

II. Rechnerkommunikation und Protokolle

- ❑ Rechnernetze
- ❑ Prinzipien des Datenaustauschs
- ❑ Netzsoftware und Kommunikationsprotokolle
- ❑ Internetworking
- ❑ Client-Server-Interaktionsmodell
- ❑ Uniform Resource Locator
- ❑ Hypertext-Transfer-Protokoll HTTP
- ❑ Fortgeschrittene HTTP-Konzepte

Hypertext-Transfer-Protokoll HTTP

Der HTTP-Standard sieht das Client-Server-Prinzip mit folgenden Funktionseinheiten vor [\[RFC 2616\]](#) :

1. WWW-Client bzw. User-Agent

Initiiert Verbindungen zu WWW-Servern; der WWW-Client ist in der Regel ein Web-Browser.

2. WWW-Server

Wartet auf Verbindungswünsche von WWW-Clients und antwortet auf die gestellten Anfragen; liefert gewünschte Ressource oder Statusinformation.

3. Proxy-Server

System zwischen WWW-Client und WWW-Server; arbeitet sowohl als WWW-Server (hat aufgrund früherer Kommunikation Antworten im Cache) als auch als WWW-Client gegenüber dem sogenannten *Origin-Server*.

4. Gateway

Vergleichbar dem Proxy-Server mit dem Unterschied, dass der WWW-Client keine Kenntnis über die Existenz des Gateways besitzt.

Hypertext-Transfer-Protokoll HTTP

Der HTTP-Standard sieht das Client-Server-Prinzip mit folgenden Funktionseinheiten vor [\[RFC 2616\]](#) :

1. **WWW-Client bzw. User-Agent**

Initiiert Verbindungen zu WWW-Servern; der WWW-Client ist in der Regel ein Web-Browser.

2. **WWW-Server**

Wartet auf Verbindungswünsche von WWW-Clients und antwortet auf die gestellten Anfragen; liefert gewünschte Ressource oder Statusinformation.

3. **Proxy-Server**

System zwischen WWW-Client und WWW-Server; arbeitet sowohl als WWW-Server (hat aufgrund früherer Kommunikation Antworten im Cache) als auch als WWW-Client gegenüber dem sogenannten *Origin-Server*.

4. **Gateway**

Vergleichbar dem Proxy-Server mit dem Unterschied, dass der WWW-Client keine Kenntnis über die Existenz des Gateways besitzt.

Hypertext-Transfer-Protokoll HTTP

Historie

1992 HTTP/0.9

1996 HTTP/1.0 [[RFC 1945](#)]

1997
– HTTP/1.1 [RFC: [2068](#), [7235](#)] [[W3C](#)]
2014

2015 HTTP/2 [[RFC 7540](#)] [[HTTP/2 Home](#)] [[Wikipedia](#)]

Das HTTP-Protokoll spezifiziert Nachrichtentypen, Datentransfer, Regeln zur Darstellung (Zeichensatz, Datenformate), Inhaltsabstimmung, Authentisierung, etc. zwischen den genannten Funktionseinheiten.

Kommunikationsablauf aus Client-Sicht:

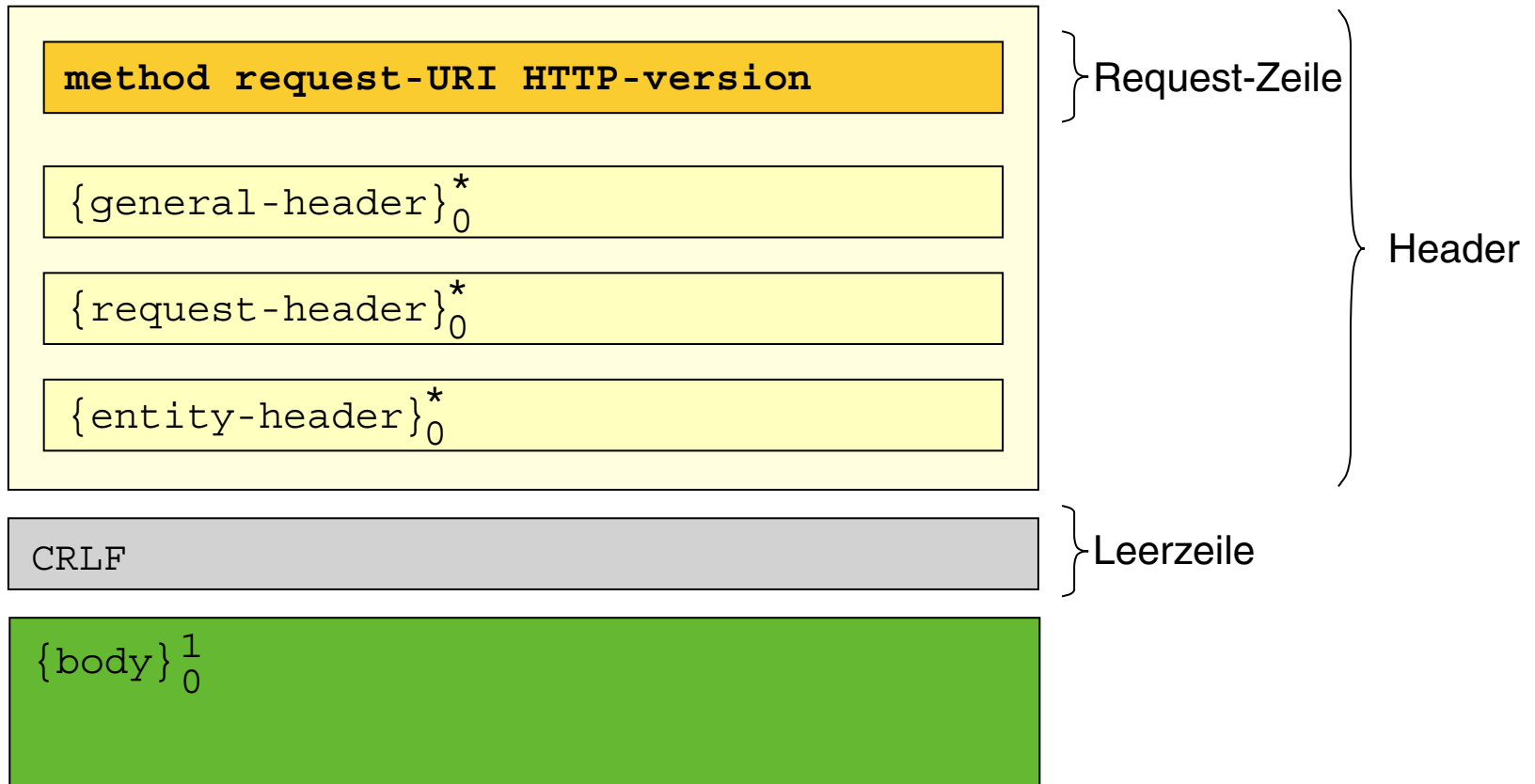
1. Öffnen einer TCP/IP Verbindung zum WWW-Server
2. **Request.** Senden der Anforderung an WWW-Server
3. **Response.** Empfangen der Antwort vom WWW-Server
4. Schließen der Verbindung

Bemerkungen:

- ❑ HTTP/0.9 versteht nur die GET-Methode, keine Statusinformation und auch keine Information über Medientypen.
- ❑ HTTP/1.1 ermöglicht unter anderem Pipelining: mehrere HTTP-Anfragen werden über *eine* TCP/IP-Verbindung abgewickelt.
- ❑ HTTP/2 ist deutlich verbessert hinsichtlich Performanz, abwärtskompatibel zu HTTP/1.1 und unterstützt das Zusammenfassen von mehreren HTTP-Anfragen via Multiplexing. Die Entwicklung wurde von Google ([SPDY](#)) und Microsoft forciert.

