Distributed Computing

Lesson 10: HTTP

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Overview

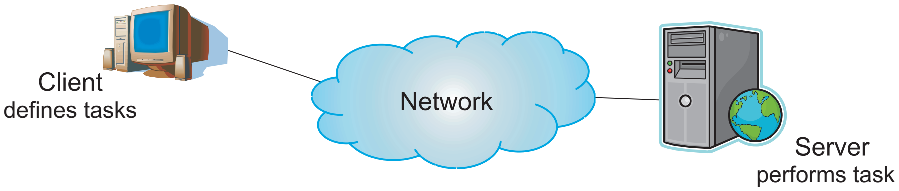
- We now know HTML, web pages, and how they can be “located” in the internet via URLs.
- But how does the concent of a web page come from the web server to our computer?
- We will learn about HTTP, the protocol existing for this purpose.
- How is HTTP related to TCP sockets and what we’ve learned so far?

Client/Server

- Client/Server systems are the most common application structure in the internet and corporate networks

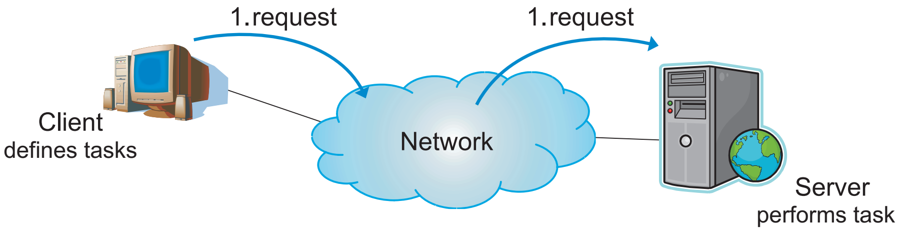
Client/Server

- Client/Server systems are the most common application structure in the internet and corporate networks



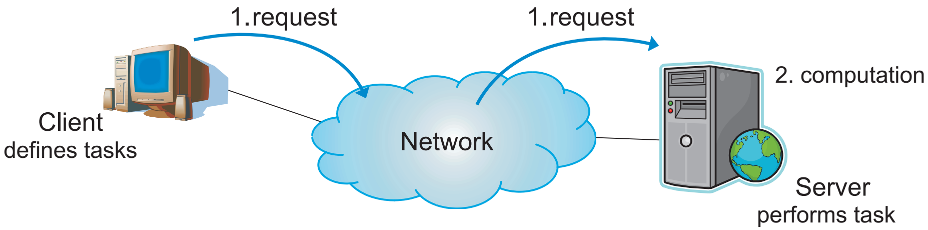
Client/Server

- Client/Server systems are the most common application structure in the internet and corporate networks



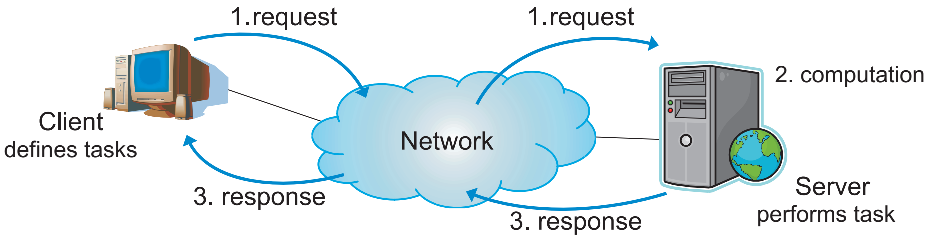
Client/Server

- Client/Server systems are the most common application structure in the internet and corporate networks

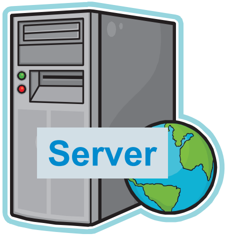


Client/Server

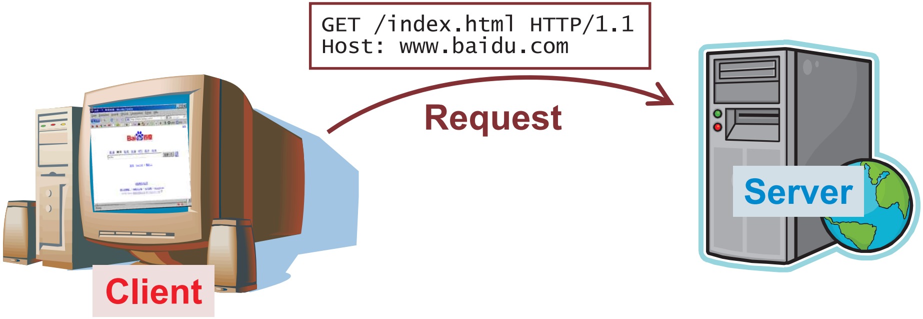
- Client/Server systems are the most common application structure in the internet and corporate networks



Browser/Web Server

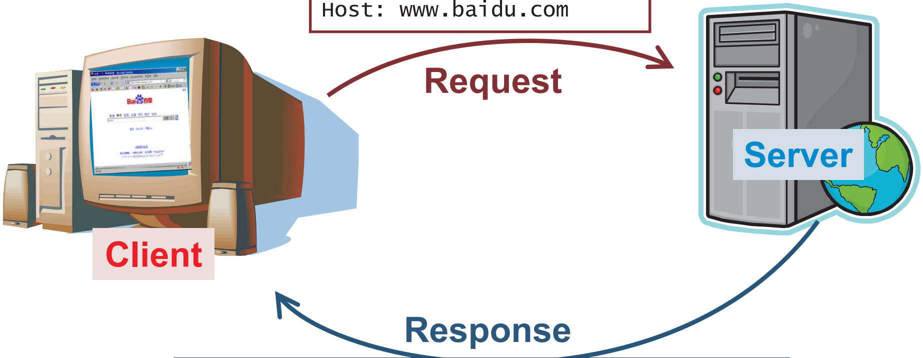


Browser/Web Server



```
GET /index.html HTTP/1.1
Host: www.baidu.com
```

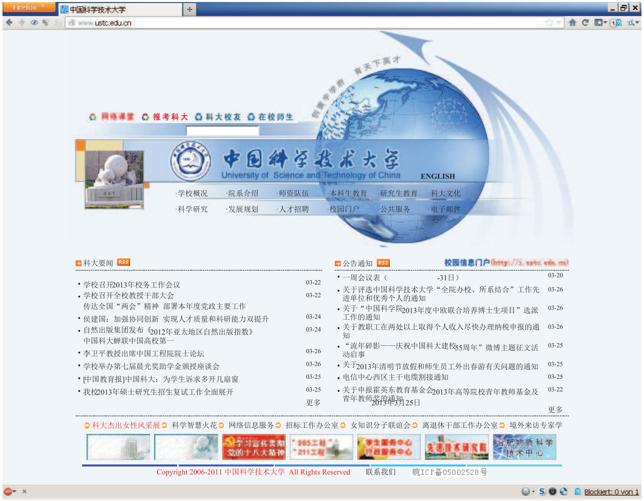
Browser/Web Server



```
GET /index.html HTTP/1.1
Host: www.baidu.com
```

```
HTTP/1.1 200 OK
...
<html><head><meta http-equiv=Content-Type
content="text/html;charset=gb2312"><title>百度
一下，你就知道</title><style>body{margin:4px...
```

Hypertext Structure



Firefox
中国科学技术大学
www.ustc.edu.cn
网络课堂 报考科大 科大校友 在校师生
中国科学技术大学
University of Science and Technology of China
ENGLISH
·学校概况 ·院系介绍 ·师资队伍 ·本科生教育 ·研究生教育 ·科大文化
·科学研究 ·发展规划 ·人才招聘 ·校园门户 ·公共服务 ·电子邮件
Blockiert: 0 von 1

科大要闻 RSS

- 学校召开2013年校务工作会议 03-22
- 学校召开全校教授干部大会 03-22
  传达全国“两会”精神 部署本年度党政主要工作
- 侯建国：加强协同创新 实现人才质量和科研能力双提升 03-24
- 自然出版集团发布《2012年亚太地区自然出版指数》 03-24
  中国科大蝉联中国高校第一
- 李卫平教授出席中国工程院院士论坛 03-26
- 学校举办第七届晨光奖助学金颁授座谈会 03-26
- [中国教育报]中国科大：为学生诉求多开几扇窗 03-25
- 我校2013年硕士研究生招生复试工作全面展开 03-25

更多

公告通知 RSS　校园信息门户

- 一周会议表（ -31日） 03-20
- 关于评选中国科学技术大学“全院办校、所系结合”工作先进单位和优秀个人的通知 03-26
- 关于“中国科学院2013年度中欧联合培养博士生项目”选派工作的通知 03-26
- 关于教职工在两处以上取得个人收入尽快办理纳税申报的通知 03-26
- “流年碎影——庆祝中国科大建校55周年”微博主题征文活动启事 03-25
- 关于2013年清明节放假和师生员工外出春游有关问题的通知 03-25
- 电信中心西区主干电缆割接通知 03-25
- 关于申报霍英东教育基金会2013年高等院校青年教师基金及青年教师奖的通知 03-22

