# 1 Servlet

## 1.1 Servlet简介

* Servlet是sun公司提供的一门用于开发**动态web资源的技术**。
* Sun公司在其API中提供了一个**servlet接口**
* Servlet 是运行在 Web 服务器中的小型 Java 程序（即：服务器端的小应用程序）。
* servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求。
* 若想开发一个动态web资源(即开发一个Java程序向浏览器输出数据)，要完成2个步骤：
* 编写一个Java类，实现servlet接口。
* 把开发好的Java类部署到web服务器中。

## 1.2 Servlet快速入门案例

* 案例需求：用Servlet向浏览器输出“hello servlet”

### 第一步：

|  |
| --- |
| * 创建一个hello的web工程 * 然后写一个java类，实现servlet接口 |
|  |

### 第二步：

|  |
| --- |
| * 在web.xml中配置Servlet映射 |
|  |

|  |
| --- |
| * 把hello工程部署到tomcat中运行 * 然后浏览器访问http://localhost:8080/hello/Hello |
|  |

### 第三步：

## 1.3 Servlet的执行过程

* Servlet作用处理请求
* 当浏览器访问一个http://localhost:8080/hello/hello 路径，就向tomcat发送一个请求

|  |
| --- |
|  |

## 1.4 Servlet生命周期（重要）

* Servlet生命周期四个状态：**实例化-->初始化-->服务->销毁**
* 出生：（实例化-->初始化）第一次访问Servlet就出生（默认情况下）
* 活着：（服务）应用活着，servlet就活着
* 死亡：（销毁）应用卸载了servlet就销毁。

|  |
| --- |
|  |

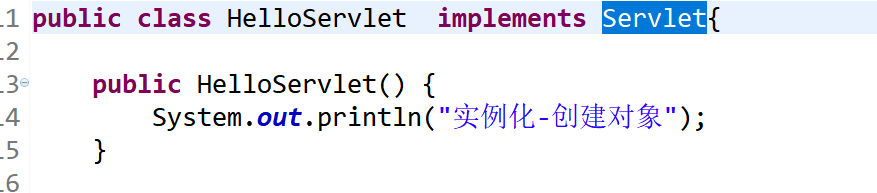
## 1.5 Servlet的创建时机

* 默认情况下Servlet在第一次使用Servlet时才创建
* 可以在web.xml中设置load-on-startup为2，Servlet就会启动时调用构造方法和初始化方法

|  |
| --- |
|  |

## 1.6 Servlet实现的三种方式

### 实现javax.servlet.Servlet接口（前面写过了）



### 继承javax.servet.GenericServlet类(适配器模式)

|  |
| --- |
|  |

### 继承javax.servlet.http.HttpServlet类（模板方法设计模式）

|  |
| --- |
|  |

### 了解servlet结构

|  |
| --- |
|  |
|  |
|  |

## 1.7 Servlet映射细节

* **掌握Servlet映射规则**
* **通配符\* 代表任意字符串，**如图：

|  |
| --- |
| url-pattern: \*.do 以\*.字符串的请求都可以访问 注：不要加/ |
| url-pattern: /\* 任意字符串都可以访问 |
| url-pattern： /action/\* 以/action开头的请求都可以访问 |
| **匹配规则：**  优先级：从高到低  绝对匹配--> /开头匹配 --> 扩展名方式匹配  **如果url-pattern的值是/，表示执行默认映射**。所有资源都是servlet |
|  |

