Einfacher Webserver und Webbrowser

Aufgabenstellung

Implementieren sie einen einfachen Webserver, welcher auf Clientanfragen (Webbrowser),wie in der unten angefügten Facharbeit, reagieren kann.

Im Prinzip handelt es sich dabei um ein Programm welches als Hintergrundprozess (Dämon) abläuft und auf einen Socket (Port 80 oder 8080) den Datenverkehr empfängt und versendet. Die Textanfragen die über diesen Port ankommen werden ausgewertet und dementsprechend reagiert der Server mit Antworten.

Um die vom Webserver gesendeten HTML Dokumente darstellen (browsen) zu können, entwickeln sie einen einfachen Webbrowser der alle unten angeführten Spezifikationen erfüllt.

Im Zeitrahmen von 36 Stunden in der Schule sind beide Programme, Webserver und einfacher Webrowser funktionsfähig zu präsentieren, um eine positive Note zu erhalten.

Bis nächste Woche ist ein ausführliches Pflichtenheft und das Grundkonstrukt der Programme vorzulegen.

Wie funktioniert das HTTP-Protokoll?

1. Einleitung

Beinahe jeder Mensch ist täglich auf das HTTP-Protokoll (kurz für „Hypertext Transfer Protocol“) angewiesen, doch nur die Wenigsten wissen überhaupt von der Existenz dieses Protokolls. Jedoch basiert genau auf diesem (unter anderem) der Datenverkehr im Internet zwischen Webserver und Browser des Anwenders. Aber auch andere Netzwerkanwendungen nutzen dieses Protokoll zur einheitlichen Übertragung von Daten auf Basis des TCP/IP[1] Protokollstacks[2].
Daher stellt sich nun die Frage, wie ein solch häufig verwendetes Protokoll funktioniert: Auf welcher Basis verläuft die Kommunikation? Was müssen Server und Client beachten? Wie müssen ihre Anfragen und Antworten aussehen? Und wie genau werden die Daten schließlich versendet?
Letztendlich klärt sich durch die Beantwortung dieser einzelnen Fragen sogar die häufig gestellte Frage nach der Funktionsweise des Internets.
Zum Abschluss dieser Facharbeit wurde ein einfacher Webserver in Java implementiert, der grundlegende Anforderungen des HTTP-Protokolls erfüllt und somit Webseiten und Dateien an den Browser ausliefern kann. In Bezug auf diesen geht es auch kurz um das Einlesen von Dateien und die Programmierung von Netzwerkanwendungen in Java.

1.1 Was ist ein Protokoll?

Bevor man in die Thematik des HTTP-Protokolls einsteigt, sollte geklärt werden, was ein Protokoll überhaupt ist und welche Funktionen es erfüllen soll.
Grundlegend kann man Protokolle mit Sprachen vergleichen: Sowohl Server, wie auch Client, müssen die gleiche “Sprache” sprechen, damit ein (Daten-)Austausch stattfinden kann. Nur wenn beide Teilnehmer die Regeln dieser “Sprache” (→ des Protokolls) einhalten, kann die Kommunikation gelingen.[3]
Ein Beispiel: Computer A sendet eine Nachricht an Computer B. Aufgrund von Unterschieden, wie zum Beispiel einem anderen Betriebssystem, kann Computer B die Nachricht jedoch nicht verstehen. Hier kommt nun ein Protokoll ins Spiel, nach welchem die Nachricht von Computer A “übersetzt” wird, damit Computer B schließlich die “Rück-Übersetzung” ermöglicht wird, so dass die Kommunikation erfolgreich stattfinden kann.

2. Das HTTP-Protokoll

2.1 Anwendungsbereiche

Bereits seit 1990 wird das Anwendungsprotokoll HTTP intensiv im World Wide Web – dem Internet – genutzt, und auch heute findet das HTTP-Protokoll in den verschiedensten Bereichen Anwendung. Wie in der Einleitung bereits erwähnt, vor allem im Internet, in lokalen Netzwerken oder in Intranets[4]: Webseiten, bzw. Text-Inhalte (wie die Seite selbst, CSS[5] JavaScript[6], …), Grafiken, sowie andere Inhalte werden über dieses Protokoll

zwischen dem Server und dem Browser, bzw. der jeweiligen Clientsoftware des Anwenders übertragen.
Als Übertragungsprotokoll kann HTTP aber auch für praktisch jede Datenübertragung in Netzwerken und dem Internet genutzt werden und ist somit vielseitig anwendbar.[7]
Gerade diese Vielseitigkeit und die Einfachheit in der Anwendung sorgen für die weite Verbreitung des Protokolls.