2013年3月25日

更多

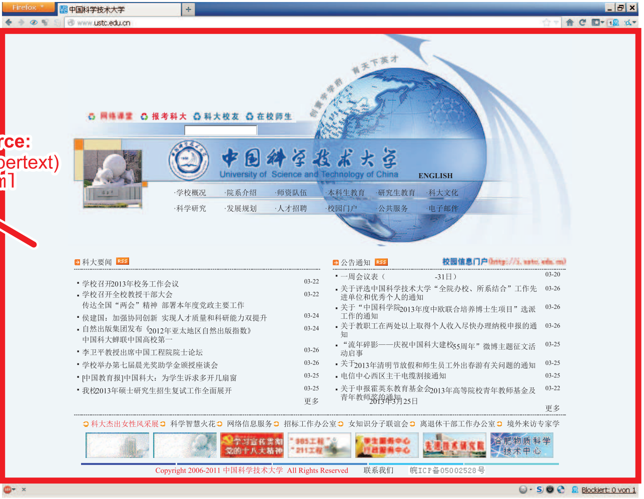
科大杰出女性风采展　科学智慧火花　网络信息服务　招标工作办公室　女知识分子联谊会　离退休干部工作办公室　境外来访专家学

学习宣传贯彻党的十八大精神　“985工程”“211工程”　学生服务中心 行政服务中心　先进技术研究院　合肥物质科学技术中心

Copyright 2006-2011 中国科学技术大学 All Rights Reserved　联系我们　皖ICP备05002528号

Hypertext Structure

Main Resource:
HTML Code (hypertext)
`index.html`



科大要闻 RSS

- 学校召开2013年校务工作会议 03-22
- 学校召开全校教授干部大会 传达全国“两会”精神 部署本年度党政主要工作 03-22
- 侯建国：加强协同创新 实现人才质量和科研能力双提升 03-24
- 自然出版集团发布《2012年亚太地区自然出版指数》 中国科大蝉联中国高校第一 03-24
- 李卫平教授出席中国工程院院士论坛 03-26
- 学校举办第七届晨光奖助学金颁授座谈会 03-26
- [中国教育报]中国科大：为学生诉求多开几扇窗 03-25
- 我校2013年硕士研究生招生复试工作全面展开 03-25

更多

公告通知 RSS 校园信息门户(http://i.ustc.edu.cn)

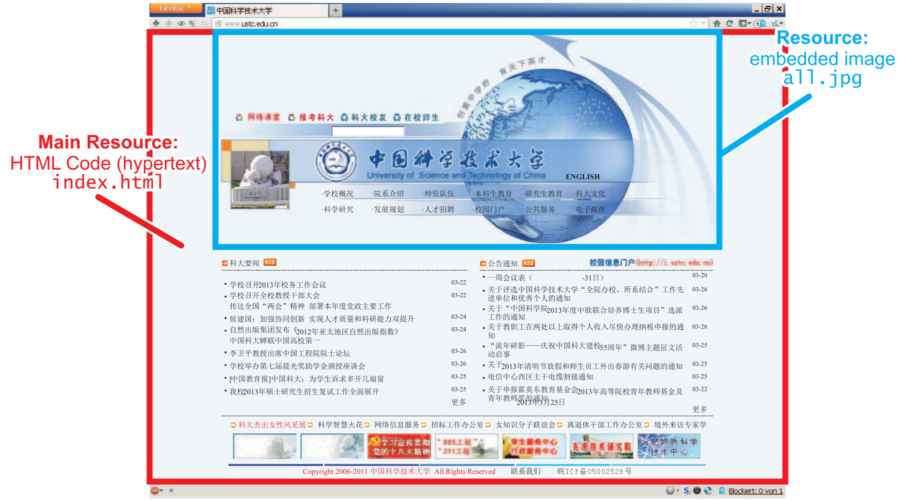
- 一周会议表（ -31日） 03-20
- 关于评选中国科学技术大学“全院办校、所系结合”工作先进单位和优秀个人的通知 03-26
- 关于“中国科学院2013年度中欧联合培养博士生项目”选派工作的通知 03-26
- 关于教职工在两处以上取得个人收入尽快办理纳税申报的通知 03-26
- “流年碎影——庆祝中国科大建校55周年”微博主题征文活动启事 03-25
- 关于2013年清明节放假和师生员工外出春游有关问题的通知 03-25
- 电信中心西区主干电缆割接通知 03-25
- 关于申报霍英东教育基金会2013年高等院校青年教师基金及青年教师奖的通知 03-22

2013年3月25日

更多

Copyright 2006-2011 中国科学技术大学 All Rights Reserved 联系我们 皖ICP备05002528号

Hypertext Structure



科大要闻 RSS

- 学校召开2013年校务工作会议 03-22
- 学校召开全校教授干部大会 03-22
  传达全国“两会”精神 部署本年度党政主要工作
- 侯建国：加强协同创新 实现人才质量和科研能力双提升 03-24
- 自然出版集团发布《2012年亚太地区自然出版指数》 03-24
  中国科大蝉联中国高校第一
- 李卫平教授出席中国工程院院士论坛 03-26
- 学校举办第七届晨光奖助学金颁授座谈会 03-26
- [中国教育报]中国科大：为学生诉求多开几扇窗 03-25
- 我校2013年硕士研究生招生复试工作全面展开 03-25

更多

公告通知 RSS 校园信息门户

- 一周会议表（ -31日） 03-20
- 关于评选中国科学技术大学“全院办校、所系结合”工作先进单位和优秀个人的通知 03-26
- 关于“中国科学院2013年度中欧联合培养博士生项目”选派工作的通知 03-26
- 关于教职工在两处以上取得个人收入尽快办理纳税申报的通知 03-26
- “流年碎影——庆祝中国科大建校55周年”微博主题征文活动启事 03-25
- 关于2013年清明节放假和师生员工外出春游有关问题的通知 03-25
- 电信中心西区主干电缆割接通知 03-25
- 关于申报霍英东教育基金会2013年高等院校青年教师基金及青年教师奖的通知 03-22

2013年3月25日

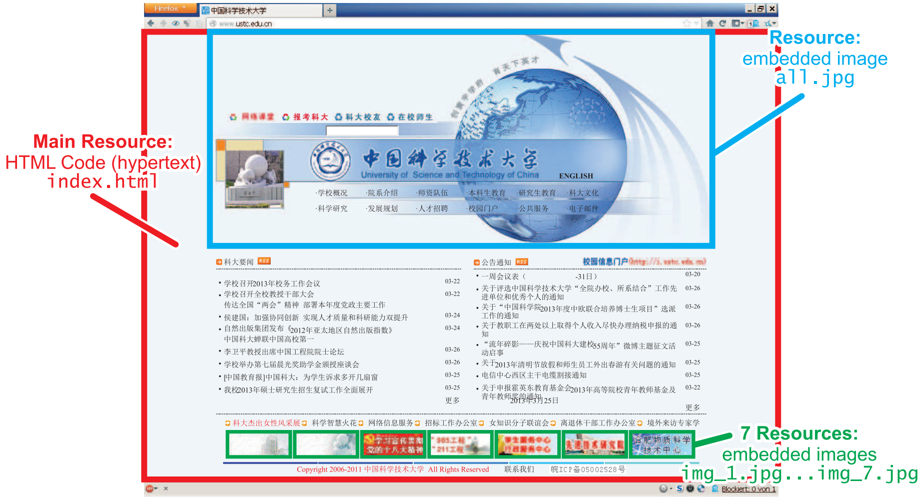
更多

科大杰出女性风采展 科学智慧火花 网络信息服务 招标工作办公室 女知识分子联谊会 离退休干部工作办公室 境外来访专家学

学习宣传贯彻党的十八大精神 “985工程” “211工程” 学生服务中心 行政服务中心 先进技术研究院 合肥物质科学技术中心

Copyright 2006-2011 中国科学技术大学 All Rights Reserved 联系我们 皖ICP备05002528号

Hypertext Structure



科大要闻 RSS

- 学校召开2013年校务工作会议 03-22
- 学校召开全校教授干部大会 传达全国“两会”精神 部署本年度党政主要工作 03-22
- 侯建国：加强协同创新 实现人才质量和科研能力双提升 03-24
- 自然出版集团发布《2012年亚太地区自然出版指数》 中国科大蝉联中国高校第一 03-24
- 李卫平教授出席中国工程院院士论坛 03-26
- 学校举办第七届晨光奖助学金颁授座谈会 03-26
- [中国教育报]中国科大：为学生诉求多开几扇窗 03-25
- 我校2013年硕士研究生招生复试工作全面展开 03-25

更多

公告通知 RSS 校园信息门户

- 一周会议表（ -31日） 03-20
- 关于评选中国科学技术大学“全院办校、所系结合”工作先进单位和优秀个人的通知 03-26
- 关于“中国科学院2013年度中欧联合培养博士生项目”选派工作的通知 03-26
- 关于教职工在两处以上取得个人收入尽快办理纳税申报的通知 03-26
- “流年碎影——庆祝中国科大建校55周年”微博主题征文活动启事 03-25
- 关于2013年清明节放假和师生员工外出春游有关问题的通知 03-25
- 电信中心西区主干电缆割接通知 03-25
- 关于申报霍英东教育基金会2013年高等院校青年教师基金及青年教师奖的通知 03-22