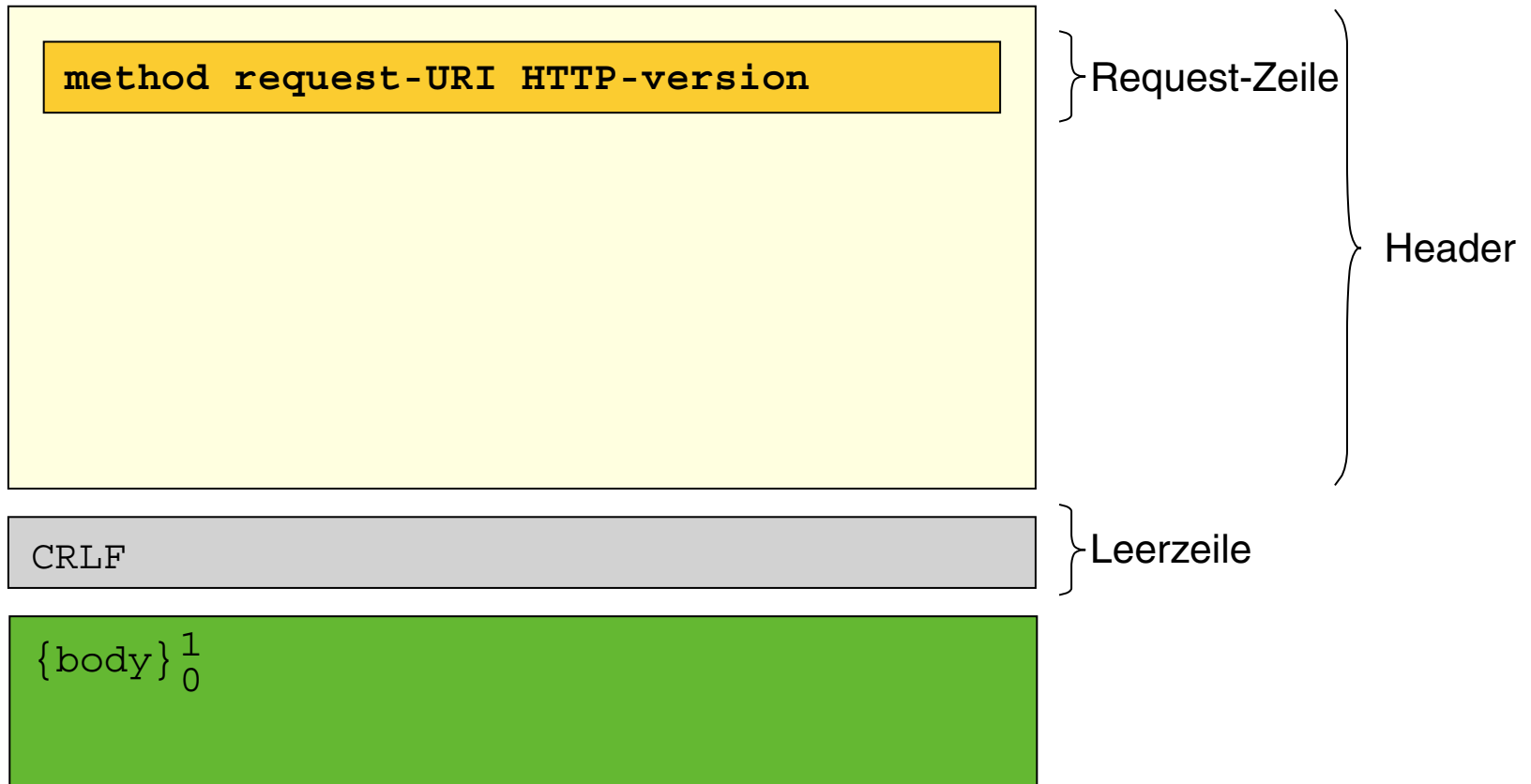
Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message [Message-Header]



Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message



Beispiel für Request-Zeile: `GET www.uni-weimar.de/index.html/ HTTP/1.0`

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Methoden

Methode

GET Anfrage der im Request-URI angegebenen Ressource. Client-Daten wie z.B. HTML-Feldwerte werden **als Bestandteil der URI** der Ressource übergeben.

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Methoden

Methode

GET	Anfrage der im Request-URI angegebenen Ressource. Client-Daten wie z.B. HTML-Feldwerte werden als Bestandteil der URI der Ressource übergeben.
POST	Wie GET, jedoch werden Client-Daten nicht an die URI angehängt, sondern im Message-Body untergebracht.
HEAD	Wie GET, jedoch darf der Server keinen Message-Body zurücksenden. Wird u.a. zur Cache-Validierung verwendet.

Hypertext-Transfer-Protokoll HTTP

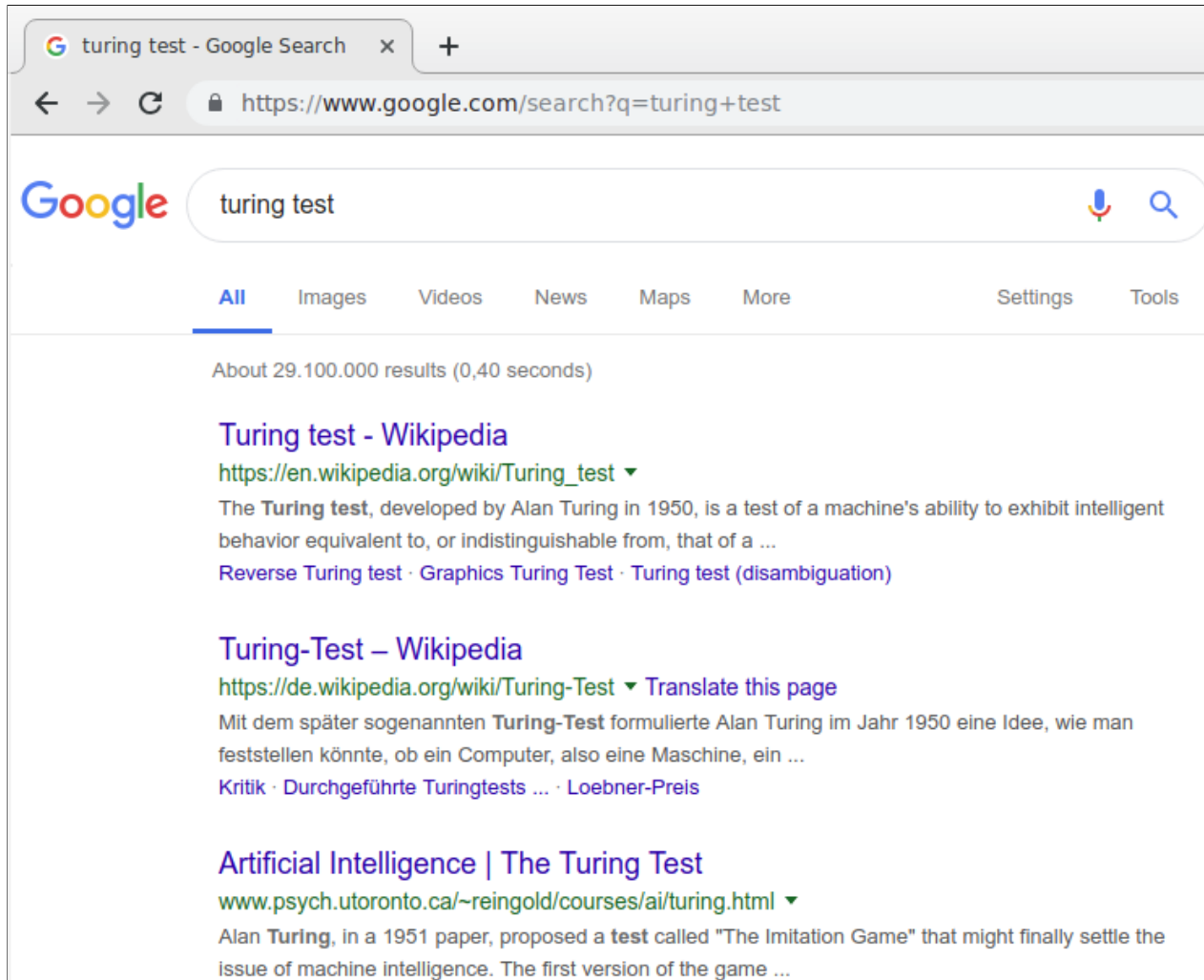
HTTP-Request-Message: Methoden

Methode

GET	Anfrage der im Request-URI angegebenen Ressource. Client-Daten wie z.B. HTML-Feldwerte werden als Bestandteil der URI der Ressource übergeben.
POST	Wie GET, jedoch werden Client-Daten nicht an die URI angehängt, sondern im Message-Body untergebracht.
HEAD	Wie GET, jedoch darf der Server keinen Message-Body zurücksenden. Wird u.a. zur Cache-Validierung verwendet.
PUT	Client erzeugt mit Daten des Message-Body auf dem Server eine neue Ressource an der Stelle der angegebenen Request-URI.
DELETE	Löschen der im Request-URI angegebenen Ressource auf dem Server.
OPTIONS	Abfrage der vorhandenen Kommunikationsmöglichkeiten entlang der Verbindungsstrecke zum Server.
TRACE	Verfolgen eines Requests auf dem Weg zum Server durch die Proxies.
CONNECT	Verbindungsherstellung zum Proxy-Server, um Tunnelbetrieb einzurichten. Anwendung: Einrichtung einer SSL-Verbindung (<i>Secure Socket Layer</i>)

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (1)



[www.google.de]

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet webtec.webis.de 80
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet webtec.webis.de 80  
Trying 141.54.132.157...  
Connected to webtec.webis.de.  
Escape character is '^]'.
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet webtec.webis.de 80
Trying 141.54.132.157...
Connected to webtec.webis.de.
Escape character is '^]'.
GET /helloworld.html HTTP/1.1
HOST: webtec.webis.de
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet webtec.webis.de 80
```

```
Trying 141.54.132.157...
```

```
Connected to webtec.webis.de.
```

```
Escape character is '^]'.
```

```
GET /helloworld.html HTTP/1.1
```

```
HOST: webtec.webis.de
```

```
HTTP/1.1 200 OK
```

```
Server: nginx
```

```
Date: Tue, 16 Apr 2019 10:08:49 GMT
```

```
Content-Type: text/html
```

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="utf-8">
```

```
  <title>Helloworld</title>
```

```
</head>
```

```
<body>
```

```
  <h1>Hello World</h1>
```

```
</body>
```