## 1.8 Servlet的线程安全问题

* Servlet是单例：多线程每次访问都是同一个对象
* 解决线程安全问题的最佳办法，**不要写全局变量,尽量不要，而写局部变量。**

|  |
| --- |
|  |

## Servlet的注解映射

### Servlet2.5以前映射配置

|  |
| --- |
|  |

### Servlet 3.0 注解配置

**注意：在配置映射时，路径要加斜杠,不然会报错**

|  |
| --- |
|  |
|  |

## 1.9 ServletContext

Context:上下文

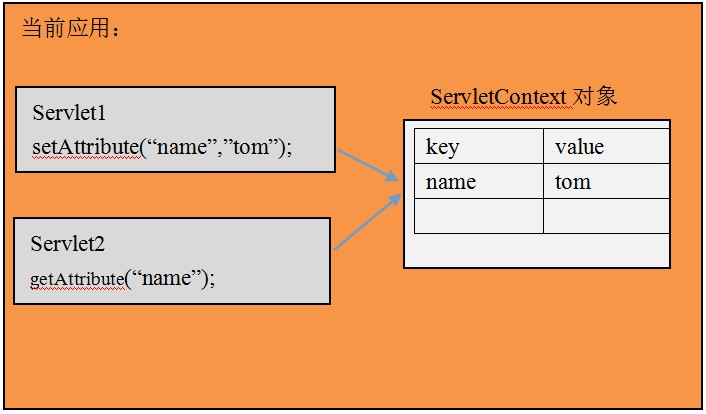
ServletContext: 代表的是整个应用。**一个应用只有一个ServletContext对象**。是**单例对象** 。

### 作用：

域对象：在一定范围内（当前应用），使多个**Servlet共享数据**。

### 常用方法：

* void setAttribute(String name,object value);//向ServletContext对象的**map**中添加数据
* Object getAttribute(String name);//从ServletContext对象的map中取数据
* void rmoveAttribute(String name);//根据name去移除数据



|  |
| --- |
|  |
|  |

### 获取全局配置信息

|  |
| --- |
| 第一步:在web.xml中配置一个context-param |
|  |
| 第二步：获取web.xml中的配置 |
|  |

### 获取资源路径

**String getRealPath(String path);**

* 根据资源名称得到资源的绝对路径.
* 可以得到当前应用任何位置的任何资源。

|  |
| --- |
| **public** **class** Lesson03Servlet1 **extends** HttpServlet{  /\*\*  \* 通过浏览器地址栏访问的路径都是get请求  \*/  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  /\* ServletContext的String getRealPath(String path)方法;  1.根据资源名称得到资源的绝对路径.  2.可以得到当前应用任何位置的任何资源\*/    //解决乱码  resp.setContentType("text/html");  resp.setCharacterEncoding("UTF-8");    /\*\*  \* String path = "src/com/gyf/web/lesson03/info.properties";  \* 如果是web项目，查找文件时，要从类路径找  \*/    String path = **this**.getServletContext().getRealPath("WEB-INF/classes/com/gyf/web/lesson03/info.properties");  System.***out***.println(path);    //在Servlet来获取info.properties数据  //1.创建属性对象  Properties pro = **new** Properties();    //2.关连属性文件的路径  pro.load(**new** FileInputStream(path));    System.***out***.println(pro.getProperty("username"));    //响应客户端  resp.getWriter().write(path);  resp.getWriter().write("-----");  resp.getWriter().write(pro.getProperty("username"));  }  } |
| **如果a.properties在src目录下**，不需要写包名路径 |
|  |

## 1.10 Servlet的转发

### 转发图解

|  |
| --- |
|  |

### 代码实现

|  |
| --- |
|  |
|  |

# 2 HTTP

## 2.1简介

**HTTP协议（HyperText Transfer Protocol，超文本传输协议）是用于从WWW服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。**

## 2.2 HTTP请求流程

|  |
| --- |
|  |

## 2.3 HTTP消息头

* HTTP消息头是指在超文本传输**协议**（ Hypertext Transfer Protocol ，HTTP）的请求和响应消息中，协议头部分的那些组件。
* HTTP消息头用来准确描述正在获取的资源、服务器或者客户端的行为
* HTTP消息头定义了HTTP事务中的具体操作参数。
* **消息头包括请求时的消息头（请求头）和响应时的消息头（响应头）**

