist. Die zwei <cr><lf> schliessen die Anfrage ab.

   Eine Anfrage kann weitere Informationen umfassen, wie etwa

   ```
   Keyword: Wert
   ```

   Ein mögliches solches Schlüsselwort ist Accept:

   ```
   Accept: text/html
   Accept: text/plain
   Accept: image/gif
   Accept: image/jpeg
   ```

Ein anderes häufig verwendetes Schlüsselwort ist User-Agent. Damit angegeben, für welchen Browser der Server die Datei aufbereiten sollte, falls möglich.

```
User-Agent: Lynx/2.4 libww/2.1.4
```

Der Abschluss einer Anfrage wird mit einer Leerzeile signalisiert. Wie oben erwähnt wird eine Anfrage mit zwei Leerzeilen abgeschlossen. Eine vollständige Anfrage könnte also folgendermassen aussehen:

```
GET /index.html HTTP/1.0
Accept: text/html
Accept: text/plain
User-Agent: Lynx/2.4 libww/2.1.4
<Leerzeile>
<Leerzeile>
```

Die letzten Zeilen müssen also Leerzeilen sein. Weitere Kommandos neben GET sind HEAD ....HEAD liest lediglich die Headerinformationen der Datei. Damit kann zum Beispiel das letzte Änderungedatum abgefragt werden. POST sendet Daten eines Formulars an den Server und PUT lädt eine Datei auf den Server.
http://www.w3.org/Protocols/HTTP/Methods.html enthält eine Liste der Befehle.

3) *die Antwort*
Der Server antwortet einem Client in einem fixen Format. Die Antwort beginnt mit einem Antwortcode, gefolgt von einem MIME Header, dann folgt eine Leerzeile, gefolgt vom angeforderten Dokument und einer allfälligen Fehlermeldung.

Hier eine Beispielantwort:

```
HTTP/1.0 OK 200
Server: NCSA/1.4.2
MIME-version: 1.0
Content-type: text/html
Content-length: 107

<html>
<head>
<title>
        Beispiel in HTML
</title>
</head>

<body>
        Hier steht weiterer Text eines einfachen Textdokuments
</body>
</html>
```

Die Interpretation ist wahrscheinlich nicht sehr schwierig:
in der ersten Zeile steht das verwendete Protokoll: HTTP/1.0 und der Anwortscode OK 200. Dieser Code zeigt an, dass alles in Ordnung war. Sie finden Hinweise auf weitere Resonse Codes im Internet oder weiter unten. HTTP 1.1 und weitere Versionen werden diese Codes noch ausweiten und auch zusätzliche Codeinformation liefern.

In der nächsten Zeile wird der Server identifiziert, eigentlich handelt es sich dabei um eine generische Angabe.
http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html enthält eine Liste der Fehlercodes.

Dann folgt der MIME Type und schliesslich die eigentliche Meldung

4) *Abschluss der Verbindung*
Entweder der Client oder der Server beenden die Verbindung.
Da es sich bei HTTP um ein zustandsloses Protokoll handelt, geht der Zustand der jeweiligen Kommunikation zwischen zwei Anfragen verloren: nach der Bearbeitung eines GET Befehls vergisst der Server den Client. Es liegt also am Programmierer die Geschichte zu schreiben, oder man setzt Servlets oder andere Techniken ein.

Die bekanntesten Statuscodes sind 200 (alles okay) und 404 (not found). Beim obigen Link bei W3C finden Sie eine aktuelle Liste der Codes.

Neben HTTP gibt es verschiedene Varianten, die für spezielle Zwecke geschaffen wurden. Ein Beispiel ist HTTPS, welches speziell eine sichere Verbindung liefern soll. In diesem Protokoll werden die Kommunikationsnachrichten zwischen dem Client und dem Server verschlüsselt. Unter HTTP-NG wurde versucht, eine "new Generation" des HTTP zu schaffen. Aber diese Bemühungen sind nicht sehr erfolgreich. Sie starteten 1997 und man hört eigentlich nicht mehr viel davon. Auch hier ist die Standardreferenz der W3C Site.

3.5. MIME

MIME ist ein offenen Standard zum Senden von Multipart, Multimedia Dokumenten mittels Internet email. Die Daten können binär oder Text sein, also ASCII oder nicht-ASCII. Ursprünglich wurde MIME für email Verkehr definiert. Aber das Konzept hat sich so gut bewährt, dass heute alle Browser und viele Applikationen dieses Konzept aufgegriffen haben.

MIME unterstützt ungefähr hundert vordefinierte Typen. Alle MIME Typen werden mittels zweier Angaben spezifiziert: Typ und Subtyp. Der Typ legt grob fest: ist es ein Bild, Text oder ein Film...? Der Subtyp legt das konkrete Format fest: gif, jpeg, ...
Sie finden eine Zusammenstellung der MIME Typen in der RFC Spezifikation auf dem W3C Site http://www.w3.org/Protocols/rfc1341/0_TableOfContents.html .

In Ihrem Browser können Sie eigene MIME Typen definieren und diesen neuen Typen auch Applikationen zuordnen.