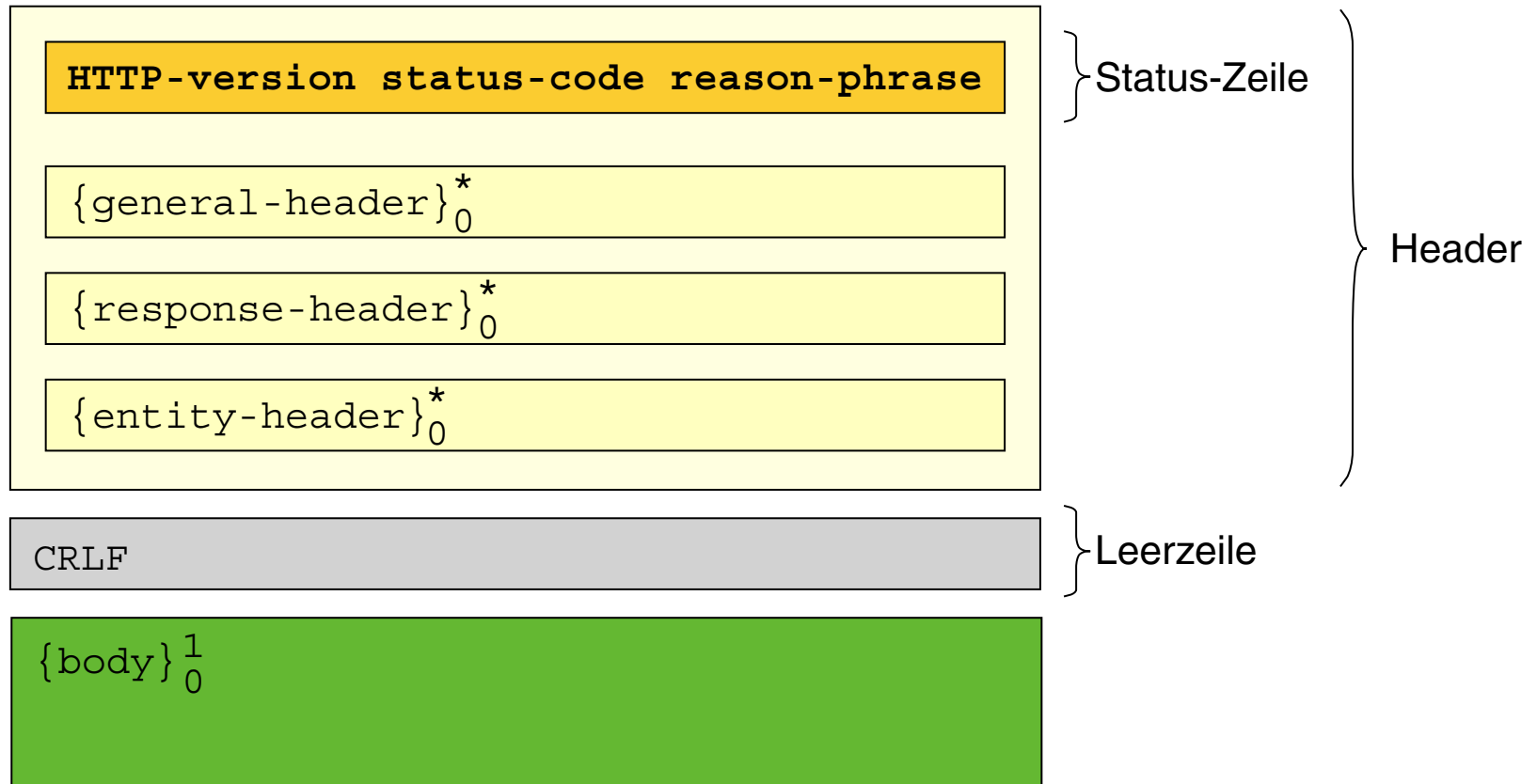
```
</html>
```



[\[webtec.webis.de\]](https://webtec.webis.de)

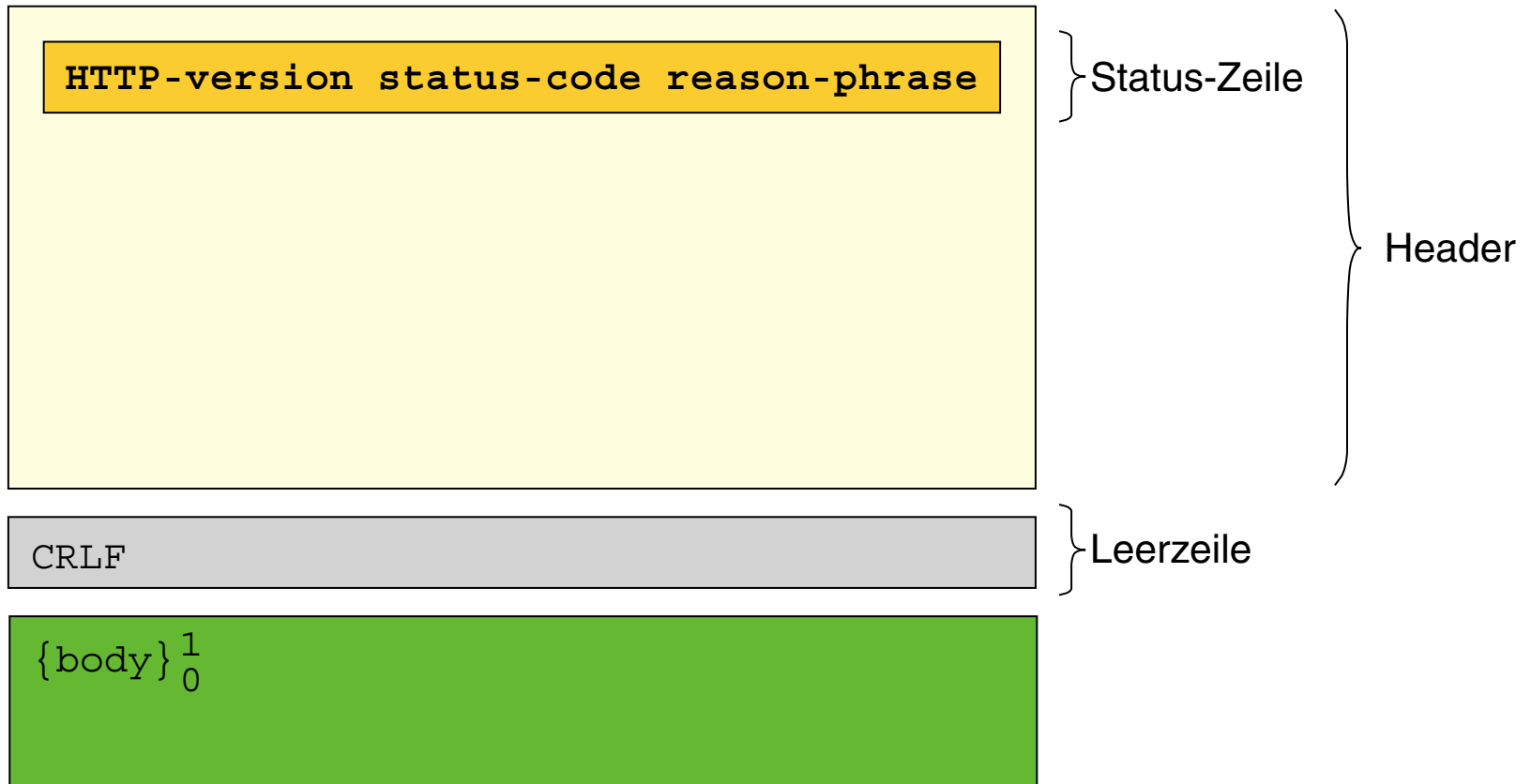
Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message [Message-Header]



Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message



Beispiel für Status-Zeile: `HTTP/1.1 200 OK`

Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message: Status-Codes

Der Status-Code besteht aus 3 Ziffern und gibt an, ob eine Anfrage erfüllt wurde bzw. welcher Fehler aufgetreten ist.

Status-Code-Kategorie

1xx	Informational	Anforderung bekommen (selten verwendet).
2xx	Success	Anforderung bekommen, verstanden, akzeptiert und ausgeführt.
3xx	Redirection	Anweisung an den Client, an welcher Stelle die Seite zu suchen ist.
4xx	Client Error	Fehlerhafte Syntax oder unerfüllbar, da z.B. Seite nicht vorhanden.
5xx	Server Error	Server kann Anforderung nicht ausführen aufgrund eines Fehlers in der Systemsoftware oder wegen Überlastung.

Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message: Status-Codes (Fortsetzung)

100 Continue

101 Switching protocols

200 OK

201 Created

202 Accepted

203 Non-authoritative information

204 No content

205 Reset content

206 Partial content

300 Multiple choices

301 Moved permanently

302 (veraltet) --> 307

303 See other

304 Not modified

305 Use proxy

307 Temporary redirect

400 Bad request

401 Unauthorized

402 Payment required

403 Forbidden

404 Not found

405 Method not allowed

406 Not acceptable

407 Proxy authentication required

408 Request timeout

411 Length required

412 Precondition failed

413 Request entity too large

414 Request URI too large

415 Unsupported media type

416 Requested range not satisfiable

417 Expectation failed

418 I'm a teapot

424 Site too ugly

500 Internal server error

501 Not implemented

502 Bad gateway

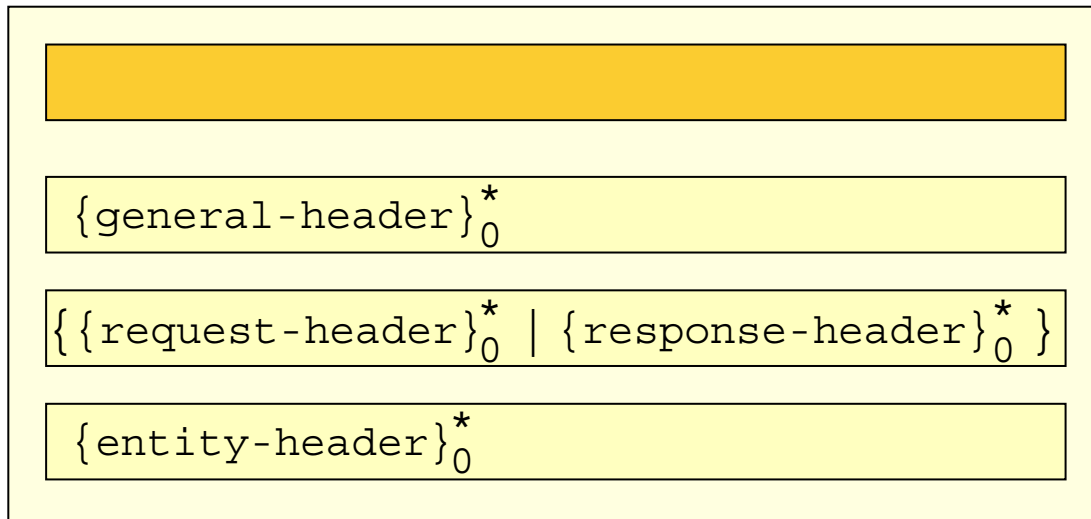
503 Service unavailable

504 Gateway time out

505 HTTP version not supported

Hypertext-Transfer-Protokoll HTTP

HTTP-Message-Header [Request-Message] [Response-Message]



