

-

整理自互联网，仅供学习交流。



2018-10-30

[公司名称]

[公司地址]

目录

123

**深入理解Http协议**

# Http协议入门

## 什么是http协议

http协议： 对浏览器客户端 和 服务器端 之间数据传输的格式规范

# 查看http协议的工具

1）使用火狐的firebug插件（右键->firebug->网络）

2）使用谷歌的“审查元素”

## 2.1 http协议内容

|  |
| --- |
| 请求（浏览器-》服务器）  GET /day09/hello HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Connection: keep-alive |

|  |
| --- |
| 响应（服务器-》浏览器）  HTTP/1.1 200 OK  Server: Apache-Coyote/1.1  Content-Length: 24  Date: Fri, 30 Jan 2015 01:54:57 GMT  this is hello servlet!!! |

# Http请求

|  |
| --- |
| GET /day09/hello HTTP/1.1 -请求行  Host: localhost:8080 --请求头（多个key-value对象）  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Connection: keep-alive  --一个空行  name=eric&password=123456 --（可选）实体内容 |

## 3.1 请求行

GET /day09/hello HTTP/1.1

### #http协议版本

http1.0：当前浏览器客户端与服务器端建立连接之后，只能发送一次请求，一次请求之后连接关闭。

http1.1：当前浏览器客户端与服务器端建立连接之后，可以在一次连接中发送多次请求。（基本都使用1.1）

### #请求资源

URL: 统一资源定位符。http://localhost:8080/day09/testImg.html。只能定位互联网资源。是URI 的子集。

URI： 统一资源标记符。/day09/hello。用于标记任何资源。可以是本地文件系统，局域网的资源（//192.168.14.10/myweb/index.html）， 可以是互联网。

### #请求方式

常见的请求方式： GET 、 POST、 HEAD、 TRACE、 PUT、 CONNECT 、DELETE

常用的请求方式： GET 和 POST

表单提交：

<form action="提交地址" method="GET/POST">

<form>

GET vs POST 区别

**1）GET方式提交**

a）地址栏（URI）会跟上参数数据。以？开头，多个参数之间以&分割。

|  |
| --- |
| **GET** /day09/testMethod.html?name=eric&password=123456 HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive |

b）GET提交参数数据有限制，不超过1KB。

c）GET方式不适合提交敏感密码。

d）注意： 浏览器直接访问的请求，默认提交方式是GET方式

**2）POST方式提交**

a）参数不会跟着URI后面。参数而是跟在请求的实体内容中。没有？开头，多个参数之间以&分割。

|  |
| --- |
| **POST** /day09/testMethod.html HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive  name=eric&password=123456 |

b）POST提交的参数数据没有限制。

c）POST方式提交敏感数据。

## 3.2 请求头

|  |
| --- |
| Accept: text/html,image/\* -- 浏览器接受的数据类型  Accept-Charset: ISO-8859-1 -- 浏览器接受的编码格式  Accept-Encoding: gzip,compress --浏览器接受的数据压缩格式  Accept-Language: en-us,zh- --浏览器接受的语言  Host: www.it315.org:80 --（必须的）当前请求访问的目标地址（主机:端口）  If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT --浏览器最后的缓存时间  Referer: http://www.it315.org/index.jsp -- 当前请求来自于哪里  User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0) --浏览器类型  Cookie:name=eric -- 浏览器保存的cookie信息  Connection: close/Keep-Alive -- 浏览器跟服务器连接状态。close: 连接关闭 keep-alive：保存连接。  Date: Tue, 11 Jul 2000 18:23:51 GMT -- 请求发出的时间 |

## 3.3 实体内容

只有POST提交的参数会放到实体内容中

## 3.4 HttpServletRequest对象

HttpServletRequest对象作用是用于获取请求数据。

核心的API：

请求行：

request.getMethod(); 请求方式

request.getRequetURI() / request.getRequetURL() 请求资源

request.getProtocol() 请求http协议版本

请求头：

request.getHeader("名称") 根据请求头获取请求值

request.getHeaderNames() 获取所有的请求头名称

实体内容:

