Das Hypertext Transfer Protokoll HTTP/1.1

Michael Dienert

18. Januar 2009

Inhaltsverzeichnis

1 RFC 2616 und RFC 2396 und die Syntaxbeschreibungssprache BNF

1.1 Request For Comments

Dieser Text basiert auf dem Dokument RFC 2616, in dem das Hypertext Transfer Protokoll (HTTP) in der Version 1.1 (**HTTP/1.1**) definiert ist und dem Dokument RFC 2396, welches beschreibt, wie Uniform Resource Identifiers (URI) aufgebaut sind.

RFC steht für Request For Comment, was soviel wie "Aufforderung zur Kommentierung" heisst. Die RFCs sind technische Veröffentlichungen, mit denen das Internet standardisiert und dokumentiert wurde und wird. Wie eine RFC gestaltet werden soll, steht selbst wieder in einer RFC: RFC 2223.

Die Themen, die in RFCs behandelt werden, müssen in Zusammenhang mit dem Internet stehen.

Bevor ein Dokument als RFC veröffentlicht wird, muss es als Vorschlag an die RFC-Redaktion (*RFC-Editor*, eine Gruppe der Internet Society, zu Anfang Jon Postel himself) eingereicht werden.

Die meisten dieser Vorschläge stammen dabei von der IETF (Internet Engineering Task Force) und werden **Internet-Draft** (Draft = Entwurf) genannt. Aber auch Privatpersonen können Vorschläge für eine RFC bei der RFC-Redaktion einreichen. Wie das geht, steht in RFC 4846.

Ein Vorschlag wird also zunächst als Internet-Draft veröffentlicht. Er wird dann von der Internet-Gemeinde geprüft und kommentiert. Hat man ausreichend positive Kommentare und Überarbeitungen gesammelt, kann man einen Antrag auf Veröffentlichung als RFC beim RFC-Editor stellen.

Eine einmal veröffentlichte RFC ist fest und kann nicht mehr geändert werden. Man kann sie bestenfalls durch eine neue RFC ablösen.

Manche RFC schaffen es zum *Internet Standard* (STD) zu werden. Ihren Namen RFC behalten sie dennoch.
Hier eine Übersicht der möglichen RFC-Kategorien (Quelle RFC 2026):

Kategorien im Non-Standards-Track Hier stehen RFCs, die gar nicht zum Standard werden sollen (z.b. Aprilscherze), oder noch nicht ausgereift genug sind.

- **Informational** - zur allgemeinen Information der Internet-Gemeinde. Beispiel: RFC 2468, I remember IANA, ein Nachruf von Vincent Cerf auf Jon Postel (IANA = Internet Assigned Numbers Authority)
- **Experimental** - hiermit kann eine Entwicklungs- oder Forschungsarbeit beschrieben und somit einem grösseren Kreis von Leuten zugänglich gemacht werden.
- **Historic** - wird nicht mehr benutzt

Kategorien im Standards-Track Alles was mal Standard werden möchte, muss die folgenden Stadien durchlaufen:

- **Proposed Standard** - erster Vorschlag für einen Standard; es muss noch keine Implementierung geben
- **Draft Standard** - es gibt mindestens zwei unabhängige Implementierungen der RFC
- **Internet Standard** - allgemein anerkannter Standard. Die RFC 2616 (HTTP 1.1) ist so ein Standard

1.2 Die Backus-Naur-Form

Zur exakten Beschreibung von HTTP/1.1 wird in beiden RFCs die *erweiterte Backus-Naur Form* verwendet.

Die Backus-Naur Form (BNF) ist eine Syntaxbeschreibungssprache, die aus ein paar wenigen, einfachen Regeln besteht. Hier die wichtigsten BNF-Regeln, mit dem man bereits das HTTP und die URIs exakt definieren kann:

`"literal":` Alles was zwischen Gänsefüsschen steht, wird als reiner Text ins Protokoll eingefügt. D.h. der Text wird so wie er ist zwischen Client und Server übertragen.