- ❑ General-Header
Meta-Information zu Protokoll und Verbindung.
- ❑ Request-Header | Response-Header
Informationen zur Anfrage oder zum Client | Antwort des Servers.
- ❑ Entity-Header
Meta-Information über den Inhalt im Message-Body.
Beispiele: Content-Encoding, Content-Language, Content-Type

Bemerkungen:

- ❑ BNF-Notation für den Aufbau von Header-Zeilen:

header ::= keyword ':' value

keyword ::= 'Date' | 'Server' | 'Last-Modified' | 'Set-Cookie' |
'Content-Length' | 'Content-Type' | ...

- ❑ Analyse der Redirects einer URL mit [HTTP-Status-Code-Checker](#).

Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message: Beispielantwort [Java]

HTTP/1.1 200 OK

Status line

Date: Tue, 15 Apr 2014 19:17:35 GMT

General header

Server: Apache/2.2.12 (Unix) DAV/1.0.3 PHP/4.3.10
mod_ssl/2.8.16 OpenSSL/0.9.7c

Response header

Last-Modified: Sat, 22 Mar 2014 14:11:21 GMT
ETag: "205e812-1479-42402789"

Entity header

Accept-Ranges: bytes

Response header

Content-Length: 5241

Entity header

Connection: close

General header

Content-Type: text/html; charset=utf-8

Entity header

```
<!DOCTYPE html>
<html lang="de-DE" xmlns="http://www.w3.org/1999/...
<head>
<base href="http://www.uni-weimar.de">
<title>Bauhaus-Universit&auml;t Weimar</title>
...
```

Hypertext-Transfer-Protokoll HTTP

Content Type / MIME Type / Media Type

Medientypen bestehen aus einem Top-Level-Typ und einem Untertyp und können durch Parameter weiter spezifiziert werden. [\[Wikipedia\]](#)

Typ	Untertyp	Beschreibung
text	plain	unformatierter ASCII-Text
	enriched	ASCII-Text mit einfachen Formatierungen
image	gif	Standbild im GIF-Format
	jpeg	Standbild im JPEG-Format
audio	basic	Klangdaten
application	octet-stream	nicht-interpretierte Byte-Folge
	postscript	druckbares Dokument im PostScript-Format
...

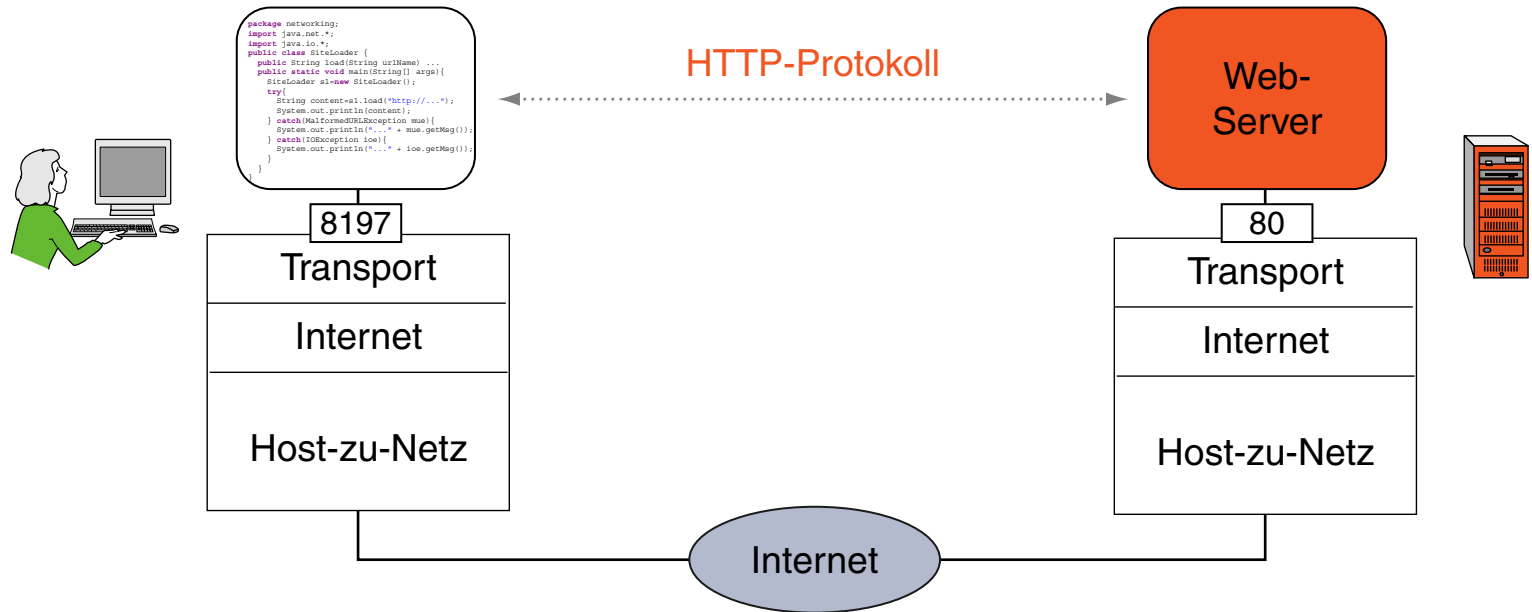
Eine Übersicht der erlaubten [Medientypen](#) findet sich bei der [IANA](#).

Bemerkungen:

- ❑ Die MIME-Type-Spezifikation ist in [RFC 2046](#) beschrieben.
- ❑ Die Abkürzung [MIME](#) steht für Multipurpose Internet Mail Extensions. Die zugehörigen Spezifikationen finden Anwendung bei der Deklaration von Inhalten in Internetprotokollen.

Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java [Server-side]

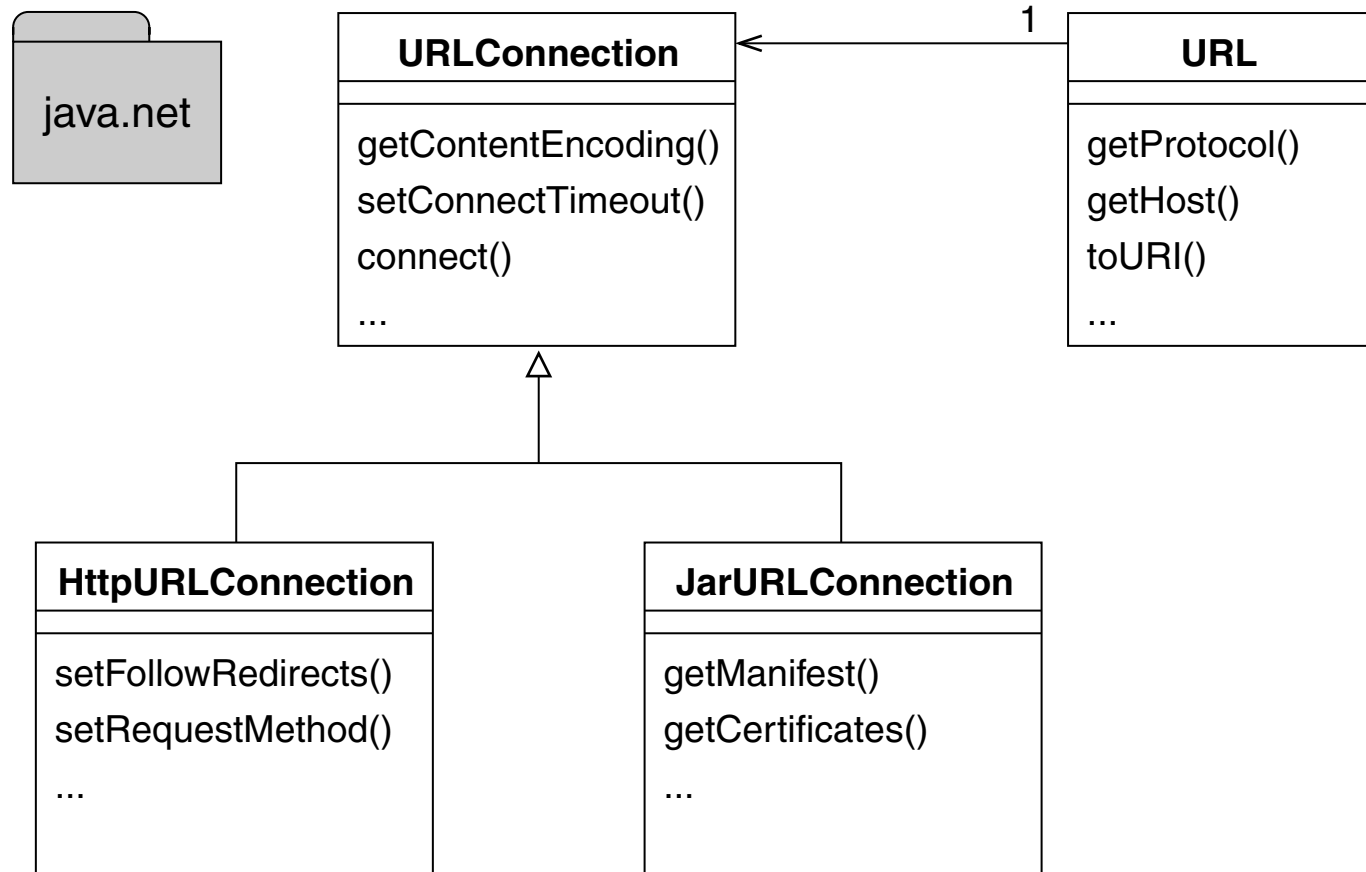


Auf der Server-Seite bildet ein WWW-Server das Gegenstück einer Verbindung zu dem Beispiel-Client. Der WWW-Server lässt sich auf diese Art anfragen, weil das HTTP-Protokoll vom Client korrekt abgewickelt wird.