2.2 Grundlegende Informationen

Das HTTP-Protokoll der Version 1.0 funktioniert nach dem Client-Server-Prinzip: Der Client (meist ein Browser) sendet eine Anfrage ("Request") an den HTTP-Server, der daraufhin seine Antwort ("Response") zurücksendet und die Verbindung schließt.
Diese Kommunikation verläuft auf Basis von Meldungen im Text-Format, die über TCP[8] auf Port 80 versendet werden.[9] Seit Version 1.1 kann (unter anderem) eine vorhandene TCP-Verbindung auch für mehrere Anfragen genutzt werden, damit der Anwender nicht zu lange auf die gewünschten Daten warten muss.
Dazu ein Beispiel: Wenn in eine Internetseite 5 Grafiken eingebettet sind, mussten mit dem HTTP-Protokoll 1.0 noch 6 Verbindungen zum Abruf aller Daten aufgebaut werden, was zu einer längeren Ladezeit der gesamten Seite geführt hat. Durch die Änderung in Version 1.1 reicht eine einzige TCP-Verbindung dazu aus (vorausgesetzt, alle Daten werden von dem gleichen Server angefordert).
Jede dieser Text-Meldungen ist in zwei "Teile" unterteilt: dem so genannten "Header" und den Daten. Der Header enthält dabei wichtige Informationen für den Empfänger der Meldung[10], während die Daten den eigentlichen Inhalt der Seite, bzw. der Mitteilung enthalten.
Die Teilung zwischen Header und Daten geschieht durch eine Leerzeile, während Informationen innerhalb des Headers durch ein Leerzeichen (in der ersten Zeile) oder einen Doppelpunkt mit anschließendem Leerzeichen (in allen weiteren Zeilen) getrennt werden.

2.2.1 Was ist eine URL?

Um überhaupt an Daten gelangen zu können, muss der Client (a) den Server "finden" und ihm (b) auch mitteilen, welche Daten er haben möchte. Zu diesem Zwecke gibt es URL[11]s, die beide Funktionen erfüllen und zusätzlich das zu verwendende Protokoll angeben. Üblicherweise sieht eine URL im Webbrowser folgendermaßen aus:
`http://www.seite.de:80/pfad/datei.html`
Mit `http://` wird das Protokoll angegeben, `www` ist eine beliebige Subdomain, `seite` ist der Domainname, `.de` die TLD[12] und `:80` der TCP-Port[13]. Die folgenden, mit einem Slash (Schrägstrich) getrennten, Informationen geben nun den Pfad und die Datei an.[14] Sollte eine Datei- oder Pfad-Angabe fehlen, sendet der Server die Standard-Datei oder eine Fehlermeldung.

2.2.2 Schichtenmodelle

Der Ablauf dieser zuvor in 2.2 beschriebenen Kommunikation lässt sich anhand von verschiedenen Schichtenmodellen näher und vor allem technischer erläutern. Eines der bekanntesten Modelle ist das so genannte **OSI-Schichtenmodell**.
Dieses wurde von der ISO "als Grundlage für die Bildung von Kommunikationsstandards entworfen und standardisiert"[15] und besteht aus 7 Schichten, wovon jede bei der Kommunikation eine bestimmte Aufgabe zu erfüllen hat. Jeder Datenstrom muss diese 7 Schichten zweimal durchlaufen: Einmal beim Absender und einmal beim Empfänger.
Mit einem Fokus auf dem HTTP-Protokoll sieht das Modell folgendermaßen aus:[16]

Schicht	Aufgabe	Beispiel(e)
7	Anwendung	Internet
6	Darstellung	HTTP
5	Kommunikation	HTTP
4	Transport	Transmission Control Protocol (TCP)
3	Vermittlung	Internet Protocol (IP)
2	Sicherheit	Ethernet
1	Übertragung	Kabel, drahtlose Verbindung, Licht

Das **TCP/IP-Referenzmodell** basiert auf dem OSI-Schichtenmodell, beschreibt die Kommunikation über lokale Netze hinaus jedoch genauer als Letzteres:

Es geht von nur vier aufeinander aufbauenden Schichten aus und zeigt, dass TCP und IP die Grundlage für die Kommunikation via HTTP (über lokale Netzwerke hinaus) ist, da sie schließlich für das Ankommen der Daten-Pakete verantwortlich sind. Die ersten beiden Schichten werden zusammengefasst und die fünfte und sechste Schicht entfallen komplett, sodass das Modell nun so aussieht:

OSI-Schicht(en)	Schicht	Aufgabe	Beispiel(e)
7	4	Anwendung	Internet: HTTP, FTP, …
4	3	Transport	Transmission Control Protocol (TCP)
3	2	Netzwerk	Internet Protocol (IP)
1 + 2	1	Netzzugang	Ethernet, Kabel, drahtlose Verbindung, Licht

Immer noch sorgt die erste Schicht für den physischen Zugang zum Netzwerk, während IP (2. Schicht) für die Weitervermittlung, bzw. Adressierung von Paketen zuständig ist und TCP (3. Schicht) für eine zuverlässige (nicht unbedingt sichere!) Ende-zu-Ende Verbindung zwischen den beiden Netzwerkteilnehmern sorgt. Auch hier stellt die letzte Schicht den Anwendungsfall (meist das Internet und eine Vielzahl an Protokollen) dar.[17]
Somit wird klar, warum Eingangs von einem "Protokollstapel" und einem "Anwendungsprotokoll" gesprochen wurde: HTTP "liegt" oben auf einem Stapel verschiedener untergeordneter Protokolle in der Anwendungs-Schicht und ist für die eigentliche Datenverarbeitung bei den beiden Netzwerkteilnehmern verantwortlich.

2.3 "Request" des Clients an den Server

Damit ein Client (im Folgenden wird von einem typischen Webbrowser ausgegangen) Daten von einem Server empfangen kann, muss dieser zuerst eine TCP-Verbindung zum Server herstellen und anschließend seine Anfrage ("Request") formulieren.

2.3.1 Aufbau eines Request-Headers

Eine Anfrage ist eine einfach aufgebaute Text-Meldung, die mehrere Informationen für den Server enthält, welche am folgenden (gekürzten) Beispiel deutlich werden. Der Anwender hat dabei mit dem Google Chrome Webbrowser die URL http://blog.marvin-menzerath.de/artikel/einfuehrung-versionsverwaltung-git-github/ aufgerufen und damit den Versand der folgenden Nachricht an den Server ausgelöst:

```
GET /artikel/einfuehrung-versionsverwaltung-git-github/ HTTP/1.1 Host: blog.marvin-menzerath.de Connection: keep-alive Accept: text/html User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) Chrome/33.0.1750.117
```

Die wichtigste Zeile ist die erste, die sogenannte Request-Zeile, welche dem Server angibt, (a) welche Methode der Client verwendet (sprich: was der Client als Antwort erwartet), (b) welche Datei / welchen Pfad der Client anfordert und © welche Protokollversion der Client in seiner Anfrage verwendet.
Die zweite Zeile benötigt der Server, damit er (falls er mehrere Webseiten unter verschiedenen Domains beherbergt) die richtigen Daten an den Client senden kann.
Mit der dritten Zeile kann der Client den Server anweisen, die Verbindung nach der Datenübertragung offen zu halten, um weitere Daten zu übertragen.[18]
Durch die vierte Zeile weiß der Server, welche Daten der Client verarbeiten kann; in diesem Falle sind es HTML-, bzw. Text-Daten. Die letzte Zeile kann der Server verwenden, um eine auf den Browser oder das Betriebssystem zugeschnittene Seite an den Client zu versenden. Dies wird meist bei mobilen Geräten genutzt, die eine Version der Seite mit weniger Grafiken erhalten, damit die Seite schneller geladen werden kann.

2.3.2 Verschiedene Methoden

Im oben verwendeten Beispiel verwendet der Client die **GET**-Methode. Diese wird in beinahe 98% aller Fälle verwendet und dient dazu, Daten vom Server anzufordern. Aufgrund dieser Aufgabe müssen keine weiteren Daten neben dem Header übermittelt werden.
Um nun beispielsweise Daten eines Formulars an den Server zu übermitteln, wird meist die **POST**-Methode verwendet. Der Unterschied zur GET-Methode ist, dass nach dem Header die Daten aus dem Formular (mit einer Leerzeile getrennt) folgen. Natürlich muss die Request-Zeile im Header nun auch statt

```
GET /formular.php HTTP/1.1 POST /formular.php HTTP/1.1
```

lauten, damit der Server die Daten richtig verwenden, bzw. für einen PHP-Interpreter[19] verfügbar machen kann.
Diese Daten werden dann im folgenden Format aufgelistet:

```
vorname=Marvin&nachname=Menzerath
```