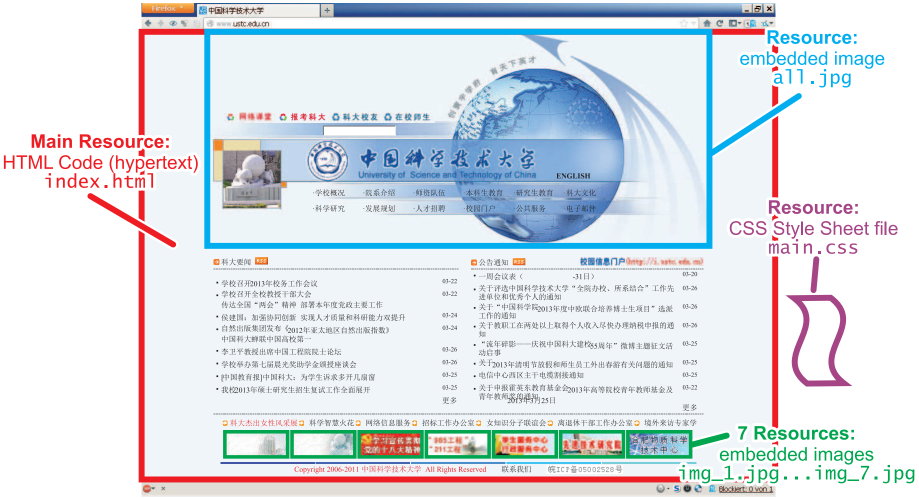
2013年3月25日

更多

科大杰出女性风采展 科学智慧火花 网络信息服务 招标工作办公室 女知识分子联谊会 离退休干部工作办公室 境外来访专家学

Copyright 2006-2011 中国科学技术大学 All Rights Reserved 联系我们 皖ICP备05002528号

Hypertext Structure



Main Resource:
HTML Code (hypertext)
index.html
Resource:
embedded image
all.jpg
Resource:
CSS Style Sheet file
main.css
7 Resources:
embedded images
img_1.jpg...img_7.jpg
Firefox
中国科学技术大学
www.ustc.edu.cn
网络课堂 报考科大 科大校友 在校师生
中国科学技术大学
University of Science and Technology of China
ENGLISH
·学校概况 ·院系介绍 ·师资队伍 ·本科生教育 ·研究生教育 ·科大文化
·科学研究 ·发展规划 ·人才招聘 ·校园门户 ·公共服务 ·电子邮件
科大要闻 RSS
公告通知 RSS
校园信息门户
更多
科大杰出女性风采展 科学智慧火花 网络信息服务 招标工作办公室 女知识分子联谊会 离退休干部工作办公室 境外来访专家学
Blockiert: 0 von 1

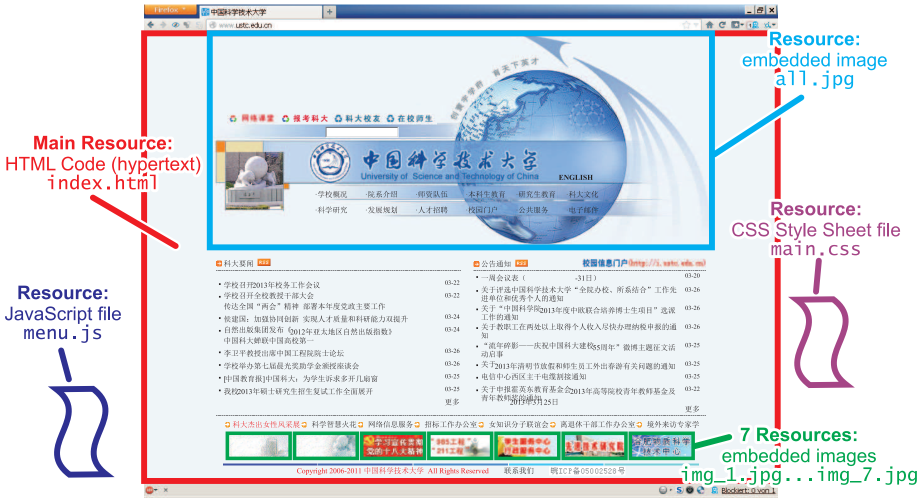
- 学校召开2013年校务工作会议 03-22
- 学校召开全校教授干部大会 传达全国“两会”精神 部署本年度党政主要工作 03-22
- 侯建国：加强协同创新 实现人才质量和科研能力双提升 03-24
- 自然出版集团发布《2012年亚太地区自然出版指数》中国科大蝉联中国高校第一 03-24
- 李卫平教授出席中国工程院院士论坛 03-26
- 学校举办第七届晨光奖助学金颁授座谈会 03-26
- [中国教育报]中国科大：为学生诉求多开几扇窗 03-25
- 我校2013年硕士研究生招生复试工作全面展开 03-25

- 一周会议表（ -31日） 03-20
- 关于评选中国科学技术大学“全院办校、所系结合”工作先进单位和优秀个人的通知 03-26
- 关于“中国科学院2013年度中欧联合培养博士生项目”选派工作的通知 03-26
- 关于教职工在两处以上取得个人收入尽快办理纳税申报的通知 03-26
- “流年碎影——庆祝中国科大建校55周年”微博主题征文活动启事 03-25
- 关于2013年清明节放假和师生员工外出春游有关问题的通知 03-25
- 电信中心西区主干电缆割接通知 03-25
- 关于申报霍英东教育基金会2013年高等院校青年教师基金及青年教师奖的通知 03-22

2013年3月25日

Copyright 2006-2011 中国科学技术大学 All Rights Reserved 联系我们 皖ICP备05002528号

Hypertext Structure



科大要闻 RSS

- 学校召开2013年校务工作会议 03-22
- 学校召开全校教授干部大会 03-22
  传达全国“两会”精神 部署本年度党政主要工作
- 侯建国：加强协同创新 实现人才质量和科研能力双提升 03-24
- 自然出版集团发布《2012年亚太地区自然出版指数》 03-24
  中国科大蝉联中国高校第一
- 李卫平教授出席中国工程院院士论坛 03-26
- 学校举办第七届晨光奖助学金颁授座谈会 03-26
- [中国教育报]中国科大：为学生诉求多开几扇窗 03-25
- 我校2013年硕士研究生招生复试工作全面展开 03-25

更多

公告通知 RSS 校园信息门户

- 一周会议表（ -31日） 03-20
- 关于评选中国科学技术大学“全院办校、所系结合”工作先进单位和优秀个人的通知 03-26
- 关于“中国科学院2013年度中欧联合培养博士生项目”选派工作的通知 03-26
- 关于教职工在两处以上取得个人收入尽快办理纳税申报的通知 03-26
- “流年碎影——庆祝中国科大建校55周年”微博主题征文活动启事 03-25
- 关于2013年清明节放假和师生员工外出春游有关问题的通知 03-25
- 电信中心西区主干电缆割接通知 03-25
- 关于申报霍英东教育基金会2013年高等院校青年教师基金及青年教师奖的通知 03-22

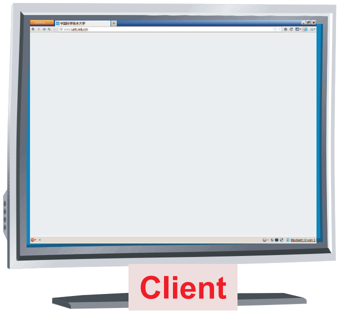
2013年3月25日

更多

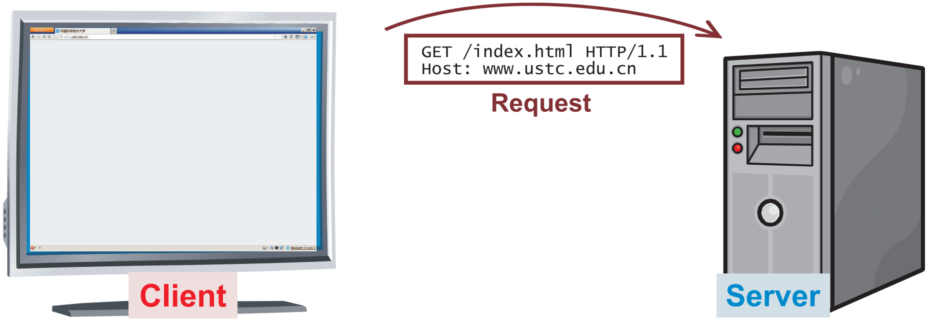
科大杰出女性风采展 科学智慧火花 网络信息服务 招标工作办公室 女知识分子联谊会 离退休干部工作办公室 境外来访专家学