[RFC 2616]

Hypertext-Transfer-Protokoll HTTP

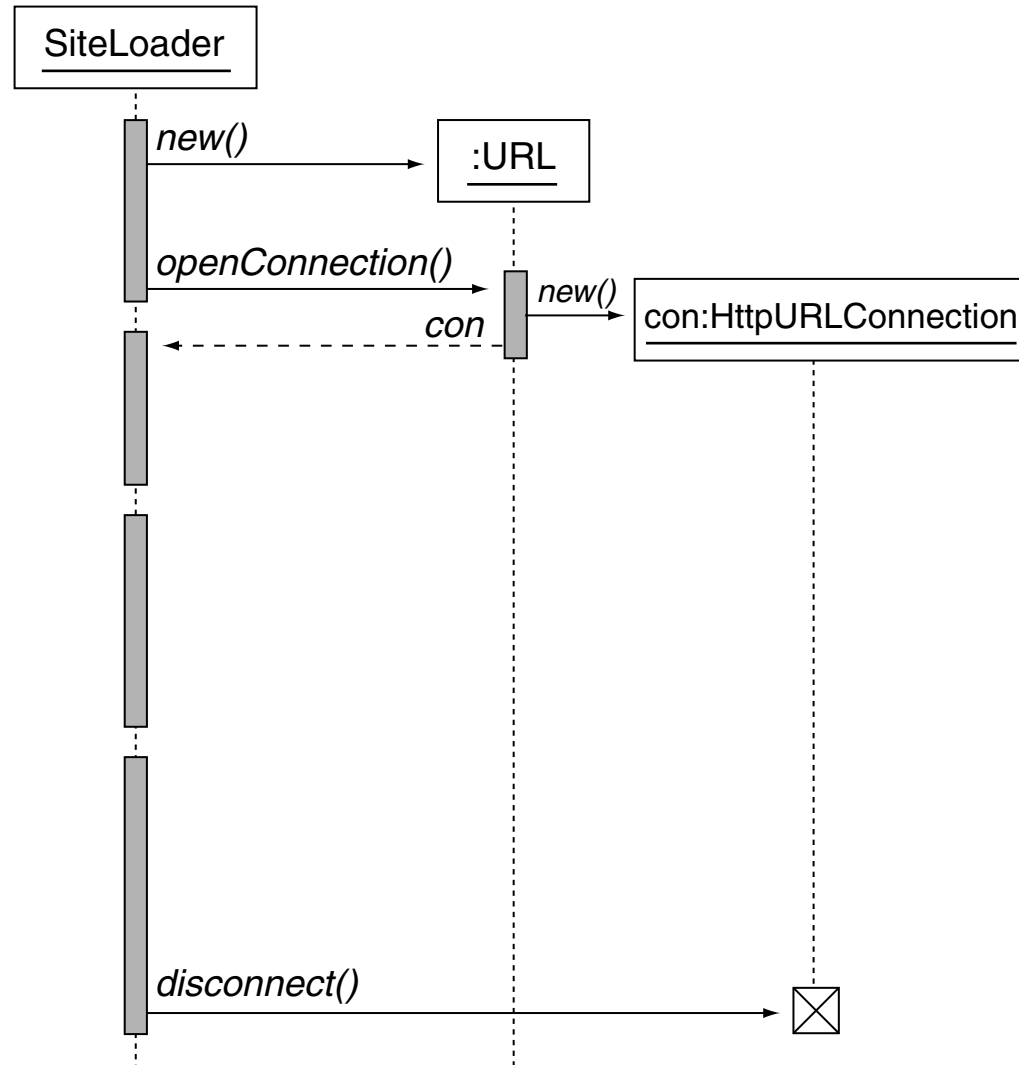
HTTP-Kommunikation mit Java



[[Javadoc](#)]

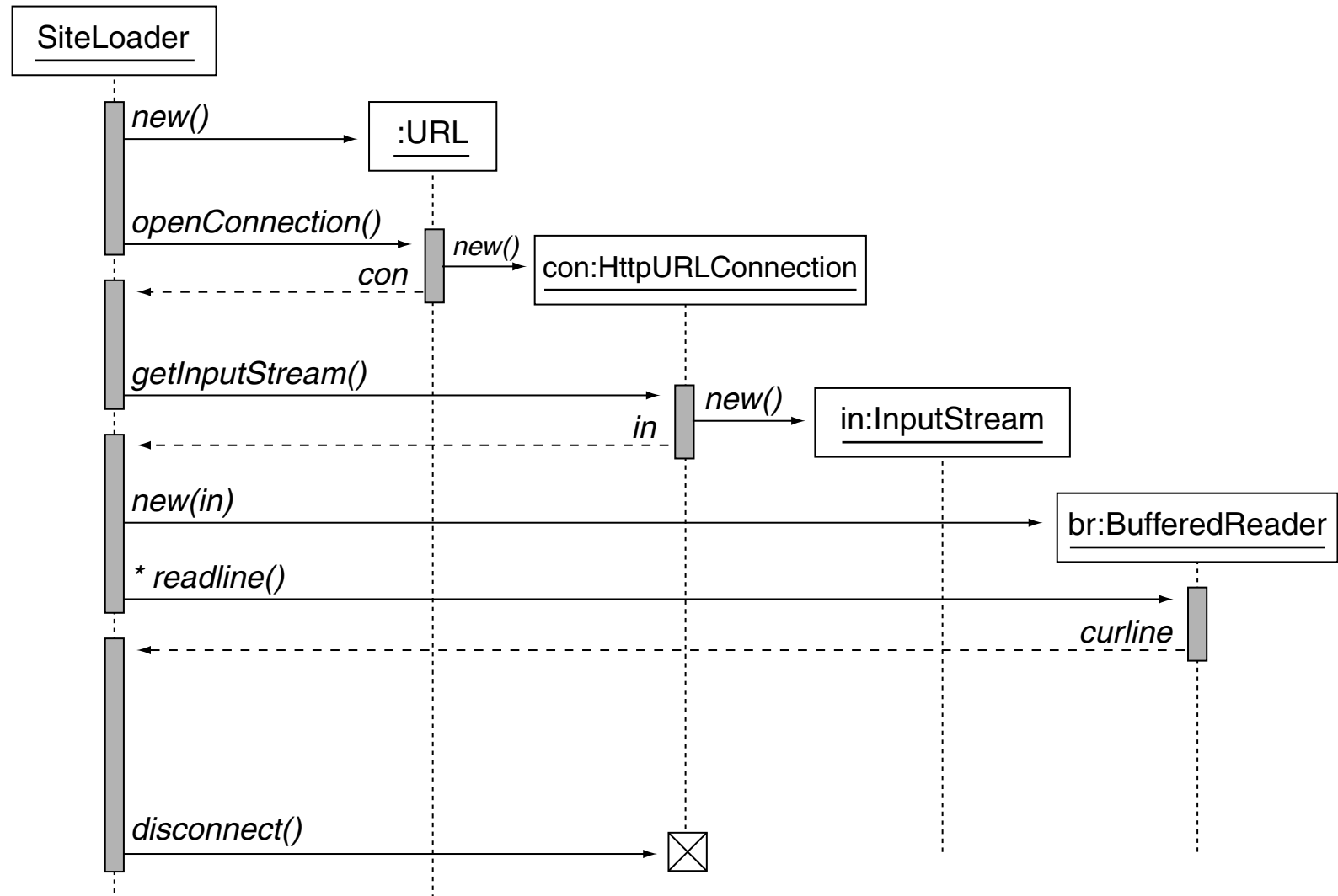
Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java