Weitere, aber eher selten genutzte Methoden sind **HEAD** (fordert nur den Response-Header an), **PUT** (erstellt/ändert eine Datei auf dem Server; meist deaktiviert), **OPTIONS** (fordert alle möglichen/erlaubten Methoden an) , **DELETE** (löscht eine Datei auf dem Server; meist deaktiviert), **TRACE** (zur Verfolgung von

Requests, die über einen Proxy-Server laufen) und **CONNECT** (nur für Proxy-Server; baut einen Tunnel zur Kommunikation zwischen Client und Server zum angegebenen Ziel auf).[20]

2.4 "Response" des Servers an den Client

Nachdem der Server nun die Anfrage des Clients empfangen und verarbeitet hat, muss er eine Antwort, die so genannte „Response" über die gleiche TCP-Verbindung an den Client senden, damit dieser schließlich die Webseite anzeigen, bzw. weitere (in der Antwort des Servers referenzierte) Inhalte anfordern kann.

2.4.1 Aufbau einer Response

Die Response ist (ebenfalls wie der Request) eine Text-Meldung und genauso in Header und Daten unterteilt. Während der Header hier Daten für den Client enthält (wie Statuscode, Größe des Inhalts, Server-Name/-Zeit, …), ist der eigentliche Seiteninhalt erst in den Daten, ebenfalls in einfacher Text-Form, enthalten.
In diesem (gekürzten) Beispiel erhält der Browser nun die Antwort auf die in 2.3.1 formulierte Anfrage:

```
HTTP/1.1 200 OK Date: Fri, 28 Feb 2014 14:11:09 GMT Server: Apache/2.2.22 (Debian) Content-Length: 9660 Connection: Keep-Alive Content-Type: text/html; charset=UTF-8  [...]
```

In der ersten Zeile der Antwort gibt der Server nun zuerst das verwendete Protokoll samt Version, sowie den Statuscode[21] als Zahlenwert und als Text-Ausdruck aus. In diesem Fall wurde die Datei gefunden und ist somit im Daten-Teil der Antwort vorzufinden.
Die nächsten beiden Zeilen geben nun (a) die Server-Zeit und (b) den verwendeten Webserver samt Version und Betriebssystem an. In diesem Fall wird einer der bekanntesten Webserver, Apache in der Version 2.2.22, auf einer Debian-Linux-Basis verwendet.
Die vierte Zeile gibt dem Client die Länge, bzw. Größe der Daten (in Bytes) an, die er nun zu erwarten hat.
Mit der fünften Zeile bestätigt der Server dem Client, dass er die genutzte TCP-Verbindung für weitere Anfragen des Clients offen lässt.
Die letzte Zeile des Header-Teils spezifiziert nun den Content-Type[22], welcher dem Client angibt, welchen Dateityp die übertragenen Daten haben (hier: Text, bzw. HTML) und in welcher Kodierung diese vorliegen (hier: UTF-8[23]).
Sollte der Client nun aber statt einer HTML-Seite oder einem anderen Text-Inhalt einen Inhalt anfordern, der normalerweise nicht in Text-Form übertragen wird (wie z.B. eine Grafik im PNG-Format), ändert sich der angegebene Content-Type von `text/html` auf `image/png`, wobei die Grafik selbst aber weiterhin binär im Daten-Teil der Response übertragen wird.
Die siebte Zeile bleibt nun als Abtrennung zwischen dem Header und den Daten frei und schließlich folgt der Inhalt der Übertragung in den weiteren Zeilen.