Copyright 2006-2011 中国科学技术大学 All Rights Reserved 联系我们 皖ICP备05002528号

Browser loads Website

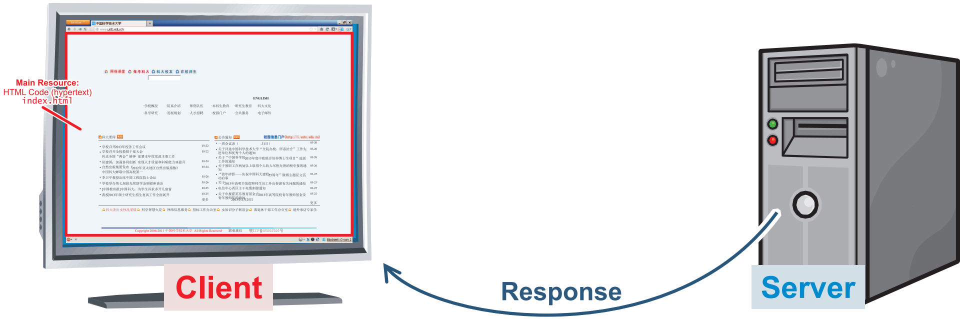


Browser loads Website



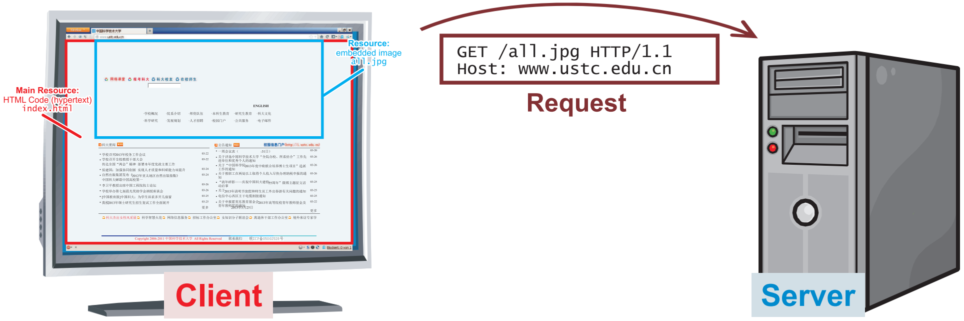
```
GET /index.html HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



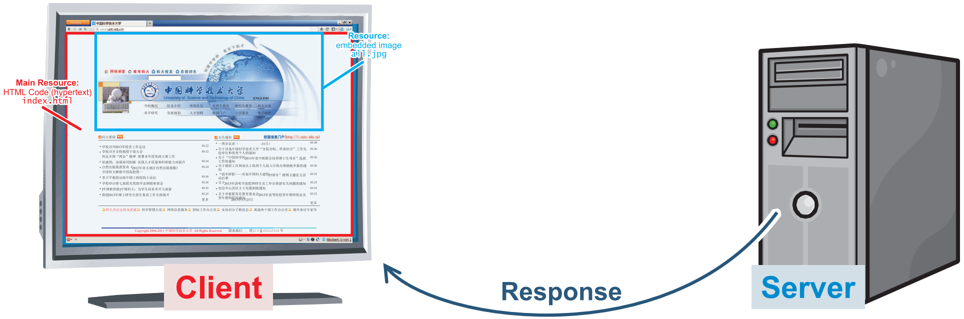
```
HTTP/1.1 200 OK
...
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML lang=zh-CN xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<link rel="shortcut icon" href="ustc.ico" type="image/x-icon">
<link rel="icon" href="ustc.ico" type="image/x-icon">
<META content="text/html; charset=utf-8" http-equiv=Content-Type>
<TITLE>中国科学技术大学</TITLE>
...
```

Browser loads Website



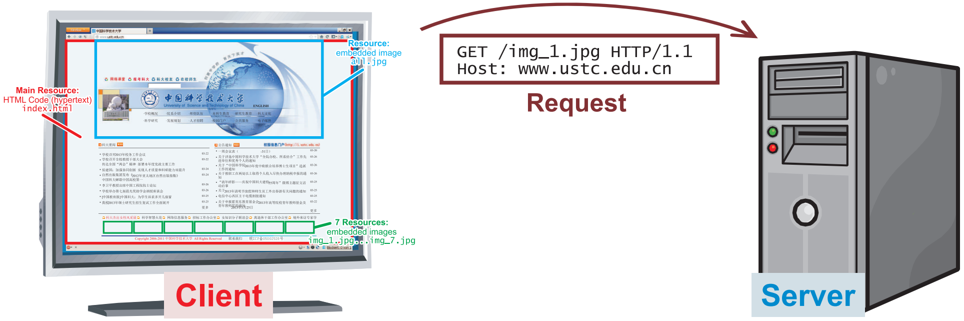
```
GET /all.jpg HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



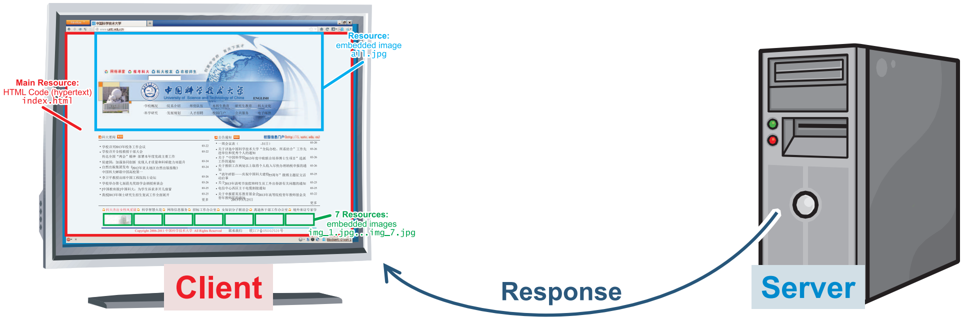
```
HTTP/1.1 200 OK
...
<binary data>
...
```

Browser loads Website



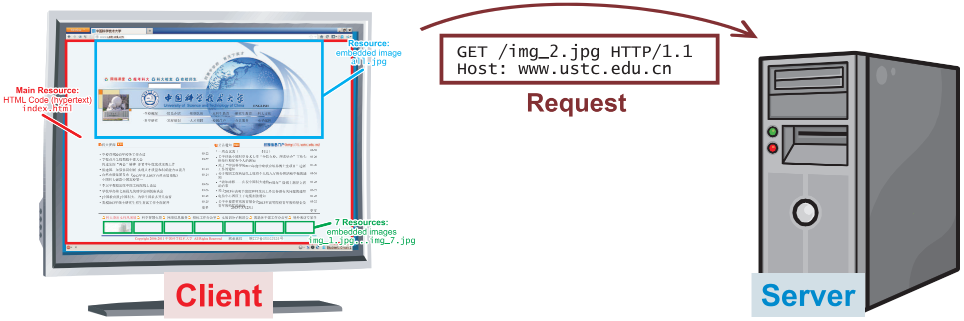
```
GET /img_1.jpg HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



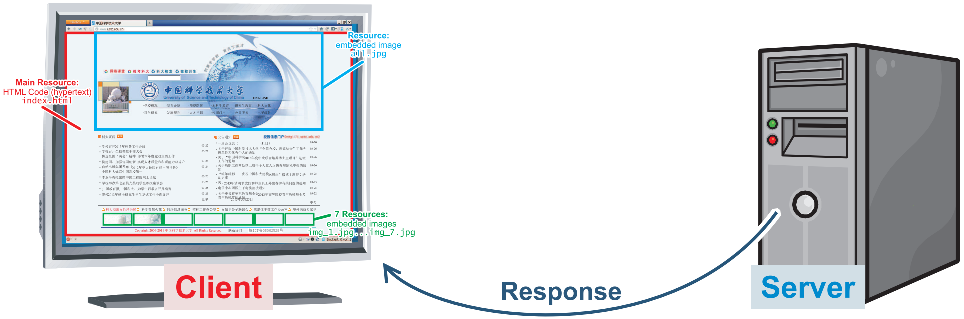
```
HTTP/1.1 200 OK
...
<binary data>
...
```

Browser loads Website



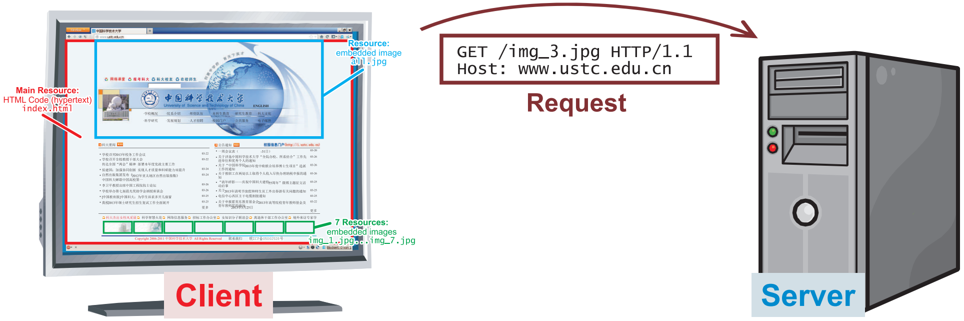
```
GET /img_2.jpg HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



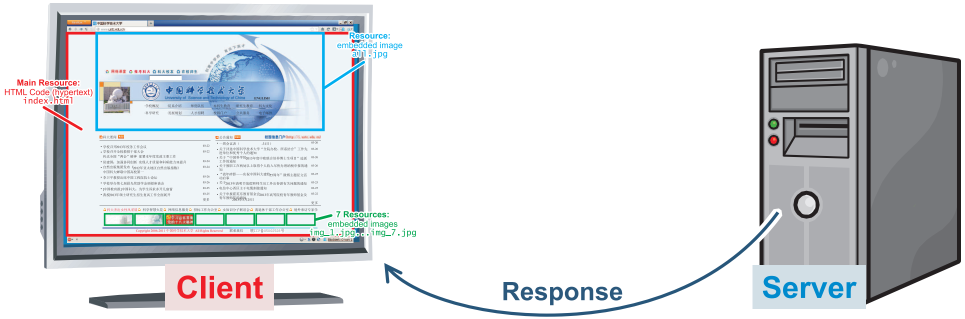
```
HTTP/1.1 200 OK
...
<binary data>
...
```