`absoluteURI | absolutePath:` Das Zeichen | trennt *Alternativen*. Dabei ist **entweder** die eine **oder** die andere Alternative erlaubt, beide zusammen nicht.

`( generalHeader CRLF ):` Runde Klammern fassen die Elemente, die zwischen ihnen stehen zu einer Einheit zusammen. Die Klammern dienen nur dazu, diese Gruppierung zu beschreiben. Sie werden *nicht* ins Protokoll eingefügt.

`*( generalHeader CRLF ):` Der Stern * bewirkt, daß das nachfolgende Element beliebig oft wiederholt werden darf. Dabei ist auch die Null als Faktor erlaubt.

[messageBody]: Was zwischen eckigen Klammern "] steht ist optional, d.h. es kann, muss aber nicht im Protokolltext auftauchen.

CR: Carriage Return, ASCII-Zeichencode 13.

LF: Line Feed (Zeilenvorschub), ASCII-Zeichencode 10.

SP: Space (Leerzeichen), ASCII-Zeichencode 32.

2 Ein Beispiel: Das Anfordern einer html-Seite mit der GET-Methode

Mit der GET-Methode fordert der Client (sprich der html-Browser) eine html-Seite vom Server an. Wie der dazu notwendige Protokolltext aussieht, soll mit der BNF beschrieben werden.

Das Anfordern einer Seite ist ein sogenannter html-**request**. Dieser ist in der RFC 2616 definiert. Die für einen Request benötigten URIs legt die RFC 2396 fest. Im Folgenden stehen die für einen Request benötigten Definitionen. Um das Ganze nicht unübersichtlicher werden zu lassen als es ohnehin schon ist, wurden ein paar Optionen weggelassen und die Schreibweise etwas geändert:

```
Request = RequestLine
          *(
              (generalHeader | requestHeader | entityHeader) CRLF
          ) CRLF
          [messageBody]

   RequestLine = Method SP RequestURI SP httpVersion CRLF
      Method = "GET" | "POST" | "PUT" | ...
      RequestURI = "*" | absoluteURI | absolutePath | authority
         absoluteURI = scheme ":" (netPath | absolutePath) ["?" query]
            netPath = "//"authority[absolutePath]
               authority = host [":"port]
                  host = hostname | IPV4adress

   requestHeader = Accept | ... | Host | ... | UserAgent
      Host = "Host" ":" host [ ":"port ]
```

Abbildung 1: Die BNF zur Definition von http-Requests

Mit diesen Deklarationen liesse sich eine Unmenge von Requests formulieren. Wir wollen aus dieser Menge aber nur diejenigen ansehen, mit denen ein http-Client (Browser) mit der GET-Methode eine html-Seite von einem Server anfordert.

Und hierfür gibt es wiederum zwei Möglichkeiten.

1. Eine Anfrage mit einer absoluten URI (`absoluteURI`). Eine absolute URI erkennt man daran, daß sie mit einem `scheme` (vgl. Abb. 1) beginnt. Die bekanntesten schemes sind `http:` und `ftp:`. Eine komplette http-Anfrage sieht dann zum Beispiel so aus:

   ```
   GET http://localhost:8080/micha/HalloUser.jsp HTTP/1.1 CRLF CRLF
   ```

   Dabei sind also:

```
method        =  GET
scheme        =  http
hostname      =  localhost
port          =  8080
absolutePath  =  /micha/HalloUser.jsp
httpVersion   =  HTTP/1.1
```

Diese Form einer GET-Anfrage ist *zwingend notwendig*, wenn der Browser die Anfrage zunächst an einen Proxy sendet. Bei zukünftigen http-Versionen sollen Requests nur mit absoluten URIs möglich sein. Um also später den übergang zu absoluten URIs zu vereinfachen, müssen bereits jetzt HTTP/1.1-Server die absoluteURI-Form akzeptieren.