2.4.2 Statuscodes und ihre Bedeutung

Die Statuscodes sind dreistellige Nummern, die in fünf grobe Gruppen (so genannte Klassen) unterteilt sind und dem Client Informationen über die Verfügbarkeit der Daten geben.
Da es weit über 60 verschiedene HTTP-Statuscodes gibt, die jedoch nicht alle von der IETF[14] festgelegt worden sind, wird im Folgenden nur auf die wichtigsten und meist genutzten Statuscodes und Gruppen eingegangen.[24]
Kurz zum Aufbau eines Statuscodes: Die erste Ziffer gibt die Statusklasse an; also beispielsweise, ob die Anfrage erfolgreich war oder nicht, während die beiden nachfolgenden Zahlen (normalerweise) von 0 an aufwärts zählen und den jeweiligen Fehler, bzw. das Problem genauer spezifizieren.
Die Statuscodes von 100 bis 199 gibt es seit der HTTP-Version 1.1 und geben dem Client Informationen wie `100 Continue` (Anfrage ist noch nicht fertig formuliert und soll weitergeführt werden) oder `101 Switching Protocols` (Protokoll wechseln). Generell werden diese Statuscodes eher selten vom Server versendet.
Statuscodes von 200 bis 299 informieren den Client über eine erfolgreiche Anfrage mit Statuscodes wie `200 OK` (alles in Ordnung), `204 No Content` (Keine Daten) oder `206 Partial Content` (Teil der Daten versendet; Client muss hier explizit mehrere Teile anfordern).
Mit Statuscodes im Bereich von 300 bis 399 weist der Server den Client auf eine Umleitung hin. Meist werden Statuscodes wie `301 Moved Permanently` (Seite dauerhaft verschoben), `302 Moved Temporarily` (Seite kurzzeitig verschoben) oder `304 Not Modified` (Seite seit letztem Abruf nicht verändert) versendet, damit der Client die korrekten Daten anfragen, bzw. anzeigen kann.
Statuscodes von 400 bis 499 sind Fehlermeldungen, die der Client ausgelöst hat, da der Server die Anfrage nicht beantworten kann. Hier werden vor allem die Statuscodes `400 Bad Request` (falsch formulierte Anfrage), `403 Forbidden` (Zugriff verweigert), `404 Not Found` (nicht gefunden) oder `408 Method Not Allowed` (vom Client verwendete Methode ist nicht erlaubt; siehe 2.3.2) versendet.
Der letzte Bereich von 500 bis 599 enthält schließlich Fehlermeldungen, die vom Server verursacht wurden: `500 Internal Server Error` (interner Server-Fehler), `501 Not Implemented` (Methode nicht implementiert) und `503 Service Unavailable` (Server meist überlastet) kann der Client (normalerweise) nicht verursachen.[25]

4. Fazit

Zusammenfassend lässt sich nun feststellen, dass das HTTP-Protokoll vielseitig und vor allem einfach einsetzbar ist. Das gesamte Internet basiert auf diesem Protokoll und schon mit wenigen Zeilen einfachen Textes lassen sich Server-Client-Anwendungen auf Netzwerkbasis erstellen.
Dabei sollte - gerade im Angesicht der aktuellen Spionage-Skandale (Stichwort NSA) - auch die Frage nach der Sicherheit des Protokolls aufkommen. Dadurch, dass alle Meldungen und Daten unverschlüsselt übertragen werden, ist es für Angreifer sehr leicht, diese Daten abzufangen und/oder zu manipulieren. Um diese Angriffe zu erschweren sollte daher (wann immer es möglich ist) eine SSL-verschlüsselte Verbindung genutzt werden.
Aktuell wird außerdem bei den größten Technikkonzernen weltweit an der Version 2.0 des Protokolls gearbeitet, die einige Neuerungen und Verbesserungen (wie eine dauerhafte Verschlüsselung der Datenübertragung) für das Internet bereithalten soll. Allerdings wurde sich hier noch nicht auf konkrete Umsetzungen geeinigt.

5. Literaturverzeichnis

- Bauer, Martin: TCP/IP-Referenzmodell http://www.uni-protokolle.de/Lexikon/TCP/IP-Referenzmodell.html, 28.02.2014
- Schnabel, Patrick: ISO/OSI-7-Schichtenmodell http://www.elektronik-kompendium.de/sites/kom/0301201.htm, 28.02.2014
- Schnabel, Patrick: HTTP - Hypertext Transfer Protocol http://www.elektronik-kompendium.de/sites/net/0902231.htm, 17.12.2013
- The Internet Society (Hrsg.): Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616). 1999 http://www.ietf.org/rfc/rfc2616.txt, 17.12.2013
- Wolf, Jürgen: C von A bis Z - 23.8 Das HTTP-Protokoll http://openbook.galileocomputing.de/c_von_a_bis_z/023_c_cgi_008.htm, 17.12.2013