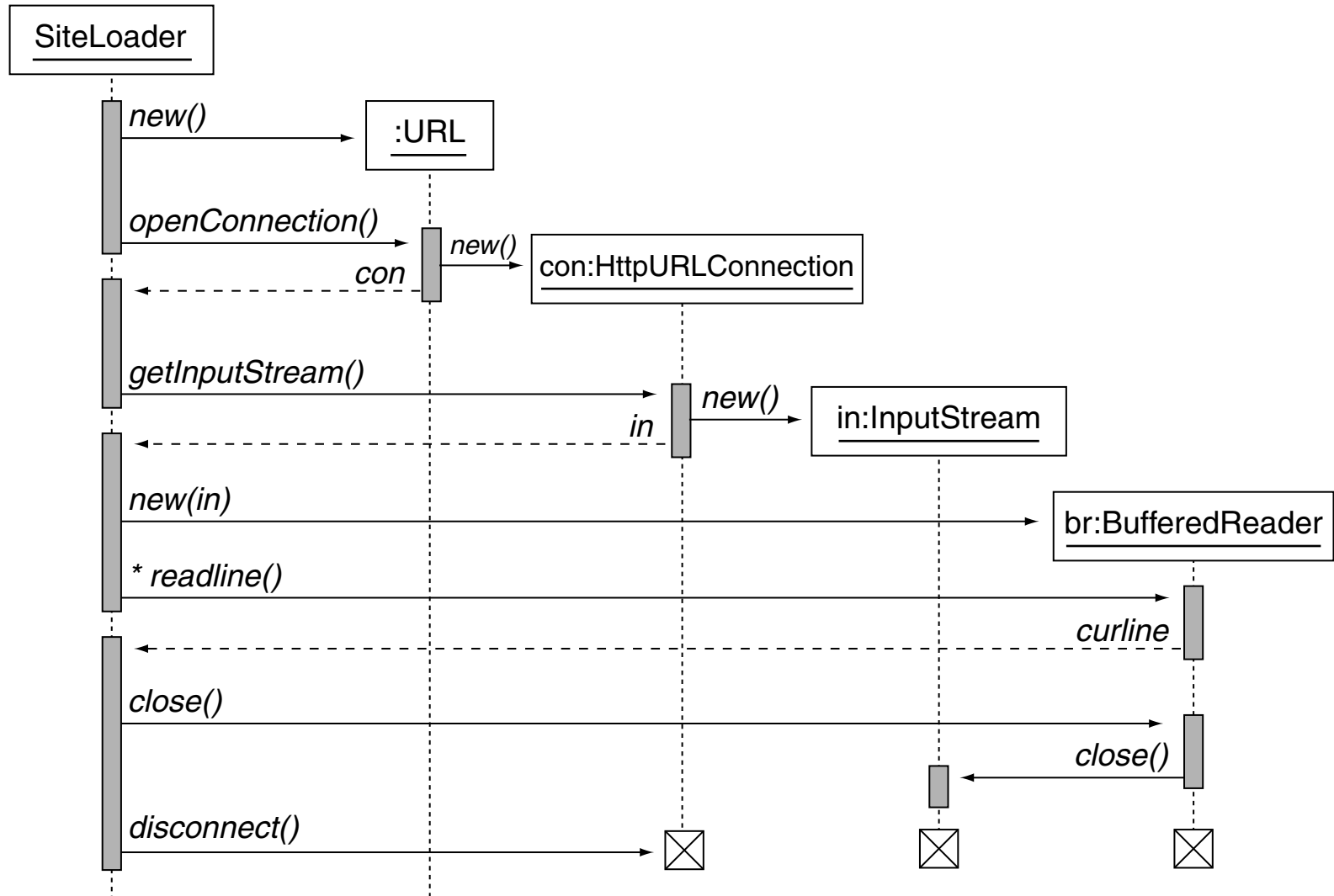
Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)



Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)



Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java [\[Response-Message\]](#)

```
public static String load(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    System.out.println(con.getResponseCode() + " "
        + con.getResponseMessage()); // will print "200 OK"

    InputStream in = con.getInputStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    String curline;
    StringBuilder content = new StringBuilder();

    while ((curline = br.readLine()) != null) {
        content.append(curline + '\n');
    }

    br.close();
    con.disconnect();
    return content.toString();
}
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)

```
package networkprotocol;

import java.net.*;
import java.io.*;

public class SiteLoader {

    public static String load(String urlString) ...

    public static void main(String[] args) {
        try{
            String content = SiteLoader.load("http://www.heise.de");
            System.out.println(content);
        }
        catch(MalformedURLException e) {
            System.out.println("MalformedURLException:" + e.getMessage());
        }
        catch(IOException e) {
            System.out.println("IOException:" + e.getMessage());
        }
    }
}
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)

```
[stein@webis bin]$ java networkprotocol.SiteLoader | less
```

```
200 OK
```

```
<!DOCTYPE html>
```

```
<html lang="de">
```

```
<head>
```

```
  <title>heise online - IT-News, Nachrichten und Hintergründe
```

```
  </title>
```

```
    <meta name="description" content="News und Foren zu Computer, IT, ...
```

```
      <meta name="keywords" content="heise online, c't, iX, Technology ...
```

```
    <script type="text/javascript" src="/js/mobi/webtrekk_abtest.js"></script>
```

```
<meta charset="utf-8">
```

```
<meta name="publisher" content="Heise Medien" />
```

```
<meta name="viewport" content="width=1175" />
```

```
<link rel="alternate" media="only screen and...
```

```
<link rel="copyright" title="Copyright" href="/impressum.html" />
```

```
      <!--googleoff: all-->
```

```
<meta property="fb:page_id"          content="333992367317" />
```

```
<meta property="og:title"            content="IT-News, c&#39;t, iX, Technology ...
```

```
<meta property="og:type"             content="website" />
```

```
<meta property="og:locale"           content="de_DE" />
```

```
<meta property="og:url"              content="http://www.heise.de/" />
```

```
...
```


Fortgeschrittene HTTP-Konzepte

Session-Management

HTTP ist ein **zustandsloses** Protokoll = ein Protokoll „ohne Gedächtnis“.

Ein zustandsloses Protokoll berücksichtigt keine Information aus bereits stattgefundenener Kommunikation.

Eine **Session** beschreibt einen Dialog, der sich über mehrere Anfrage/Antwort-Zyklen erstreckt. Innerhalb einer Session soll sich auf die vorangegangene Kommunikation bezogen werden können.

Fortgeschrittene HTTP-Konzepte

Session-Management

HTTP ist ein **zustandsloses** Protokoll = ein Protokoll „ohne Gedächtnis“.

Ein zustandsloses Protokoll berücksichtigt keine Information aus bereits stattgefundenener Kommunikation.

Eine **Session** beschreibt einen Dialog, der sich über mehrere Anfrage/Antwort-Zyklen erstreckt. Innerhalb einer Session soll sich auf die vorangegangene Kommunikation bezogen werden können.

Techniken zur Codierung von Session-Information:

1. URL Rewriting

Session-Information wird in der URL codiert.

2. Cookies

Session-Information wird beim WWW-Client gespeichert.

3. Hidden Fields

Session-Information wird in unsichtbaren Formularfeldern untergebracht.

Fortgeschrittene HTTP-Konzepte

Session-Management

HTTP ist ein **zustandsloses** Protokoll = ein Protokoll „ohne Gedächtnis“.

Ein zustandsloses Protokoll berücksichtigt keine Information aus bereits stattgefundenener Kommunikation.

Eine **Session** beschreibt einen Dialog, der sich über mehrere Anfrage/Antwort-Zyklen erstreckt. Innerhalb einer Session soll sich auf die vorangegangene Kommunikation bezogen werden können.

Techniken zur Codierung von Session-Information:

1. URL Rewriting

Session-Information wird in der URL codiert.

2. Cookies

Session-Information wird beim WWW-Client gespeichert.

3. Hidden Fields

Session-Information wird in unsichtbaren Formularfeldern untergebracht.

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies

Ein Cookie (Keks) ist eine Zeichenkette (<4KB), die zwischen WWW-Client und WWW-Server ausgetauscht wird. Standardisiert in [RFC 6265:3](#).