request.getInputStream() 获取实体内容数据

## 3.5 service 和 doXX方法区别

|  |
| --- |
| **HttpSevlet类的源码：**  **protected** **void** service(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  //得到请求方式  String method = req.getMethod();  **if** (method.equals(*METHOD\_GET*)) {  **long** lastModified = getLastModified(req);  **if** (lastModified == -1) {  // servlet doesn't support if-modified-since, no reason  // to go through further expensive logic  doGet(req, resp);  } **else** {  **long** ifModifiedSince;  **try** {  ifModifiedSince = req.getDateHeader(*HEADER\_IFMODSINCE*);  } **catch** (IllegalArgumentException iae) {  // Invalid date header - proceed as if none was set  ifModifiedSince = -1;  }  **if** (ifModifiedSince < (lastModified / 1000 \* 1000)) {  // If the servlet mod time is later, call doGet()  // Round down to the nearest second for a proper compare  // A ifModifiedSince of -1 will always be less  maybeSetLastModified(resp, lastModified);  doGet(req, resp);  } **else** {  resp.setStatus(HttpServletResponse.*SC\_NOT\_MODIFIED*);  }  }  } **else** **if** (method.equals(*METHOD\_HEAD*)) {  **long** lastModified = getLastModified(req);  maybeSetLastModified(resp, lastModified);  doHead(req, resp);  } **else** **if** (method.equals(*METHOD\_POST*)) {  doPost(req, resp);    } **else** **if** (method.equals(*METHOD\_PUT*)) {  doPut(req, resp);    } **else** **if** (method.equals(*METHOD\_DELETE*)) {  doDelete(req, resp);    } **else** **if** (method.equals(*METHOD\_OPTIONS*)) {  doOptions(req,resp);    } **else** **if** (method.equals(*METHOD\_TRACE*)) {  doTrace(req,resp);    } **else** {  //  // Note that this means NO servlet supports whatever  // method was requested, anywhere on this server.  //  String errMsg = *lStrings*.getString("http.method\_not\_implemented");  Object[] errArgs = **new** Object[1];  errArgs[0] = method;  errMsg = MessageFormat.*format*(errMsg, errArgs);    resp.sendError(HttpServletResponse.*SC\_NOT\_IMPLEMENTED*, errMsg);  }  } |

## 3.6 案例- 什么是时间戳

很多网站在发布版本之前，都会在URL请求地址后面加上一个实现戳进行版本更新。

## 3.7 案例- 防止非法链接(referer)

第1次 蚂蚁课堂 -> 页面(展示一张图片)

第2次 直接查看展示图片

非法链接：

1）直接访问访问资源

referer： 当前请求来自于哪里。

代码:

|  |
| --- |
| **<filter>**  **<filter-name>ImgFilter</filter-name>**  **<filter-class>com.itmayiedu.filter.ImgFilter</filter-class>**  **</filter>**  **<filter-mapping>**  **<filter-name>ImgFilter</filter-name>**  **<url-pattern>/static/\*</url-pattern>**  **</filter-mapping>** |

|  |
| --- |
| public class ImgFilter implements Filter {  public void destroy() {  // TODO Auto-generated method stub  }  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)  throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) servletRequest;  HttpServletResponse response = (HttpServletResponse) servletResponse;  String referer = request.getHeader("referer");  System.*out*.println("refer is" + "" + referer);  if (referer == null || !referer.contains(request.getServerName())) {  request.getRequestDispatcher("/static/error.png").forward(request, response);  } else {  filterChain.doFilter(request, response);  }  }  public void init(FilterConfig arg0) throws ServletException {  // TODO Auto-generated method stub  }  } |

## 传递的请求参数如何获取

GET方式： 参数放在URI后面

POST方式： 参数放在实体内容中

获取GET方式参数：

request.getQueryString();

获取POST方式参数：

request.getInputStream();

问题：但是以上两种不通用，而且获取到的参数还需要进一步地解析。

所以可以使用统一方便的获取参数的方式：

核心的API：

request.getParameter("参数名"); 根据参数名获取参数值（注意，只能获取一个值的参数）

request.getParameterValue("参数名“)；根据参数名获取参数值（可以获取多个值的参数）

request.getParameterNames(); 获取所有参数名称列表

# 五、 Http响应

|  |
| --- |
| HTTP/1.1 200 OK --响应行  Server: Apache-Coyote/1.1 --响应头（key-vaule）  Content-Length: 24  Date: Fri, 30 Jan 2015 01:54:57 GMT  --一个空行  this is hello servlet!!! --实体内容 |