Browser loads Website



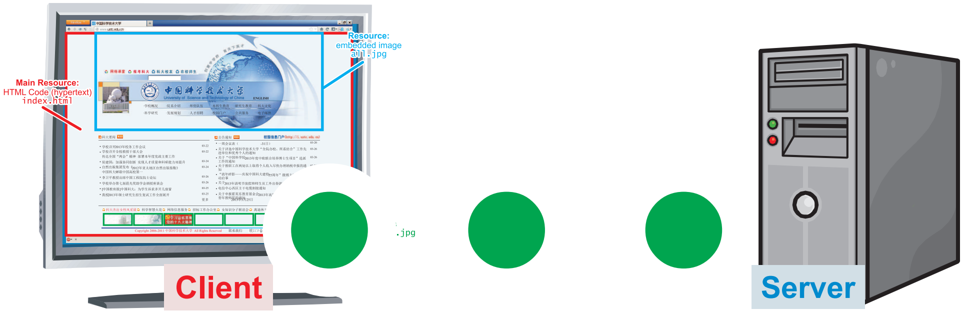
```
GET /img_3.jpg HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website

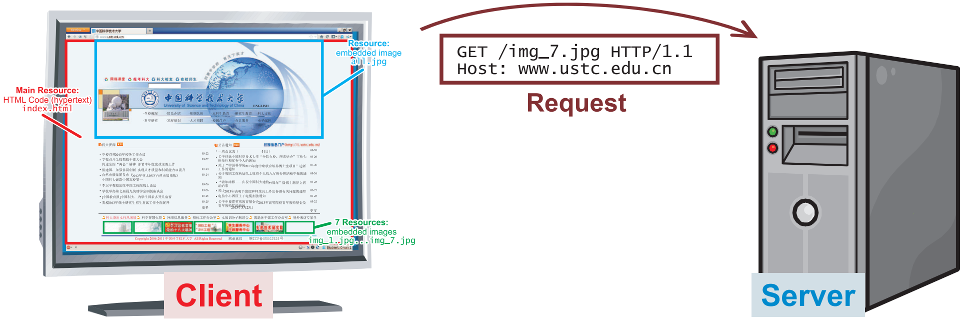


```
HTTP/1.1 200 OK
...
<binary data>
...
```

Browser loads Website

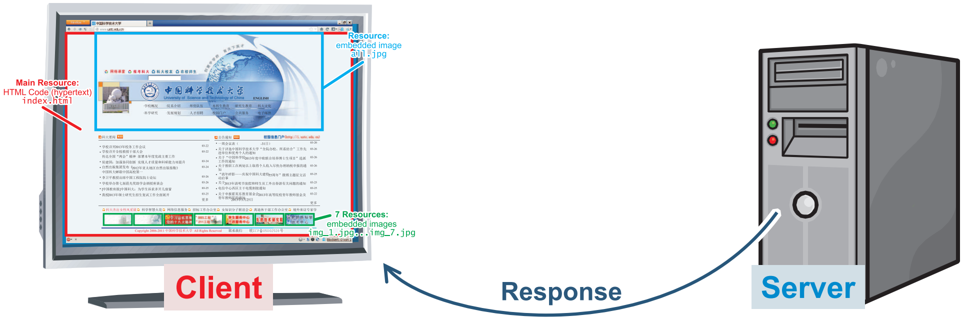


Browser loads Website



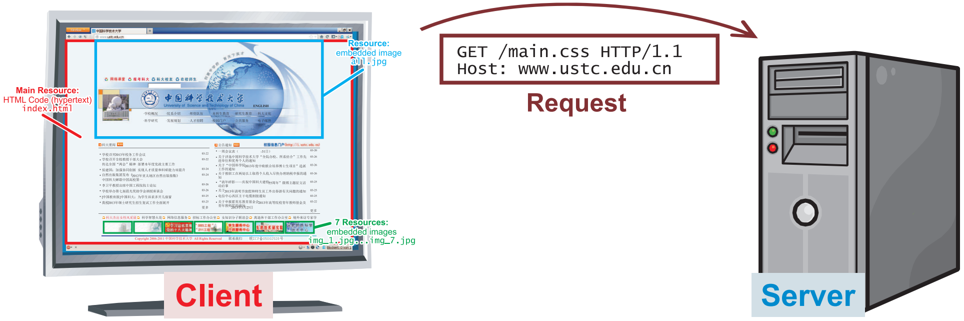
```
GET /img_7.jpg HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



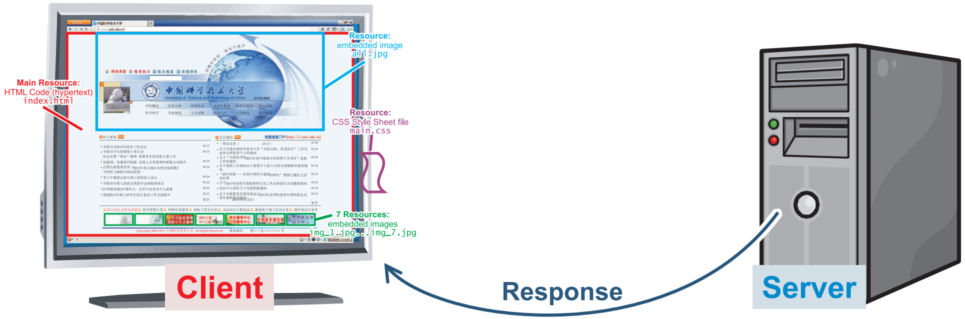
```
HTTP/1.1 200 OK
...
<binary data>
...
```

Browser loads Website



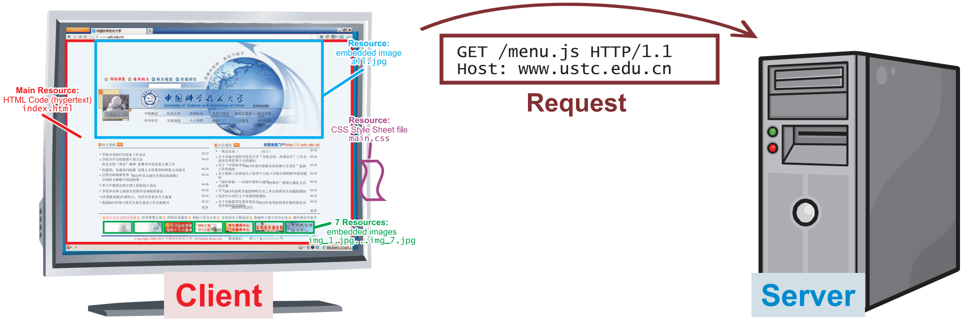
```
GET /main.css HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



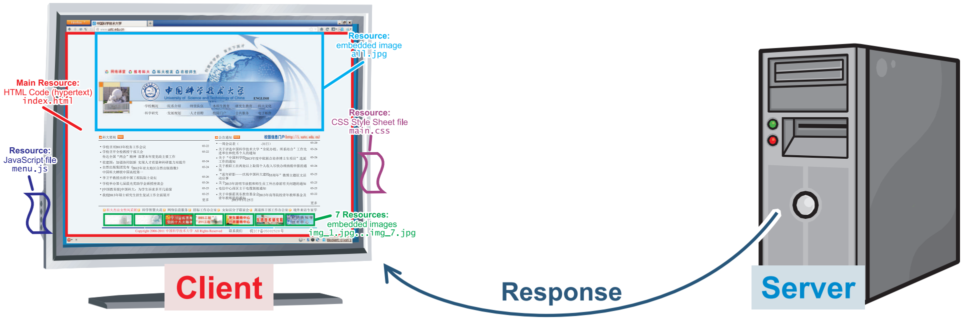
```
HTTP/1.1 200 OK
...
body{background-color:white;color:black;
font-family:sans-serif;}img{background-color
:transparent;border-style:none;}a{background
-color:transparent;border-style:none;text
-decoration:none;color:blue;}a:hover{text...
```

Browser loads Website