---

1. Transmission Control Protocol / Internet Protocol; dazu später in 2.2.2 mehr [return]
2. Zu Deutsch: Protokollstapel; dazu später in 2.2.2 mehr [return]
3. Vgl. Wolf, Jürgen: C von A bis Z - 23.8 Das HTTP-Protokoll, Abschnitt 23.8.2 / 23.8.3 [return]
4. begrenzt großes und nicht öffentliches Rechnernetzwerk [return]
5. Cascading Style Sheets: Stilvorlage, die das Aussehen einer Webseite bestimmt [return]
6. Skriptsprache, die Inhalte in Webseiten dynamisch verändern kann [return]
7. “HTTP allows basic […] access to resources available from diverse applications”, The Internet Society (Hrsg.): Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616), 1999, S. 8 [return]
8. Transmission Control Protocol: Definiert, wie Computer innerhalb eines Netzes Daten austauschen [return]
9. Vgl. Schnabel, Patrick: HTTP - Hypertext Transfer Protocol, Abschnitt “Wie funktioniert HTTP?” [return]
10. siehe Abschnitt 2.3.1 und 2.4.1 [return]
11. Uniform Resource Locator [return]
12. Top Level Domain [return]
13. 80 ist der Standard-Port, daher ist diese Angabe nur bei Abweichung davon nötig [return]
14. Vgl. Wolf, Jürgen: C von A bis Z - 23.8 Das HTTP-Protokoll, Abschnitt 23.8.6 [return]
15. Schnabel, Patrick: ISO/OSI-7-Schichtenmodell; http://www.elektronik-kompendium.de/sites/kom/0301201.htm, 28.02.2014 [return]
16. Vgl. Schnabel, Patrick: ISO/OSI-7-Schichtenmodell [return]
17. Vgl. Bauer, Martin: TCP/IP-Referenzmodell [return]
18. Eine Änderung in HTTP 1.1; siehe Abschnitt 2.2 [return]

19. PHP ist eine Server-seitige Programmiersprache für Webseiten, die zur Laufzeit interpretiert und ausgeführt wird
[return]
20. Vgl. Schnabel, Patrick: HTTP - Hypertext Transfer Protocol, Abschnitt “HTTP-Methoden”
[return]
21. Siehe Abschnitt 2.4.2
[return]
22. Auch “Internet Media Type” (MIME-Type) genannt: gibt den Datentyp der im Daten-Teil der Nachricht versendeten Daten an (z.B.: Text, Bild, Video, …)
[return]
23. 8-Bit Universal Character Set Transformation Format: eine oft genutzte Zeichenkodierung
[return]
24. Auswahl basiert unter anderem auf den 12 am häufigsten durch den Webserver versendeten Status-Codes im Dezember 2013 auf https://blog.marvin-menzerath.de.
[return]
25. Vgl. Schnabel, Patrick: HTTP - Hypertext Transfer Protocol, Abschnitt “HTTP-Response-Codes / HTTP-Status-Codes”
[return]

Implementation eines einfachen Webservers in Java

12. May 2014 / 03. October 2016 Facharbeit, HTTP-Protokoll, Java, Webserver, Programmierung

In diesem zweiten Teil meiner Facharbeit geht es nun um die Implementation, bzw. Entwicklung eines einfachen Webservers in Java. Dieser lässt sich bereits mit dem Wissen über das HTTP-Protokoll und Grundlagen der Netzwerkprogrammierung in Java entwickeln. Bevor Sie diesen Artikel weiterlesen, sollten Sie den ersten Teil meiner Facharbeit über die Funktionsweise des HTTP-Protokolls gelesen haben.

3. Implementation eines einfachen Webservers in Java

Mit diesem Wissen über das HTTP-Protokoll und über die Programmierung von Netzwerkanwendung in der Programmiersprache Java[1] lässt sich nun ein einfacher Webserver in Java implementieren, der auf GET- und POST-Requests eines Browsers mit den Daten aus einem festgelegten Verzeichnis antworten kann und auf Fehler passend reagiert. Der gesamte Quellcode der Anwendung ist unter https://gist.github.com/MarvinMenzerath/16dbf192de45e11eab48 vorzufinden. Der Aufruf dieser URL wird für eine verbesserte Lesbarkeit mit Zeilennummern und Syntax Highlighting[2] empfohlen.