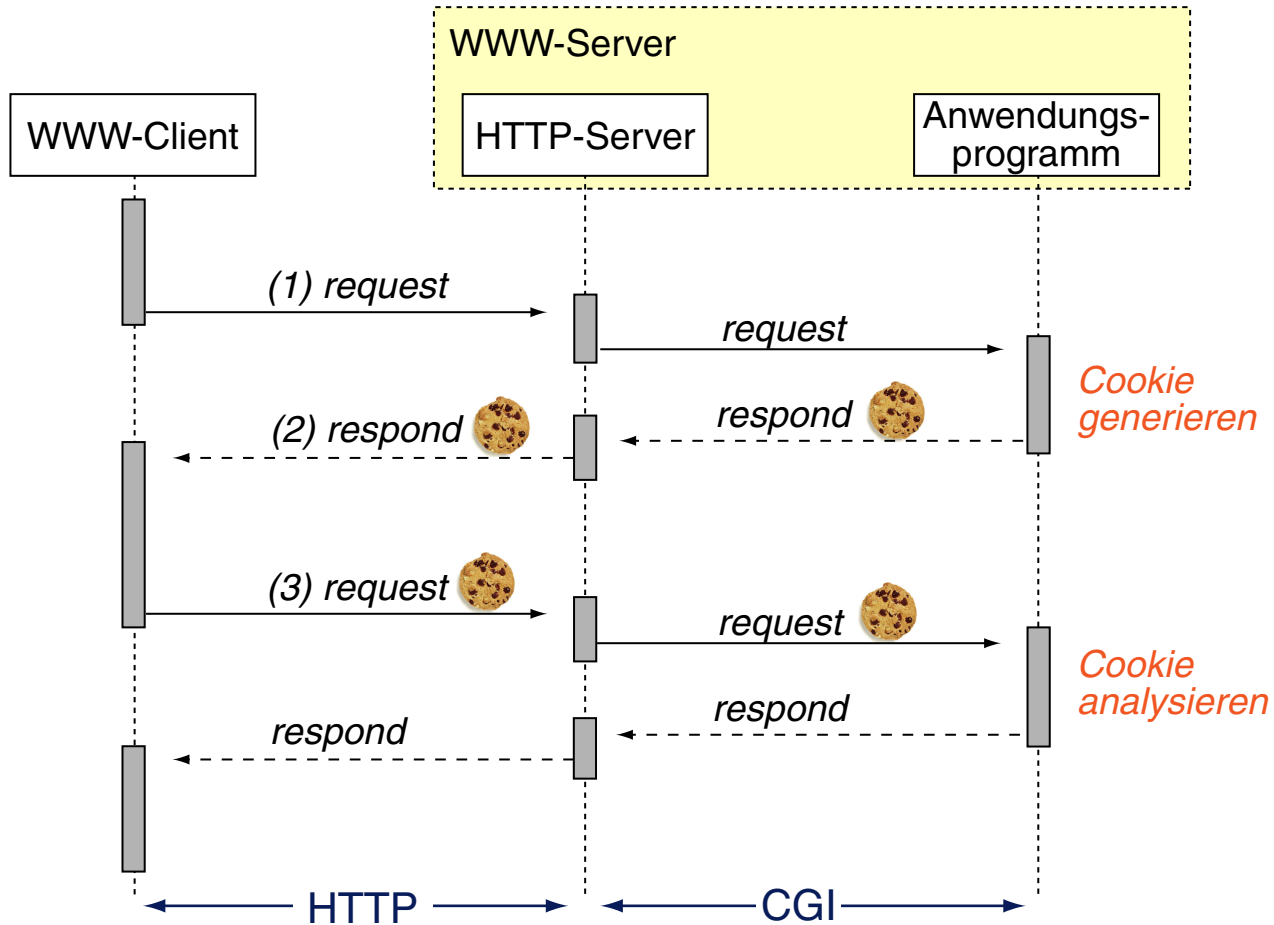


Verwendungsmöglichkeiten von Cookies:

- ❑ Session-Management
- ❑ Benutzer-Authentisierung
- ❑ Seitenabfolgesteuerung
- ❑ Erstellung von Nutzerprofilen
- ❑ Generierung nutzerspezifischer Seiten
- ❑ Informationsaustausch ergänzend zum HTTP-Protokoll

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies



[Meinel/Sack 2004]

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies

1. WWW-Client fragt Google-Startseite an:

```
Ethernet II, Src: 00:0c:f1:e8:fe:be , Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123 , Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577 , Dst Port: http (80), ...
Hypertext Transfer Protocol
  GET / HTTP/2.0
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;...
  Accept-Language: de
  Accept-Encoding: gzip, deflate, br
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) ...
  Host: www.google.de
  Connection: Keep-Alive
```

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies

1. WWW-Client fragt Google-Startseite an:

Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...

Hypertext Transfer Protocol

GET / HTTP/2.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;...

Accept-Language: de

Accept-Encoding: gzip, deflate, br

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) ...

Host: www.google.de

Connection: Keep-Alive

5	Anwendung:	HTTP-Message
4	Transport:	Port
3	Internet:	IP-Adresse
1+2	Host-zu-Netz:	Mac-Adresse

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies (Fortsetzung)

2. WWW-Server antwortet:

Ethernet II, Src: 00:09:e9:a6:0b:fc, Dst: 00:0c:f1:e8:fe:be
Internet Protocol, Src Addr: 66.249.85.99, Dst Addr: 141.54.178.123
Transmission Control Protocol, Src Port: http (80), Dst Port: 1577, ...

Hypertext Transfer Protocol

HTTP/2.0 200 OK

Cache-Control: private, max-age=0

Content-Type: text/html; charset=UTF-8

Set-Cookie: NID=181=qUoo...9Ya8; domain=.google.de; HttpOnly

Content-Encoding: gzip

Content-Length: 57433

Date: Tue, 16 Apr 2019 10:23:32 GMT

...

Content-encoded entity body (gzip)

Line-based text data: text/html

<html><head><meta http-equiv=content-typecontent=text/html; ...

5	Anwendung:	HTTP-Message
4	Transport:	Port
3	Internet:	IP-Adresse
1+2	Host-zu-Netz:	Mac-Adresse

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies (Fortsetzung)

3. WWW-Client sendet Google-Query „test“:

Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...

Hypertext Transfer Protocol

GET /search?...&q=test&...

Referer: https://www.google.de/

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;...

Accept-Language: de

Accept-Encoding: gzip, deflate, br

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) ...

Host: www.google.de

Connection: Keep-Alive

Cookie: NID=181=qUoo...9Ya8; ...

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies (Fortsetzung)

3. WWW-Client sendet Google-Query „test“:

Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...

Hypertext Transfer Protocol

GET /search?...&q=test&...

Referer: https://www.google.de/

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;...

Accept-Language: de

Accept-Encoding: gzip, deflate, br

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) ...

Host: www.google.de

Connection: Keep-Alive

Cookie: NID=181=qUoo...9Ya8; ...

Auf Client-Seite gespeicherter Cookie:

NID

181=qUoo...9Ya8

Bemerkungen:

- ❑ Nicht der Web-Server, sondern ein Anwendungsprogramm ist am Setzen von Cookies interessiert.
- ❑ Der Client kann Cookies mit einer Lebensdauer versehen und spezifisch für Web-Seiten erlauben oder sperren.
- ❑ Der Client wählt das passende Cookie anhand der URL der Informationsquelle aus und schickt es mit der Anfrage zum Web-Server. Dabei darf der Client nur Cookies an denjenigen Web-Server senden, von dem diese Cookies stammen.
- ❑ Der Client bringt Cookie-Information im [Request-Header](#) Cookie unter.
- ❑ Der Server bringt Cookie-Information im [Reponse-Header](#) Set-Cookie unter.
- ❑ Der Internet-Explorer unter Windows speichert Cookies im Verzeichnis Dokumente und Einstellungen/Cookies.
- ❑ Analyse des Protokollstapels mit dem Programm [wireshark](#).

Fortgeschrittene HTTP-Konzepte

Content-Negotiation

Ressourcen auf dem WWW können in sprachspezifischen, qualitätsspezifischen oder codierungsspezifischen Varianten vorliegen, besitzen jedoch dieselbe URI.

Seit HTTP/1.1 können WWW-Client und WWW-Server aushandeln, welche der angebotenen Varianten einer Informationsressource geliefert werden soll.

Arten der Content-Negotiation:

1. **Server-driven.** WWW-Server verantwortlich für Auswahl.
Kriterien: Header des HTTP-Requests, Informationen über die Ressourcen
2. **Agent-driven.** WWW-Client verantwortlich für Auswahl.
Client informiert sich zuerst, trifft dann Auswahlentscheidung.
3. **Transparent.** Proxy-Server verhandelt in der Agenten-Rolle.
Vorteile: Lastverteilung, WWW-Client stellt nur eine Anfrage.

Fortgeschrittene HTTP-Konzepte

Content-Negotiation

Ressourcen auf dem WWW können in sprachspezifischen, qualitätsspezifischen oder codierungsspezifischen Varianten vorliegen, besitzen jedoch dieselbe URI.

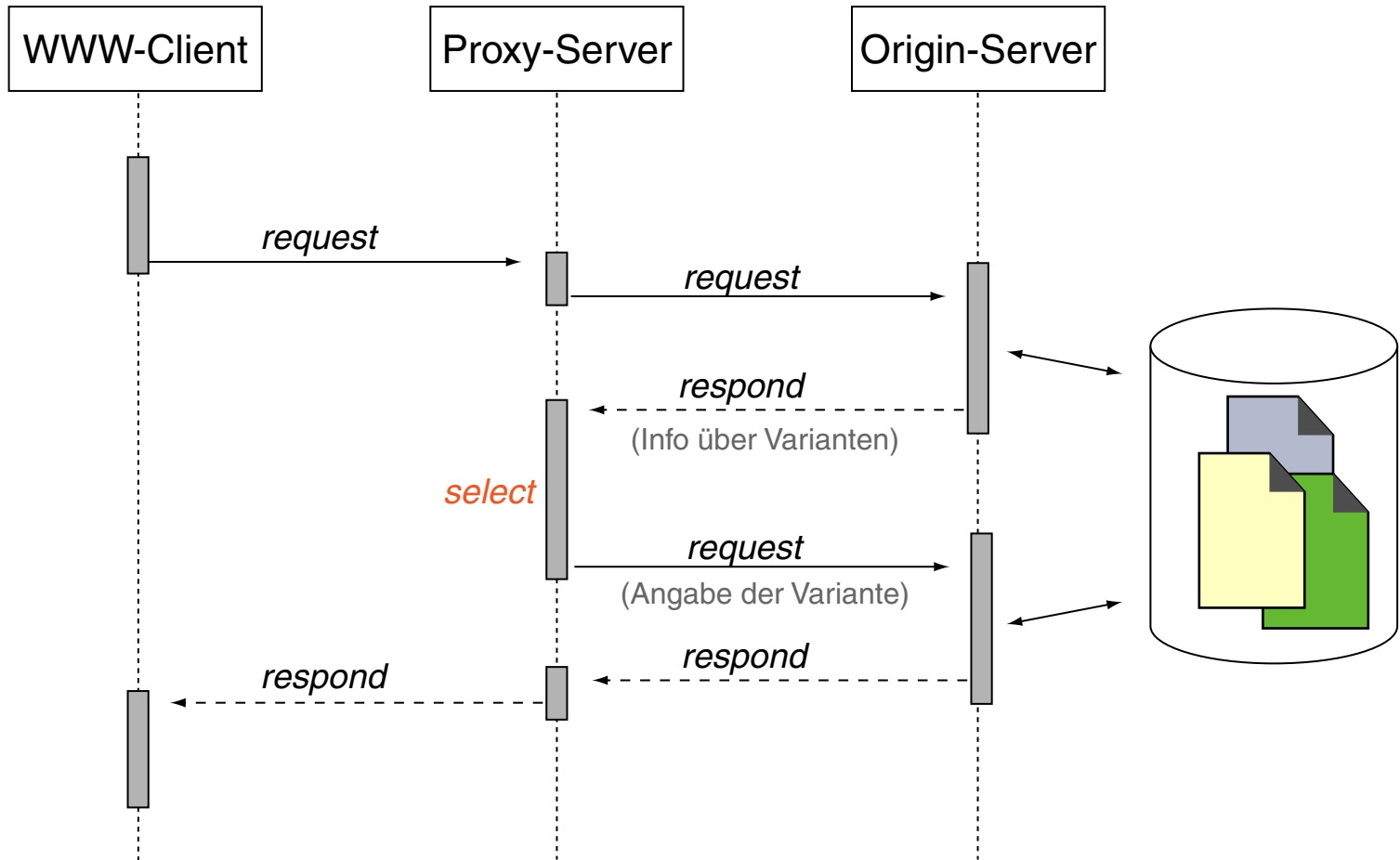
Seit HTTP/1.1 können WWW-Client und WWW-Server aushandeln, welche der angebotenen Varianten einer Informationsressource geliefert werden soll.