```
GET /menu.js HTTP/1.1
Host: www.ustc.edu.cn
```

Browser loads Website



```
HTTP/1.1 200 OK
...
function menuFix(menuid) {
var sfEls = document.getElementById(menuid).getElementsByTagName...
for (var i=0; i<sfEls.length; i++) {
  sfEls[i].onmouseover=function() {
  this.className+=(this.className.length>0? " ": "") + "sfhover";
  }
sfEls[i].onMouseDown=function() {
  this.className+=(this.className.length>0? " ": "") + "sfhover";
```

HTTP?

- But how do web browser and web server communicate?

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams
- But TCP is not enough

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams
- But TCP is not enough: The web browser must, at least, also send the `path/object` part of the URL

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams
- But TCP is not enough: The web browser must, at least, also send the `path/object` part of the URL, so that the web server knows *which* page or ressource the browser wants...

HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams
- But TCP is not enough: The web browser must, at least, also send the `path/object` part of the URL, so that the web server knows *which* page or ressource the browser wants... ...and the server must be able to send errors or other answers

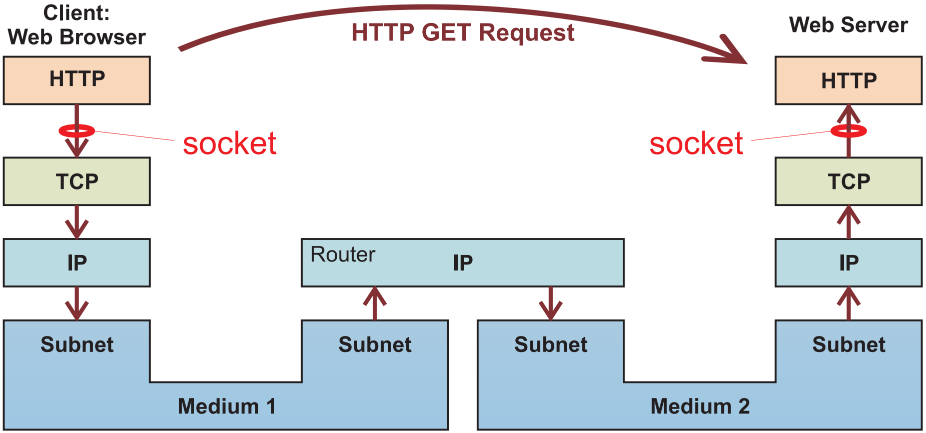
HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams
- But TCP is not enough: The web browser must, at least, also send the `path/object` part of the URL, so that the web server knows *which* page or ressource the browser wants. . . . . . and the server must be able to send errors or other answers
- Protocol on top of TCP, designed to achieve this

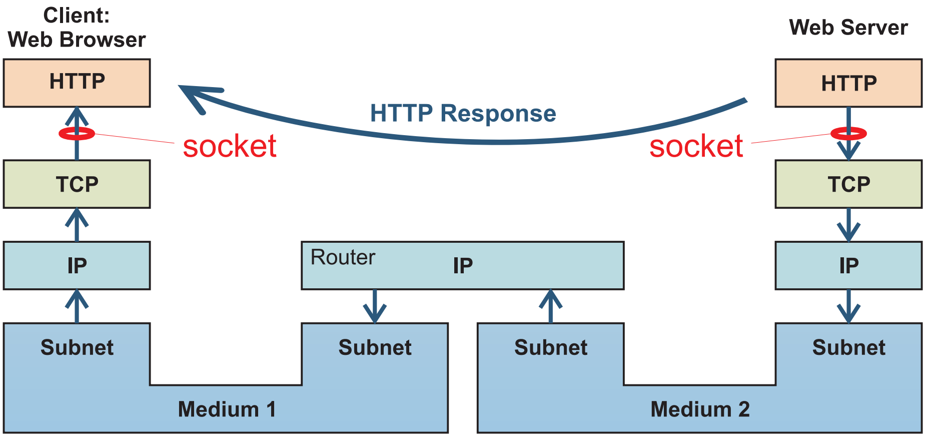
HTTP?

- But how do web browser and web server communicate?
- Well, we already know the Socket API which gives us access to transport layer protocols such as TCP and UDP
- Using TCP seems to be a good idea: We transfer files, and files can be characterized as streams
- But TCP is not enough: The web browser must, at least, also send the `path/object` part of the URL, so that the web server knows *which* page or ressource the browser wants... ...and the server must be able to send errors or other answers
- Protocol on top of TCP, designed to achieve this: HTTP

HTTP & Protocol Layers



HTTP & Protocol Layers



HTTP Protocol

- HyperText Transfer Protocol (HTTP)

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
    - text-based and human readable

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
  - text-based and human readable
  - state-less (no information preserved between different HTTP requests)

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
    - text-based and human readable
    - state-less (no information preserved between different HTTP requests)
    - anonymous

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
  - text-based and human readable
  - state-less (no information preserved between different HTTP requests)
  - anonymous
- HTTP 1.0 [1]

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
  - text-based and human readable
  - state-less (no information preserved between different HTTP requests)
  - anonymous
- HTTP 1.0 [1]:
  - Messages in the MIME (Multipurpose Internet Mail Extensions) format

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
    - text-based and human readable
    - state-less (no information preserved between different HTTP requests)
    - anonymous
- HTTP 1.0 [1]:
    - Messages in the MIME (Multipurpose Internet Mail Extensions) format
- HTTP 1.1 [2]

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
  - text-based and human readable
  - state-less (no information preserved between different HTTP requests)
  - anonymous
- HTTP 1.0 [1]:
  - Messages in the MIME (Multipurpose Internet Mail Extensions) format
- HTTP 1.1 [2]:
  - Support for hierarchical proxies, caching, persistent connections, virtual hosts

HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
  - text-based and human readable
  - state-less (no information preserved between different HTTP requests)
  - anonymous
- HTTP 1.0 [1]:
  - Messages in the MIME (Multipurpose Internet Mail Extensions) format
- HTTP 1.1 [2]:
  - Support for hierarchical proxies, caching, persistent connections, virtual hosts
  - Determine applications' capabilities

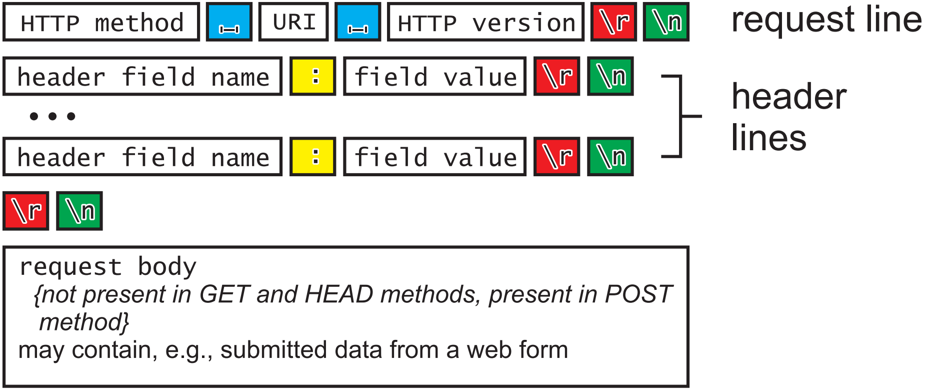
HTTP Protocol

- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
    - text-based and human readable
    - state-less (no information preserved between different HTTP requests)
    - anonymous
- HTTP 1.0 [1]:
    - Messages in the MIME (Multipurpose Internet Mail Extensions) format
- HTTP 1.1 [2]:
    - Support for hierarchical proxies, caching, persistent connections, virtual hosts
    - Determine applications' capabilities
    - TCP connections can be re-used

HTTP Protocol

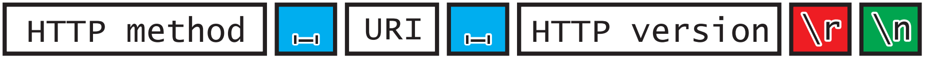
- HyperText Transfer Protocol (HTTP)
- Application-layer protocol for communication between web browser and web server
- Uses reliable, bi-directional, ordered byte stream provided by TCP
- Features:
  - text-based and human readable
  - state-less (no information preserved between different HTTP requests)
  - anonymous
- HTTP 1.0 [1]:
  - Messages in the MIME (Multipurpose Internet Mail Extensions) format
- HTTP 1.1 [2]:
  - Support for hierarchical proxies, caching, persistent connections, virtual hosts
  - Determine applications' capabilities
  - TCP connections can be re-used (good for inline resources/images in HTML)

HTTP Request



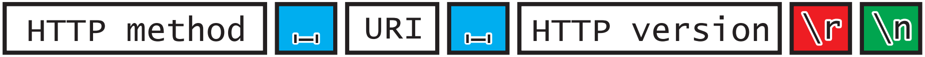
request body
{not present in GET and HEAD methods, present in POST method}
may contain, e.g., submitted data from a web form

HTTP Request: Request Line



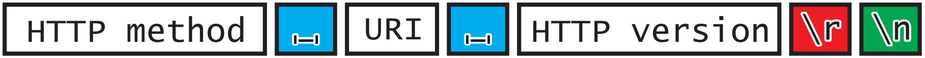
- `HTTP method`

HTTP Request: Request Line



- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page

HTTP Request: Request Line



- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)

HTTP Request: Request Line

`HTTP method` `␣` `URI` `␣` `HTTP version` `\r` `\n`

- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)
  - `POST`: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file

HTTP Request: Request Line