3.1 Klassenstruktur

Das gesamte Projekt besteht aus 2 Packages mit jeweils 3 Klassen. Der Aufbau dieser Struktur sieht dabei folgendermaßen aus:

1	2	3
de.menzerath.	httpserver.	HTTPServer
		HTTPThread
		WebResources
	util.	FileManager
		Logger
		ServerHelper

Der Einstiegspunkt der Anwendung liegt in der Klasse HTTPServer und dort (wie üblich) in der `main()`-Methode:

```
public static void main(String[] args) {      new HTTPServer(8080, new File("./www"), true, new File("log.txt")); }
```

Dort wird nun ein neues Objekt der Klasse `HTTPServer` erstellt, welches wiederum einen ServerSocket für den angegebenen Port erstellt und auf diesem auf eingehende Verbindungen wartet, die dann jeweils an ein neues Objekt der Klasse `HTTPThread` übergeben werden. Letztere behandelt dann die eingehende Verbindung und den HTTP-Request entsprechend in einem eigenen Thread[3] und sendet die HTTP-Response zurück an den anfragenden Client.[4]

Die Klasse `WebResources` beinhaltet das HTML-Grundgerüst sowie das CSS für Dateiauflistungen und die

Fehlerseiten.
In der Klasse `FileManager` finden sich passende Content-Types zu verschiedenen ausgewählten Dateiendungen, die im HTTP-Header gesendet werden müssen. Neben zwei weiteren, kleinen Hilfsmethoden, kann die Methode `getReadableFileSize(long size)` auch die Dateigröße (die durch `File.length()` ausgegeben wird) in eine lesbare Dateigröße (wie "2.21 MB") umwandeln.
Durch die statischen Methoden der Klasse `Logger` werden Meldungen des Webservers mit einem Zeitstempel versehen, auf der Konsole ausgegeben und in einer festgelegten Datei angezeigt.
Zu guter Letzt existieren noch zwei Hilfsmethoden in der Klasse `ServerHelper`, die die IP-Adresse des Servers im Netzwerk, sowie den absoluten Pfad zum Datenverzeichnis zurückgeben.

3.2 Verarbeitung des Requests

Die Verarbeitung des HTTP-Requests geschieht hauptsächlich im zur jeweiligen TCP-Verbindung gehörenden Thread[5]. Bevor der Request aber überhaupt geparst[6] und somit verarbeitet werden kann, muss der Server erst einmal den Request vom Client empfangen. Dazu wird ein BufferedReader verwendet, der wiederum die Daten von einem InputStreamReader erhält, der den InputStream - also die eingehenden Daten vom Client - verwendet. Außerdem wird ein Socket-Timeout von 30 Sekunden festgelegt, sodass der Client die Verbindung nicht zu lange offen halten kann.
Nun wird jede Zeile des Requests (insofern diese nicht „leer" ist) in eine ArrayList eingetragen, damit die Anfrage schließlich im weiteren Verlauf Zeile für Zeile bearbeitet werden kann.

3.2.1 Parsen des Requests

Das eigentliche Parsen des Requests geschieht in wenigen Schritten: Insofern die erste Zeile des Requests mit `HTTP/1.0` oder `HTTP/1.1` endet – also das HTTP-Protokoll verwendet wird – und diese Zeile auch mit `GET` oder `POST` beginnt (andere Methoden werden hier mit einem 501er [Fehler] beantwortet), wird der Vorgang fortgesetzt. Andernfalls wird entweder ein 400er (Bad Request) oder ein 501er (Not Implemented) an den Client versendet.
Nun muss die angeforderte Datei gefunden werden: Dazu werden einfach alle Informationen aus der ersten Zeile der Anfrage (bis auf die Pfad-/Dateiangabe) entfernt. Sollte ein GET-Request mit einer URL mit Parametern ankommen, werden letztere ebenfalls entfernt.
Nachdem nun also der relative Pfad zur angeforderten Datei bekannt ist, benötigt der Server den absoluten Pfad zu dieser. Dazu (und zur weiteren Verarbeitung des Requests) eignet sich die Klasse `File`, die Java bereits mitbringt. Dem Konstruktor wird der festgelegte WebRoot[7], sowie der Pfad aus der Anfrage übergeben und anschließend wird die Methode `getCanonicalFile()` an diesem Objekt aufgerufen, sodass das Objekt schließlich (mit einem absoluten Pfad) auf eine bestimmte Datei oder ein Verzeichnis zeigt[8].
Der letzte Schritt (vor der eigentlichen Vorbereitung der Response) prüft, ob (insofern das gerade erzeugte `File`-Objekt ein Verzeichnis ist), eine Index-Datei im Verzeichnis vorhanden ist. Diese muss `index.html` heißen und wird (falls vorhanden) statt einer Dateiauflistung an den Client gesendet.