## 5.1 响应行

### #http协议版本

### #状态码: 服务器处理请求的结果（状态）

常见的状态：

200： 表示请求处理完成并完美返回

302： 表示请求需要进一步细化。

404： 表示客户访问的资源找不到。

500： 表示服务器的资源发送错误。（服务器内部错误）

### #状态描述

## 5.2 常见的响应头

|  |
| --- |
| Location: http://www.it315.org/index.jsp -表示重定向的地址，该头和302的状态码一起使用。  Server:apache tomcat ---表示服务器的类型  Content-Encoding: gzip -- 表示服务器发送给浏览器的数据压缩类型  Content-Length: 80 --表示服务器发送给浏览器的数据长度  Content-Language: zh-cn --表示服务器支持的语言  Content-Type: text/html; charset=GB2312 --表示服务器发送给浏览器的数据类型及内容编码  Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT --表示服务器资源的最后修改时间  Refresh: 1;url=http://www.it315.org --表示定时刷新  Content-Disposition: attachment; filename=aaa.zip --表示告诉浏览器以下载方式打开资源（下载文件时用到）  Transfer-Encoding: chunked  Set-Cookie:SS=Q0=5Lb\_nQ; path=/search --表示服务器发送给浏览器的cookie信息（会话管理用到）  Expires: -1 --表示通知浏览器不进行缓存  Cache-Control: no-cache  Pragma: no-cache  Connection: close/Keep-Alive --表示服务器和浏览器的连接状态。close：关闭连接 keep-alive:保存连接 |

## 5.3 HttpServletResponse对象

HttpServletResponse对象修改响应信息：

响应行：

response.setStatus() 设置状态码

响应头：

response.setHeader("name","value") 设置响应头

实体内容：

response.getWriter().writer(); 发送字符实体内容

response.getOutputStream().writer() 发送字节实体内容

## 5.4 案例- 请求重定向（Location）

|  |
| --- |
| **resp.setStatus(302);**  **resp.setHeader("Location", "OtherServlet");** |

# 六、Https与Http

## 6.1.1 https 与http区别？

　1、https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用。

　2、http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

　3、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

　4、http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

## 6.1.2 https工作原理？

我们都知道 HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。

　客户端在使用 HTTPS 方式与 Web 服务器通信时有以下几个步骤，如图所示。

　　（1）客户使用 https 的 URL 访问 Web 服务器，要求与 Web 服务器建立 SSL 连接。

　　（2）Web 服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

　　（3）客户端的浏览器与 Web 服务器开始协商 SSL 连接的安全等级，也就是信息加密的等级。

　　（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

　　（5）Web 服务器利用自己的私钥解密出会话密钥。

　　（6）Web 服务器利用会话密钥加密与客户端之间的通信。



## 6.1.3 https优缺点？

　虽然说 HTTPS 有很大的优势，但其相对来说，还是存在不足之处的：

　　（1）HTTPS 协议握手阶段比较费时，会使页面的加载时间延长近 50%，增加 10% 到 20% 的耗电；

　　（2）HTTPS 连接缓存不如 HTTP 高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

　　（3）SSL 证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

　 （4）SSL 证书通常需要绑定 IP，不能在同一 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗。

　　（5）HTTPS 协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL 证书的信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间人攻击一样可行。