```
HTTP method ␣ URI ␣ HTTP version \r \n
```

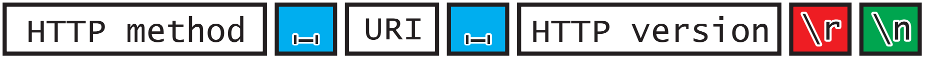
- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)
  - `POST`: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT`: upload a representation of the specified resource

HTTP Request: Request Line

`HTTP method` `␣` `URI` `␣` `HTTP version` `\r` `\n`

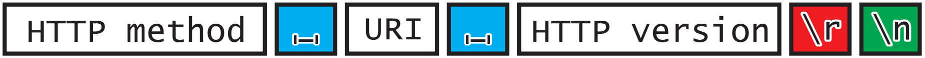
- `HTTP method`:
  - `GET` : download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD` : returns only the headers of a `GET` response (no request body)
  - `POST` : data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT` : upload a representation of the specified resource
  - `DELETE` : delete the specified resource

HTTP Request: Request Line



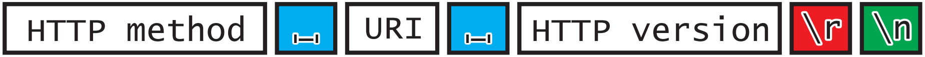
- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)
  - `POST`: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT`: upload a representation of the specified resource
  - `DELETE`: delete the specified resource
- relative part of `URI`

HTTP Request: Request Line



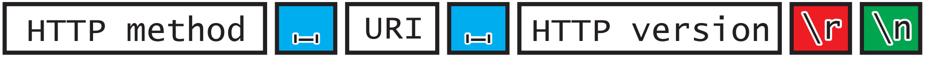
- `HTTP method`:
  - `GET` : download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD` : returns only the headers of a `GET` response (no request body)
  - `POST` : data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT` : upload a representation of the specified resource
  - `DELETE` : delete the specified resource
- relative part of `URI`: (contains, e.g., `path/object`)

HTTP Request: Request Line



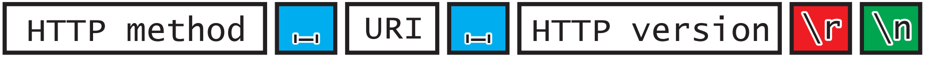
- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)
  - `POST`: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT`: upload a representation of the specified resource
  - `DELETE`: delete the specified resource
- relative part of `URI`: (contains, e.g., `path/object`) and identifies a resource relative to server

HTTP Request: Request Line



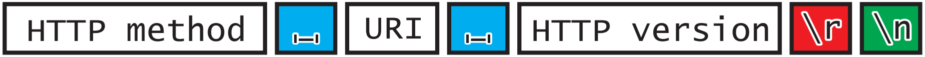
- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)
  - `POST`: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT`: upload a representation of the specified resource
  - `DELETE`: delete the specified resource
- relative part of `URI`: (contains, e.g., `path/object`) and identifies a resource relative to server
- `HTTP version`

HTTP Request: Request Line



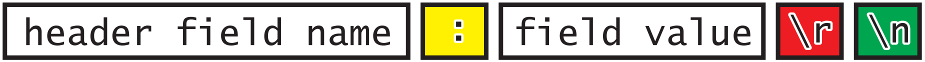
- `HTTP method`:
  - `GET`: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - `HEAD`: returns only the headers of a `GET` response (no request body)
  - `POST`: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - `PUT`: upload a representation of the specified resource
  - `DELETE`: delete the specified resource
- relative part of `URI`: (contains, e.g., `path/object`) and identifies a resource relative to server
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]

HTTP Request: Request Line



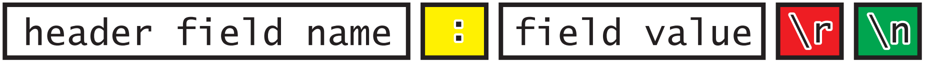
- HTTP method:
  - GET: download the data that belongs to a specified resource (can include parameters), e.g., a web page
  - HEAD: returns only the headers of a GET response (no request body)
  - POST: data is sent in the request body that should be handed to the requested resource, maybe the data filled into a form or a file
  - PUT: upload a representation of the specified resource
  - DELETE: delete the specified resource
- relative part of URI: identifies a resource relative to server
- HTTP version:
  - HTTP/1.1 for HTTP 1.1 [2]
  - HTTP/1.0 for HTTP 1.0 [1]

HTTP Request: Request Header



- Different header fields, each has its own format

HTTP Request: Request Header



- Different header fields, each has its own format
- Examples

HTTP Request: Request Header

`header field name`	`:`	`field value`	`\r`	`\n`

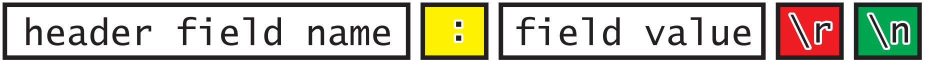
- Different header fields, each has its own format
- Examples
  - `Accept: text/html`

HTTP Request: Request Header

```
header field name : field value \r \n
```

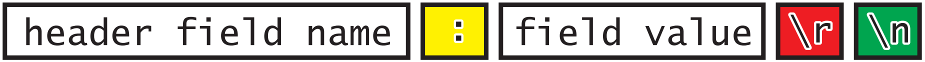
- Different header fields, each has its own format
- Examples
  - `Accept: text/html`
  - `Accept: image/jpg`

HTTP Request: Request Header

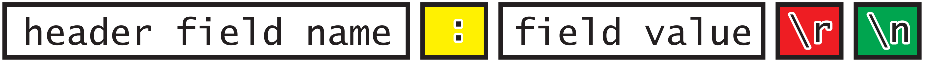


- Different header fields, each has its own format
- Examples
    - `Accept: text/html`
    - `Accept: image/jpg`
    - `Accept-Language: zh-CN`

HTTP Request: Request Header



- Different header fields, each has its own format
- Examples
  - `Accept: text/html`
  - `Accept: image/jpg`
  - `Accept-Language: zh-CN`
  - `Accept-Charset: utf-8, gb2312`



- Different header fields, each has its own format
- Examples
  - `Accept: text/html`
  - `Accept: image/jpg`
  - `Accept-Language: zh-CN`
  - `Accept-Charset: utf-8, gb2312`
  - `If-Modified-Since: Wed, 27 Mar 2013 12:01:32 GMT`

HTTP Request Printing Server

- Let's look how requests generated by web browsers look like

HTTP Server Printing Requests

Listing: HTTPServerPrintingRequests.java

```java
import java.io.BufferedReader;          import java.io.BufferedWriter;      import java.io.InputStreamReader;
import java.io.OutputStreamWriter;      import java.net.ServerSocket;       import java.net.Socket;

public class HTTPServerPrintingRequests {
  public static final void main(final String[] args) {
    String s;     StringBuilder sb;

    try (ServerSocket server = new ServerSocket(9995)) { //create server socket
      try (Socket client = server.accept()) { //accept incoming client

        sb = new StringBuilder(); //allocate buffer

        try (InputStreamReader ir = new InputStreamReader(client.getInputStream());//request=character stream
             BufferedReader    br = new BufferedReader(ir)) { //read request line-by-line
          while ((s = br.readLine()) != null) { //as long as lines can be read...
            sb.append(s);                       //append them to the buffer
            sb.append("<br/>");                 //add HTML line breaks
            if(s.length()<=0) { break; } // the final newline of the header
          }
          client.shutdownInput(); //no more input is requests

          try (OutputStreamWriter pw = new OutputStreamWriter(client.getOutputStream())) {
            pw.write("HTTP/1.1␣200␣OK\r\n\r\n<html><body><pre>"); //now write the answer: HTTP OK + HTML document
            pw.write(sb.toString()); //buffered content
            pw.write("</pre></body></html>"); //close the HTML document
          }
        }
      }
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

HTTP Request: Example

We run the `HTTPServerPrintingRequests` locally and access `localhost:9995` with Firefox

HTTP Request: Example

We run the `HTTPServerPrintingRequests` locally and access `localhost:9995` with Firefox:

```
GET / HTTP/1.1
Host: localhost:9995
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.8,de-de;q=0.5,de;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cache-Control: max-age=0
```

HTTP Request: Example

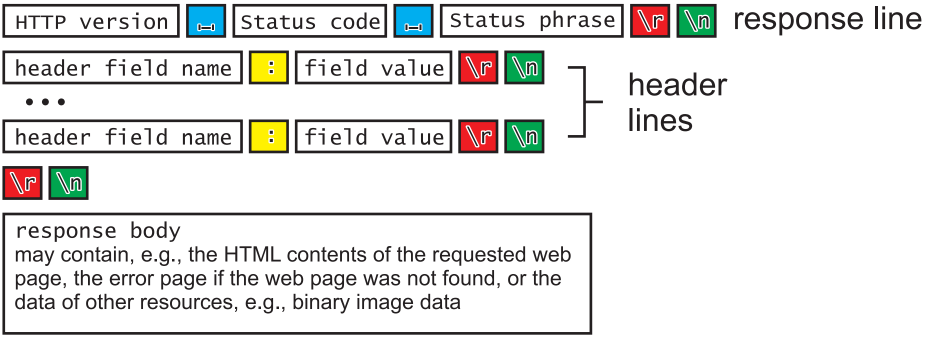
We run the `HTTPServerPrintingRequests` locally and access `http://localhost:9995` with Internet Explorer

HTTP Request: Example

We run the HTTPServerPrintingRequests locally and access http://localhost:9995 with Internet Explorer:

```
GET / HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml, image/gif,...
Accept-Language: de-DE
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0;;...
Accept-Encoding: gzip, deflate
Host: localhost:9995
Connection: Keep-Alive
```

HTTP Response



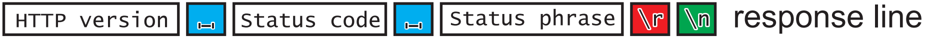
HTTP version
␣
Status code
␣
Status phrase
\r
\n
response line
header field name
:
field value
\r
\n
...
header field name
:
field value
\r
\n
header lines
\r
\n
response body
may contain, e.g., the HTML contents of the requested web page, the error page if the web page was not found, or the data of other resources, e.g., binary image data

HTTP Response: Response Line

`HTTP version` ␣ `Status code` ␣ `Status phrase` `\r` `\n` response line

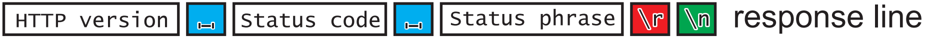
- `HTTP version`

HTTP Response: Response Line



- `HTTP version`:
    - `HTTP/1.1` for HTTP 1.1 [2]