|  |
| --- |
|  |
|  |

## 2.4 HTTP的常见请求头

* Accept：浏览器可接受的MIME类型。
* Accept-Charset：浏览器可接受的字符集。
* Accept-Encoding：浏览器能够进行解码的数据编码方式，比如gzip。Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。
* Accept-Language：浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。
* Authorization：授权信息，通常出现在对服务器发送的WWW-Authenticate头的应答中。
* Connection：表示是否需要**持久连接**。如果Servlet看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点，Servlet需要在应答中发送一个Content-Length头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。
* **Content-Length**：表示请求消息正文的长度。
* **Cookie：**这是最重要的请求头信息之一
* From：请求发送者的email地址，由一些特殊的Web客户程序使用，浏览器不会用到它。
* **Host**：初始URL中的主机和端口。
* If-Modified-Since：只有当所请求的内容在指定的日期之后又经过修改才返回它，否则返回304“Not Modified”应答。
* Pragma：指定“no-cache”值表示服务器必须返回一个刷新后的文档，即使它是代理服务器而且已经有了页面的本地拷贝。
* Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。
* User-Agent：**浏览器类型**，如果Servlet返回的内容与浏览器类型有关则该值非常有用。

## 2.5 HTTP的常见响应头

* **Allow：** 服务器支持哪些请求方法（如GET、POST等）。
* **Content-Encoding**：文档的编码（Encode）方法。只有在解码之后才可以得到Content-Type头指定的内容类型。利用gzip压缩文档能够显著地减少HTML文档的下载时间。Java的GZIPOutputStream可以很方便地进行gzip压缩，但只有Unix上的Netscape和Windows上的IE 4、IE 5才支持它。因此，Servlet应该通过查看Accept-Encoding头（即request.getHeader("Accept-Encoding")）检查浏览器是否支持gzip，为支持gzip的浏览器返回经gzip压缩的HTML页面，为其他浏览器返回普通页面。
* **Content-Length**：表示内容长度。只有当浏览器使用持久HTTP连接时才需要这个数据。如果你想要利用持久连接的优势，可以把输出文档写入 ByteArrayOutputStream，完成后查看其大小，然后把该值放入Content-Length头，最后通过byteArrayStream.writeTo(response.getOutputStream()发送内容。
* **Content-Type** 表示后面的文档属于什么MIME类型。Servlet默认为text/plain，但通常需要显式地指定为text/html。由于经常要设置Content-Type，因此HttpServletResponse提供了一个专用的方法setContentType。
* **Date**  当前的GMT时间。你可以用setDateHeader来设置这个头以避免转换时间格式的麻烦。
* **Expires** 应该在什么时候认为文档已经过期，从而不再缓存它？
* **Last-Modified** 文档的最后改动时间。客户可以通过If-Modified-Since请求头提供一个日期，该请求将被视为一个条件GET，只有改动时间迟于指定时间的文档才会返回，否则返回一个304（Not Modified）状态。Last-Modified也可用setDateHeader方法来设置。
* **Location** 表示客户应当到哪里去提取文档。Location通常不是直接设置的，而是通过HttpServletResponse的sendRedirect方法，该方法同时设置状态代码为302。
* **Refresh** 表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过setHeader("Refresh", "5; URL=http://host/path")让浏览器读取指定的页面。
  + 注意Refresh头不属于HTTP 1.1正式规范的一部分，而是一个扩展，但Netscape和IE都支持它。
* **Server**服务器名字。Servlet一般不设置这个值，而是由Web服务器自己设置。
* **Set-Cookie** 设置和页面关联的Cookie。Servlet不应使用response.setHeader("Set-Cookie", ...)，而是应使用HttpServletResponse提供的专用方法addCookie。参见下文有关Cookie设置的讨论。
* **WWW-Authenticate**客户应该在Authorization头中提供什么类型的授权信息？在包含401（Unauthorized）状态行的应答中这个头是必需的。例如，response.setHeader("WWW-Authenticate", "BASIC realm=＼"executives＼"")。
  + 注意Servlet一般不进行这方面的处理，而是让Web服务器的专门机制来控制受密码保护页面的访问（例如.htaccess）。