3.6. CGI

CGI, das Common Gateway Interface, wird in der Regel benutzt, um Webseiten dynamisch zu erweitern. Der Browser startet aus einer Formularseite auf dem Server ein CGI Programm, häufig in Perl geschrieben. Dieses Programm hat freien Zugriff auf den Server oder einen dedizierten Bereich. Dadurch wird es möglich beispielsweise Datenbankanbindungen zu realisieren. Zum Thema CGI finden Sie viele Bücher. CGI Skripts sind ein einfacher Erweiterungsmechanismus. Allerdings haben CGI Skripts den Nachteil, dass sie viele Ressourcen benötigen. Beispielsweise wird bei jedem CGI Aufruf eine eigene Shell generiert. Das kann zu Systemengpässen auf dem Server führen. Wir werden unterschiedliche Techniken kennen lernen CGI Skripte zu umgehen. Diese reichen von einfachen Applets bis zu RMI (Remote Methode Invocation) oder CORBA (Common Object Request Broker) Lösungen. Java bietet eine Vielzahl von Lösungen an.

3.7. Applets und Security

Nachdem wir gelernt haben wie mit Hilfe von GET und PUT Dateien hin und her geschoben werden, wollen wir auch noch genauer anschauen, wie dies mittels Applets geschieht. Dabei muss beachtet werden, dass das Java Sicherheitsmodell sich wesentlich geändert hat seit Java 2. Mit Hilfe des Security Tools lassen sich individuelle Sicherheitslevel und Policies definieren, so dass man neu auch aus Applets auf das lokale Dateisystem schreiben kann.

3.7.1. Wo kommen die Applets und die Class Dateien her?

Wann immer ein Web Browser ein Applet Tag sieht und das Applet herunterlädt, wird eine ganze Ereigniskette ausgelöst:

1. der Browser reserviert die im Applet Tag vordefinierte Fläche. In den meisten Browsern kann diese Fläche dann nicht mehr verändert werden. Sun's Appletviewer ist in dieser Beziehung flexibler.
2. der Web Browser öffnet eine Verbindung zum Server, der normalerweise als CODEBASE Parameter angegeben wird. Falls die CODEBASE fehlt, wird einfach angenommen, dass die Class Datei im selben Verzeichnis wie die HTML Seite ist. Standardmässig wird hier der Port 80 als Kommunikationsport angenommen.
3. der Browser verlangt das *.class File vom Server genauso wie andere Dateien verlangt werden können. Es wird also ein GET Befehl abgesetzt.
4. der Server antwortet mit einem MIME Header und einer Leerzeile gefolgt von den binären Daten der Class Datei. In der Regel sollte der MIME Type application/octet-stream sein. Einige Server senden die binäre Datei aber als Text.
5. der Web Browser empfängt die Class Datei und speichert alle Daten in einem Byte Array.
6. der Byte Code Verifier des Browsers verifiziert den Code des Applets und stellt sicher, dass nichts illegales transferiert wurde. In diesem Punkt versagte beispielsweise Netscape: man konnte einen Mini-Webserver als Applet herunter laden und hatte dann Zugriff auf das Dateisystem, mit dem Webserver Root. Dieses Sicherheitsloch in Netscape 4.x ist in den neueren Versionen behoben.
7. nachdem der Bytecode verifiziert ist, wird er zu einem Class File umgewandelt. Dies geschieht mittels einer defineClass() Methode. Mit der Methode loadClass() wird die Klasse geladen. Damit keine Namenskonflikte entstehen, wird vor den Klassennamen die download Adresse hinzugefügt.
8. nun muss die Klasse gestartet werden. Normalerweise sucht der Class Loader zuerst im Classpath nach den erforderlichen Klassen. Falls der Browser die Klasse dort nicht findet, versucht er es erneut in der CODEBASE. Falls auch dort nichts gefunden wurde bleibt nur noch das werfen einer Exception.

3.7.2. Security: mit wem kann ein Applet kommunizieren und was?

In den ersten Versionen wurde eine sogenannte Sandbox definiert, mit deren Hilfe die Security eindeutig definiert war. Alle Applets hatten die Möglichkeit aus der Codebase Daten zu transferieren, auch zum Teil aus relativ adressierten Verzeichnissen. Jedoch war es völlig unmöglich, dass ein Applet auf das lokale Dateisystem zugreifen konnte. Das wurde in Java 2 geändert.

Grundsätzlich gelten aber immer noch die selben Regeln:

1. Applets können nicht auf beliebige Speicheradressen im Hauptspeicher zugreifen. Dies ist eine Eigenschaft von Java, nicht des SecurityManagers des Browsers.

2. Applets können nicht auf das lokale Dateisystem zugreifen. Es sind somit alle Lese- und Schreibzugriffe verboten. Auch schon die Information, ob eine Datei existiert oder nicht, kann nicht abgefragt werden.
3. Applets können keine anderen Programme starten, also im Speziellen System.exec() oder Runtime.exec().
4. Applets können keine native Methoden aufrufen oder Bibliotheken laden.
5. Applets können keine Systemeigenschaften bestimmen: System.getProperties() ist nicht gestattet. Damit kann auch kein Home Verzeichnis, Benutzername oder ähnliches bestimmt werden.
6. Applets können auch keine Systemeigenschaften definieren.
7. Ab Version 1.1 können Applets keine Threads oder Threadgruppen manipulieren.
8. Applets können keine eigenen ClassLoader, SecurityManager, ContentHandlerFactories, SocketImplFactories oder URLStreamHandlerFactories definieren.
9. Applets können nur Verbindungen zu dem Host eröffnen, von dem sie herunter geladen wurden.
10. Applets können keine Clientverbindungen empfangen, also nicht als Server auftreten. Hier schlampte Netscape.

Einige der Einschränkungen sind gravierend, einige sind zwingend. Es gibt verschiedene Mechanismen, um einige der Einschränkungen zu umgehen, zum Beispiel das Java Plugin oder die Sicherheitspolicies. Diese können aber auch nur nach expliziter Bewilligung durch den Benutzer auf dem Client installiert und aktiviert werden.