HTTP Response: Response Line



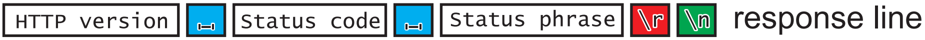
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]

HTTP Response: Response Line

`HTTP version` `␣` `Status code` `␣` `Status phrase` `\r` `\n` response line

- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`

HTTP Response: Response Line



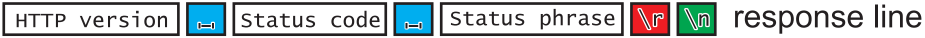
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing

HTTP Response: Response Line

`HTTP version` `␣` `Status code` `␣` `Status phrase` `\r` `\n` response line

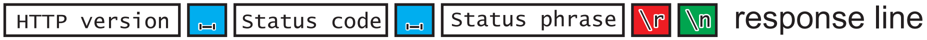
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing

HTTP Response: Response Line



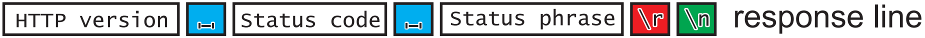
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1 Information

HTTP Response: Response Line



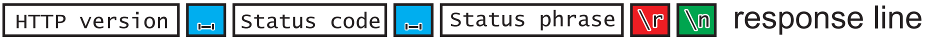
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1. Information
  2. Success

HTTP Response: Response Line



- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1. Information
  2. Success
  3. Redirection

HTTP Response: Response Line



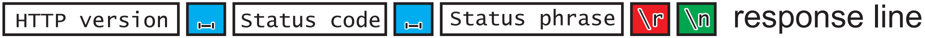
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1. Information
  2. Success
  3. Redirection
  4. Error on client side (e.g., wrong URI)

HTTP Response: Response Line

`HTTP version` `␣` `Status code` `␣` `Status phrase` `\r` `\n` response line

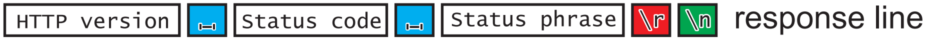
- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1. Information
  2. Success
  3. Redirection
  4. Error on client side (e.g., wrong URI)
  5. Error on server side

HTTP Response: Response Line



- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1. Information
  2. Success
  3. Redirection
  4. Error on client side (e.g., wrong URI)
  5. Error on server side
- `Status phrase`

HTTP Response: Response Line



- `HTTP version`:
  - `HTTP/1.1` for HTTP 1.1 [2]
  - `HTTP/1.0` for HTTP 1.0 [1]
- `Status code`: three digits, with the first digit representing
  1. Information
  2. Success
  3. Redirection
  4. Error on client side (e.g., wrong URI)
  5. Error on server side
- `Status phrase`: short textual representation of status code, e.g., `OK`

HTTP Status Codes

- Some examples for status codes

HTTP Status Codes

- Some examples for status codes
    - 200 OK

HTTP Status Codes

- Some examples for status codes
  - 200 OK
  - 301 Moved Permanently

HTTP Status Codes

- Some examples for status codes
  - 200 OK
  - 301 Moved Permanently
  - 400 Bad Request

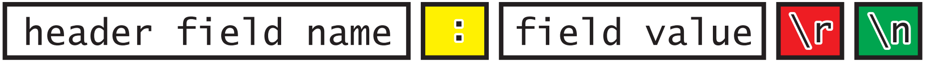
HTTP Status Codes

- Some examples for status codes
  - 200 OK
  - 301 Moved Permanently
  - 400 Bad Request
  - 404 Not Found

HTTP Status Codes

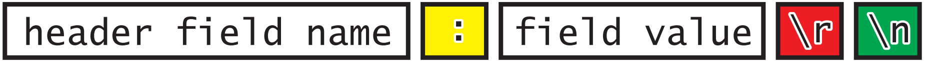
- Some examples for status codes
  - 200 OK
  - 301 Moved Permanently
  - 400 Bad Request
  - 404 Not Found
  - 505 HTTP Version Not Supported

HTTP Response: Response Header



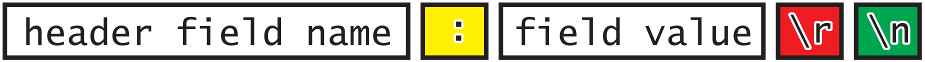
- Different header fields, each has its own format

HTTP Response: Response Header



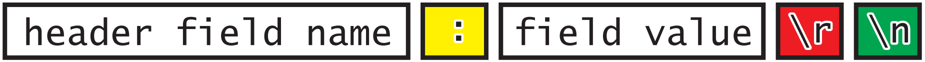
- Different header fields, each has its own format
- Examples

HTTP Response: Response Header



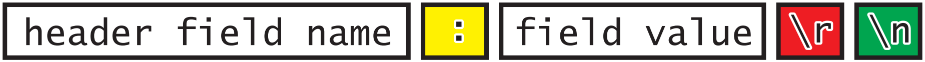
- Different header fields, each has its own format
- Examples
    - `Content-Type: text/html`

HTTP Response: Response Header



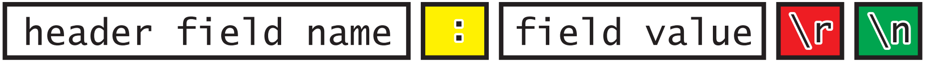
- Different header fields, each has its own format
- Examples
  - `Content-Type: text/html`
  - `Content-Length: 16384`

HTTP Response: Response Header



- Different header fields, each has its own format
- Examples
  - `Content-Type: text/html`
  - `Content-Length: 16384`
  - `Language: zh-CN;`

HTTP Response: Response Header



- Different header fields, each has its own format
- Examples
  - `Content-Type: text/html`
  - `Content-Length: 16384`
  - `Language: zh-CN;`
  - `Last-modified: 28 Mar 2013`

HTTP Response Printing Client

- Let's look how responses generated by web servers look like

HTTP Client Printing Server's Response

Listing: MinHTTPClientJava17.java Min HTTP Client + Try-With-Resource

```java
import java.io.BufferedReader;          import java.io.InputStreamReader;
import java.io.OutputStreamWriter;      import java.net.Socket;

public class MinHTTPClientJava17 {//this is a minimum web client; see lesson 07 coming later
  public static final void main(final String[] args) {
    String dest, request, response;

    dest    = "www.baidu.com";      // a random example for a Chinese host
    request = "GET␣/index.html␣HTTP/1.1\nHost:␣" + dest + "\n\n\n";

    try(Socket sock = new Socket(dest, 80)) { // web servers are usually listening at port 80
      try(OutputStreamWriter w = new OutputStreamWriter(sock.getOutputStream())) {
      w.write(request);             // write the HTTP request [1–3]
      w.flush();                    // make sure that all data has been sent
      sock.shutdownOutput();        // closing down the channel for sending data to the server

      try (InputStreamReader is = new InputStreamReader(sock.getInputStream());
          BufferedReader     r  = new BufferedReader(is)) { // Baidu uses UTF-8 encoding
          while ((response = r.readLine()) != null) { // read strings line-by-line until connection closed by serve
            System.out.println(response);             // print to output
          }
        }
      }
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

HTTP Response: Example

To `www.baidu.com`, we send

HTTP Response: Example

To `www.baidu.com`, we send:

```
GET /index.html HTTP/1.1
Host: www.baidu.com
```

HTTP Response: Example

To `www.baidu.com`, we send:

```
GET /index.html HTTP/1.1
Host: www.baidu.com
```

and get the response:

```
HTTP/1.1 200 OK
Date: Wed, 27 Mar 2013 23:44:43 GMT
Server: BWS/1.0
Content-Length: 10319
Content-Type: text/html;charset=utf-8
Cache-Control: private
Expires: Wed, 27 Mar 2013 23:44:43 GMT
Set-Cookie: H_PS_PSSID=2097_1430_2132_1945_1788; path=/; domain=.baidu.com
Set-Cookie: BAIDUID=1BBB7C987D5159BE0741B675A88B3E0C:FG=1; expires=Wed, 27-Mar-43...
P3P: CP=" OTI DSP COR IVA OUR IND COM "
Connection: Keep-Alive

<!DOCTYPE html><!--STATUS OK-->    <html><head>  <meta http-equiv="content-type"
content="text/html;charset=utf-8">  <title>...
```

Summary

- HTTP is a general, text-based protocol to request resources.
- Web pages are served by web servers which implement the HTTP protocol.
- Such servers can be implemented with the simple stuff we have learned about sockets.
- Actually, several Java-based web servers use exactly the technologies we have learned so far.

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui, China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Timothy John Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*, volume 1945 of *Request for Comments (RFC)*. Network Working Group, May 1996. URL `http://tools.ietf.org/html/rfc1945`.
2. R. Fielding, J. Gettys, Jeffrey Mogul, H. Frystyk, L. Masinter, P. Leach, and Timothy John Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, volume 2616 of *Request for Comments (RFC)*. Network Working Group, June 1999. URL `http://tools.ietf.org/html/rfc2616`.
3. David Gourley and Brian Totty. *HTTP: The Definitive Guide*. Definitive Guide. Sebastopol, CA, USA: O'Reilly Media, Inc., 2002. ISBN 1565925092 and 9781565925090. URL `http://books.google.de/books?id=qEoOl9bcV_cC`.