2. Die zweite, zur Zeit bekanntere Möglichkeit ist eine Anfrage, bei der als `requestURI` ein absoluter Pfad auf einem Rechner angegeben wird. Die Netzadresse des Rechners wird dann im `requestHeader` festgelegt:

   ```
   GET /micha/HalloUser.jsp HTTP/1.1 CRLF
   Host: localhost:8080 CRLF
   ```

   Hier sind also:

   ```
   requestHeader  =  Host
   Host           =  Host: localhost:8080
   ```

3 Die Antwort des Servers auf die Anfrage

Schickt man eine der beiden Requests an den entsprechenden http-Server, erhält man z.B. folgende Antwort (`Response`):

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=24C9704B2EF60B1F4E17784503364523; Path=/micha
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 545
Date: Mon, 15 Sep 2003 20:02:52 GMT
Server: Apache Coyote/1.0

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Erster Test mit Parametern</title>
  </head>

  <body>
    <h3> Bitte einen Parameter mit der Hand am Arm an das URI anhaengen </h3>
        <p> Beispiel: http://localhost:8080/micha/HalloUser.jsp?name=micha </p>
        Hallo, salute null!

    <hr>
    <address><a href="mailto:dienert@wara.de">michael</a></address>
<!-- Created: Tue Jan  7 00:08:25 CET 2003 -->
<!-- hhmts start -->
Last modified: Tue Jan  7 00:10:40 CET 2003
<!-- hhmts end -->
  </body>
</html>
```

Die allgemeine Form einer Response sieht so aus:

```
Response = StatusLine
           *(
               (generalHeader | requestHeader | entityHeader) CRLF
           ) CRLF
           [messageBody]
   StatusLine = HTTPVersion SP StatusCode SP ReasonPhrase CRLF
```

Die erste Zeile der Response ist noch einfach zu identifizieren: Es ist die `StatusLine`, zusammengesetzt aus HTTP-Version, `StatusCode` und einer Klartextangabe des StatusCodes (`ReasonPhrase`). Hier im Beispiel hat der Status-Code den Wert 200, die Anfrage war also in Ordnung.

Die Status-Codes sind dreistellig. Mit der ersten Ziffer wird eine Grobeinteilung in verschiedene Response-Arten vorgenommen:

- 1xx: Request erhalten, Verarbeitung ist noch im Gange
- 2xx: Request wurde erfolgreich empfangen, interpretiert und beantwortet
- 3xx: Redirect = umadressieren (Weiterleiten) der Anfrage
- 4xx: Fehler des Client, z.B. 400 Bad Request
- 5xx: Fehler des Servers, z.B. 501 Not Implemented

Bei den Header-Zeilen wird es wegen der drei Optionen und dem *-Zeichen (Wiederholungszeichen) mit der Zuordnung schon schwieriger, weshalb hier nur kurz die Zeilen des Beispiels erklärt werden.

`Content-Type:` Dieser Header legt fest, um was es sich bei dem `messageBody` handelt. Ist es Text oder Daten eines Bildes etc. Diese Information ist für den Client = Browser wichtig, um den Inhalt des `messageBody` korrekt darstellen zu können. Ein Server *sollte* immer den Content-Type angeben, das ist aber nicht zwingend vorgeschrieben.

`Content-Length:` Gibt die Länge des `messageBody` in Bytes an. Der Server sollte die `Content-Length` angeben.

`Content-Encoding:` Das kommt hier im Beispiel nicht vor, soll aber kurz erläutert werden: Mit dem `Content-Encoding` kann man angeben, dass der message-Body z.B. mit **`gzip`** (GNU-Zip) komprimiert wurde.

`Date:` Datum und Uhrzeit des Servers beim Absenden der Antwort. Der Date-Header *muss* in einem Format wie in RFC 1123 beschrieben, gesendet werden.

`Server:` Dieser Header enthält Informationen über die Server-Software. Ein Proxy *darf* dieses Feld nicht verändern, sondern kann einen **`Via:`**-Header hinzufügen.