3.2.2 Vorbereitung der Response

Der Inhalt der Response ist nun von weiteren Faktoren abhängig:
Ist die angeforderte Datei innerhalb des WebRoots? Existiert sie überhaupt?
Je nach Antwort auf diese Überprüfungen muss die entsprechende Fehlermeldung gesendet werden.
Nun gibt es nur noch zwei Fälle, in denen die Applikation eine Response wirklich vorbereiten muss: Wenn **ein Verzeichnis** oder **eine Datei** angefordert wird. Bei ersterem wird eine alphabetisch sortierte Auflistung aller Dateien und Unter-Verzeichnisse im angeforderten Verzeichnis erstellt – insofern das Verzeichnis nicht leer ist und der Zugriff erlaubt ist. Auch das gestaltet sich recht einfach, da mit einer `for each`-Schleife durch alle `Files` iteriert wird und sich so auch Dateigröße und das Datum der letzten Bearbeitung einfach auslesen lassen:

```
File[] files = file.listFiles(); for (File myFile : files) { String
fileName = myFile.getName();  String fileSize =
FileManager.getReadableFileSize(myFile.length());    String fileLastMod =
new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(myFile.lastModified()) }
```

Sollte "nur" eine Datei angefordert werden (wie es meistens der Fall ist), muss nicht mal eine wirkliche Ausgabe vorbereitet werden: Es wird nur ein InputStream, der die Datei zur Ausgabe an den Client einliest, benötigt und der Content-Typ der Datei muss anhand der Dateiendung festgestellt werden. Schon können die Daten an den Client versendet werden.

3.2.3 Senden der Daten

Das Senden der Daten besteht aus zwei Schritten: dem Senden des Headers und der eigentlichen Daten, wobei beides natürlich über die gleiche Verbindung und nacheinander erfolgen muss.
Das Senden des Headers übernimmt die Methode `sendHeader(...)`, die mehrere Parameter erwartet: Den zu verwendenden OutputStream, den HTTP-Status-Code (inkl. zugehöriger Nachricht), sowie Typ, Größe und den Zeitpunkt der letzten Bearbeitung der Datei/der Daten.
Nachdem der Header dann versendet wurde, müssen die Daten übertragen werden: Zuerst muss die angeforderte Datei eingelesen und der Content-Type bestimmt werden. Dazu werden ein InputStream und die statische Methode `getContentType(...)` der `FileManager`-Klasse verwendet. Danach wird die Datei Byte für

Byte eingelesen und an den Client übertragen, bis die Übertragung abgeschlossen ist und der OutputStream, also die Verbindung zum Client, beendet werden kann.
Beim Senden einer Fehler-Seite ist zu beachten, dass keine Datei eingelesen werden muss und praktisch "nur" ein String mit der Fehlermeldung gesendet wird. Daher muss an diesem String die Methode `getBytes()` aufgerufen werden.

Literaturverzeichnis

- Ullenboom, Christian: Java ist auch eine Insel. 9. Auflage 2011 http://openbook.galileocomputing.de/javainsel9/, 22.12.2013
- siehe Teil 1 der Facharbeit

---

1. Vgl. Ullenboom, Christian: Java ist auch eine Insel, Abschnitt „21. Netzwerkprogrammierung“ [return]
2. Farblich markierte Codefragmente [return]
3. deutsch: Faden; erlaubt parallel ablaufende Aktivitäten. Hier: Verarbeitung mehrerer Anfragen ohne die gesamte Anwendung zu blockieren. [return]
4. siehe 3.2 [return]
5. gemeint: Thread, der von einem Objekt der Klasse HTTPThread erzeugt, bzw. gestartet wurde [return]
6. Request wird eingelesen und auf Korrektheit überprüft. Außerdem werden die wichtigen Informationen extrahiert und für die weitere Verwendung gespeichert; siehe Abschnitt 3.2.1 [return]
7. Verzeichnis, in welchem die Daten liegen, die vom Server ausgeliefert werden dürfen. Außerhalb (–> "über") diesem Verzeichnis ist kein Zugriff erlaubt. [return]
8. Anzumerken ist, dass diese Datei / dieses Verzeichnis nicht existieren muss [return]