# http请求工具

## 客户端模拟http请求工具

Postmen(谷歌插件)、RestClient

## 服务器模拟http请求工具

httpclient、HttpURLConnection

httpCient请求代码

|  |
| --- |
| /\*\*  \* 发送 post请求访问本地应用并根据传递参数不同返回不同结果  \*/  public void post() {  // 创建默认的httpClient实例.  CloseableHttpClient httpclient = HttpClients.*createDefault*();  // 创建httppost  HttpPost httppost = new HttpPost("http://localhost:8080/myDemo/Ajax/serivceJ.action");  // 创建参数队列  List<NameValuePair> formparams = new ArrayList<NameValuePair>();  formparams.add(new BasicNameValuePair("type", "house"));  UrlEncodedFormEntity uefEntity;  try {  uefEntity = new UrlEncodedFormEntity(formparams, "UTF-8");  httppost.setEntity(uefEntity);  System.*out*.println("executing request " + httppost.getURI());  CloseableHttpResponse response = httpclient.execute(httppost);  try {  HttpEntity entity = response.getEntity();  if (entity != null) {  System.*out*.println("--------------------------------------");  System.*out*.println("Response content: " + EntityUtils.*toString*(entity, "UTF-8"));  System.*out*.println("--------------------------------------");  }  } finally {  response.close();  }  } catch (ClientProtocolException e) {  e.printStackTrace();  } catch (UnsupportedEncodingException e1) {  e1.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  } finally {  // 关闭连接,释放资源  try {  httpclient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  /\*\*  \* 发送 get请求  \*/  public void get() {  CloseableHttpClient httpclient = HttpClients.*createDefault*();  try {  // 创建httpget.  HttpGet httpget = new HttpGet("http://www.baidu.com/");  System.*out*.println("executing request " + httpget.getURI());  // 执行get请求.  CloseableHttpResponse response = httpclient.execute(httpget);  try {  // 获取响应实体  HttpEntity entity = response.getEntity();  System.*out*.println("--------------------------------------");  // 打印响应状态  System.*out*.println(response.getStatusLine());  if (entity != null) {  // 打印响应内容长度  System.*out*.println("Response content length: " + entity.getContentLength());  // 打印响应内容  System.*out*.println("Response content: " + EntityUtils.*toString*(entity));  }  System.*out*.println("------------------------------------");  } finally {  response.close();  }  } catch (ClientProtocolException e) {  e.printStackTrace();  } catch (ParseException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  } finally {  // 关闭连接,释放资源  try {  httpclient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  } |

## 前端ajax请求

ajax是异步请求。做的局部刷新。但是也可以设置为同步。async设置成true

|  |
| --- |
| $.ajax({  type : 'post',  dataType : "text",  url : "http://a.a.com/a/FromUserServlet",  data : "userName=余胜军&userAge=19",  success : **function**(msg) {  alert(msg);  }  }); |

## 跨域实战解决方案

跨域原因产生：在当前域名请求网站中，默认不允许通过ajax请求发送其他域名。

XMLHttpRequest cannot load 跨域问题解决办法

1. 在相应头设置允许跨域。最简单的。
2. jsonp解决，只能解决get，post无效。
3. 搭建网关系统。适合大型网关。
4. 使用后台服务转发。用的比较少，比较占用带宽。但是不会暴露接口，比较安全。

## 使用后台response添加header

后台response添加header，**response.setHeader("Access-Control-Allow-Origin", "\*"); 支持所有网站**

## **使用JSONP**

前端代码:

|  |
| --- |
| $.ajax({  type : "POST",  async : false,  url : "http://a.a.com/a/FromUserServlet?userName=张三",  dataType : "jsonp",//数据类型为jsonp  jsonp : "jsonpCallback",//服务端用于接收callback调用的function名的参数  success : function(data) {  alert(data.result);  },  error : function() {  alert('fail');  }  }); |

后端代码:

|  |
| --- |
| @WebServlet("/FromUserServlet")  public class FromUserServlet extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  doPost(req, resp);  }  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  resp.setCharacterEncoding("UTF-8");  // resp.setHeader("Access-Control-Allow-Origin", "\*");  String userName = req.getParameter("userName");  String userAge = req.getParameter("userAge");  System.*out*.println(userName + "----" + userAge+"---"+req.getMethod());  // JSONObject JSONObject1 = new JSONObject();  // JSONObject1.put("success", "添加成功!");  // resp.getWriter().write("callbackparam(" + JSONObject1.toJSONString()  // + ")");  try {  resp.setContentType("text/plain");  resp.setHeader("Pragma", "No-cache");  resp.setHeader("Cache-Control", "no-cache");  resp.setDateHeader("Expires", 0);  PrintWriter out = resp.getWriter();  JSONObject resultJSON = new JSONObject(); // 根据需要拼装json  resultJSON.put("result", "content");  String jsonpCallback = req.getParameter("jsonpCallback");// 客户端请求参数  out.println(jsonpCallback + "(" + resultJSON.toJSONString() + ")");// 返回jsonp格式数据  out.flush();  out.close();  } catch (Exception e) {  // TODO: handle exception  }  }  } |

JSONP的优缺点:

JSONP只支持get请求不支持psot请求

jsonp实现原理

生成一个get请求发送到服务器端

因为jsonp底层就是发的get请求。

405，请求方法不对。

面试技巧。

你在项目中遇到了哪些问题，跨域问题，12345解决方案，优缺点。

## **使用接口网关**

使用nginx转发。

# http抓包工具

fiddler、wireshark

# 本节课程maven坐标

|  |
| --- |
| <dependencies>  <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  <scope>provided</scope>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  <version>4.3.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.29</version>  </dependency>  </dependencies> |