Arten der Content-Negotiation:

1. **Server-driven.** WWW-Server verantwortlich für Auswahl.
Kriterien: Header des HTTP-Requests, Informationen über die Ressourcen
2. **Agent-driven.** WWW-Client verantwortlich für Auswahl.
Client informiert sich zuerst, trifft dann Auswahlentscheidung.
3. **Transparent.** Proxy-Server verhandelt in der Agenten-Rolle.
Vorteile: Lastverteilung, WWW-Client stellt nur eine Anfrage.

Fortgeschrittene HTTP-Konzepte

Content-Negotiation: transparent



Bemerkungen:

- ❑ Die drei Varianten der Content-Negotiation unterscheiden sich dahingehend, wer die select-Operation ausführt.
- ❑ Vorteil der Server-driven Content-Negotiation: der Server ist nah an den Ressourcen und hat den besten Überblick. Nachteile: der Server ist auf Information aus den Request-Headern des Clients angewiesen, ist nicht so performant.
- ❑ Vorteil der Agent-driven Content-Negotiation: der Client weiß genau, was er will. Nachteil: Overhead an Kommunikation. Erst nachfragen, was es alles gibt, dann Auswahl.

Fortgeschrittene HTTP-Konzepte

Optimierte Ausnutzung der Verbindung

HTTP/1.0 öffnet für jede Anfrage eine Verbindung, die nach jeder Antwort unmittelbar geschlossen wird:

- + Protokoll sehr einfach, leicht zu implementieren
- + Verbindungsabbruch signalisiert auch Abschluss der HTTP-Antwort
- Verbindungsaufbau zeit- und ressourcenintensiv

HTTP/1.1 hat persistente Verbindungen, die Client-seitig mit `Connection: close` (General-Header) oder nach einem Timeout geschlossen werden:

- + effizientere Nutzung von Betriebssystemressourcen, weniger Pakete
- + *Pipelining*: Versenden einer weiteren Anfrage ohne auf Antwort zu warten
- aufwändigeres Protokoll, da *Chunked Encoding* notwendig

HTTP/2 hat Multiplexing (und andere Verbesserungen) [HTTP/2: [Home](#), [key differences](#)]

Fortgeschrittene HTTP-Konzepte

Optimierte Ausnutzung der Verbindung

HTTP/1.0 öffnet für jede Anfrage eine Verbindung, die nach jeder Antwort unmittelbar geschlossen wird:

- + Protokoll sehr einfach, leicht zu implementieren
- + Verbindungsabbruch signalisiert auch Abschluss der HTTP-Antwort
- Verbindungsaufbau zeit- und ressourcenintensiv

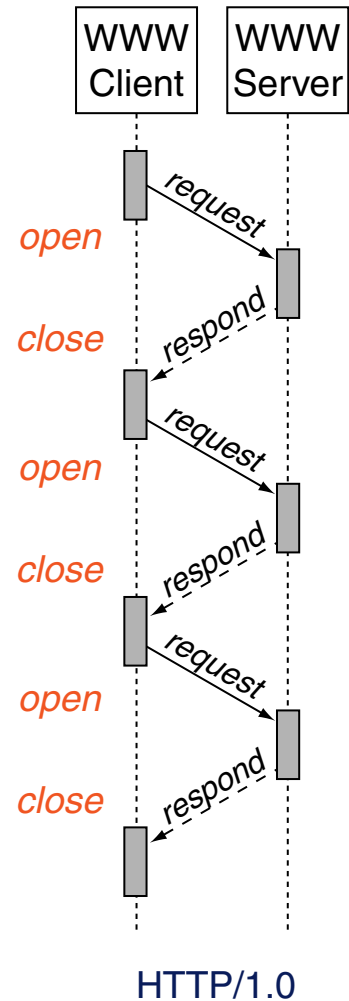
HTTP/1.1 hat persistente Verbindungen, die Client-seitig mit `Connection: close` (General-Header) oder nach einem Timeout geschlossen werden:

- + effizientere Nutzung von Betriebssystemressourcen, weniger Pakete
- + *Pipelining*: Versenden einer weiteren Anfrage ohne auf Antwort zu warten
- aufwändigeres Protokoll, da *Chunked Encoding* notwendig

HTTP/2 hat Multiplexing (und andere Verbesserungen) [HTTP/2: [Home](#), [key differences](#)]

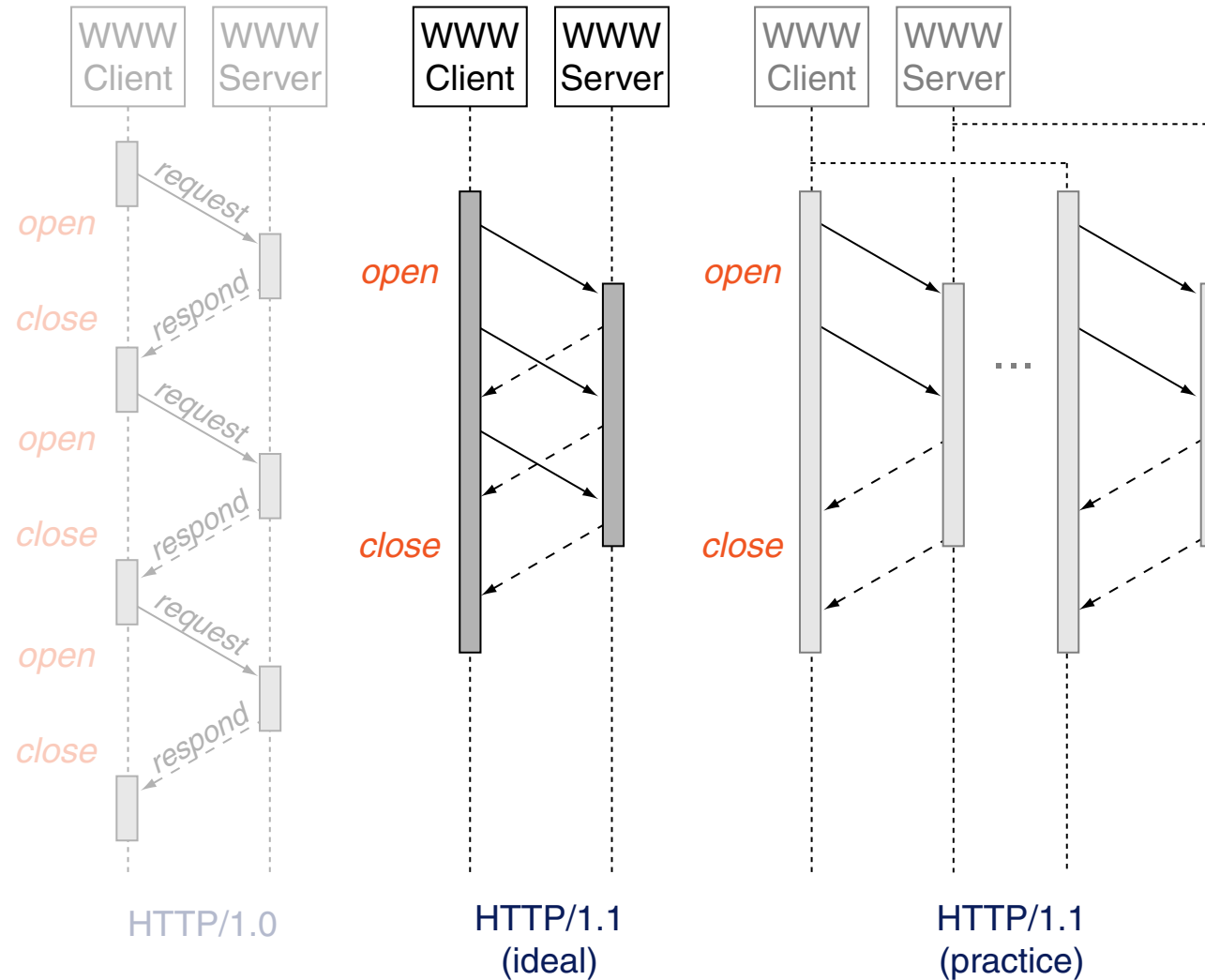
Fortgeschrittene HTTP-Konzepte

Optimierte Ausnutzung der Verbindung



Fortgeschrittene HTTP-Konzepte

Optimierte Ausnutzung der Verbindung



Fortgeschrittene HTTP-Konzepte

Optimierte Ausnutzung der Verbindung

