# 

**Servlet学习笔记**

**Java相关课程系列笔记之九**

**笔记内容说明**

Servlet（程祖红老师主讲，占笔记内容100%）；

**目 录**

[一、 Servlet概述 1](#_Toc17184)

[1.1 B/S架构（了解） 1](#_Toc19912)

[1.2什么是Servlet 1](#_Toc26)

[1.3什么是Tomcat 2](#_Toc7412)

[1.4如何写一个Servlet（不使用开发工具） 3](#_Toc27369)

[1.5使用MyEclipse开发Servlet 5](#_Toc5043)

[1.6 Servlet是如何运行的 6](#_Toc8393)

[1.7常见错误及解决方式 6](#_Toc25816)

[1.8案例：根据请求次数显示结果和显示当前时间 7](#_Toc4517)

[二、 HTTP协议 8](#_Toc12160)

[2.1什么是HTTP协议 8](#_Toc31642)

[2.2通讯的过程 8](#_Toc12028)

[2.3数据格式 8](#_Toc29621)

[2.4 TCP/IP Monitor的使用 9](#_Toc22305)

[2.5 get请求与post请求 9](#_Toc28041)

[2.6如何读取请求参数 9](#_Toc10236)

[2.7访问数据库（MySql） 10](#_Toc144)

[2.8案例：添加员工（访问MySql数据库） 10](#_Toc7227)

[2.9异常：IllegalStateException 11](#_Toc24948)

[三、 编码问题 12](#_Toc28620)

[3.1 Java语言在内存当中默认使用的字符集 12](#_Toc30381)

[3.2编码 12](#_Toc10365)

[3.3解码 12](#_Toc27426)

[3.4 Servlet如何输出中文 12](#_Toc22697)

[3.5如果表单有中文参数值，也需要注意编码问题 12](#_Toc30372)

[3.6案例：根据请求正确显示中文 12](#_Toc5169)

[3.7将中文数据插入到数据库 13](#_Toc4995)

[四、 重定向 14](#_Toc4687)

[4.1什么是重定向 14](#_Toc20880)

[4.2如何重定向 14](#_Toc2832)

[4.3注意两个问题 14](#_Toc14905)

[4.4两个特点 14](#_Toc5485)

[4.5重定向原理图：以2.8案例为例 14](#_Toc5815)

[五、 DAO 15](#_Toc13147)

[5.1什么是DAO 15](#_Toc12376)

[5.2如何写一个DAO 15](#_Toc25430)

[5.3工厂类 15](#_Toc9325)

[六、 工厂设计模式 16](#_Toc2855)

[6.1什么是工厂 16](#_Toc28778)

[6.2使用工厂设计模式的好处 16](#_Toc28153)

[6.3如何使用工厂模式 16](#_Toc16437)

[6.4案例：为2.8案例添加新功能，并使用DAO和工厂模式 16](#_Toc12281)

[七、 Servlet容器处理细节 19](#_Toc6071)

[7.1 Servlet容器如何处理请求资源路径以及匹配 19](#_Toc15379)

[7.2一个Servlet如何处理多种请求 19](#_Toc31274)

[八、 Servlet的生命周期 20](#_Toc17554)

[8.1 Servlet的生命周期的含义 20](#_Toc1219)

[8.2 Servlet生命周期的四个阶段 20](#_Toc7587)

[8.3实例化 20](#_Toc24477)

[8.4初始化 20](#_Toc26892)

[8.5就绪 20](#_Toc16151)

[8.6销毁 21](#_Toc1183)

[8.7 Servlet生命周期图 21](#_Toc16640)

[8.8 Servlet生命周期相关的几个接口与类 21](#_Toc31014)

[九、 JSP（简要介绍，详细内容见JSP笔记） 23](#_Toc3140)

[9.1什么是JSP 23](#_Toc21989)

[9.2为什么要使用JSP 23](#_Toc3842)

[9.3 JSP与Servlet的关系 23](#_Toc1245)

[9.4如何写一个JSP文件 23](#_Toc22054)

[9.5 JSP是如何运行的 23](#_Toc18036)

[9.6指令 24](#_Toc4860)

[9.7案例：创建emplist.jsp页面，将表示逻辑交给JSP处理 24](#_Toc14140)

[十、 请求转发 26](#_Toc18022)

[10.1什么是转发 26](#_Toc6364)

[10.2如何转发 26](#_Toc1676)

[10.3编程需要注意的两个问题 26](#_Toc8052)

[10.4转发原理图 26](#_Toc15170)

[10.5转发的特点 27](#_Toc20211)

[10.6转发和重定向的区别 27](#_Toc20095)

[10.7何时用重定向 27](#_Toc4143)

[10.8何时用转发 27](#_Toc31121)

[10.9案例：修改6.4案例中step7中的ListEmpServlet.java 28](#_Toc9162)

[十一、 异常的处理 30](#_Toc23797)

[11.1用输出流out 30](#_Toc2338)

[11.2用转发的方式 30](#_Toc16385)

[11.3让容器处理系统异常 30](#_Toc25055)

[11.4案例：将10.9案例中的step3中的所有catch块修改 30](#_Toc32447)

[十二、 路径问题 31](#_Toc24289)

[12.1什么是相对路径 31](#_Toc8100)

[12.2什么是绝对路径 31](#_Toc27028)

[12.3如何写绝对路径 31](#_Toc6156)

[12.4如何防止硬编码 31](#_Toc5368)

[12.5案例：相对、绝对路径对比 31](#_Toc15787)

[12.6四种情况下，正确的绝对路径写法 32](#_Toc31712)

[十三、 状态管理 33](#_Toc5189)

[13.1什么是状态管理 33](#_Toc7735)

[13.2如何进行状态管理 33](#_Toc25201)

[13.3 cookie 33](#_Toc7493)

[13.4如何创建一个cookie 33](#_Toc3381)

[13.5查询cookie 33](#_Toc30541)

[13.6编码问题 34](#_Toc31163)

[13.7 cookie的生存时间 34](#_Toc9474)

[13.8 cookie的路径问题 34](#_Toc28013)

[13.9 cookie的限制 35](#_Toc20377)

[13.10案例：写一个CookieUtil 35](#_Toc26686)

[13.11 session（会话） 36](#_Toc19593)

[13.12如何创建一个session对象 36](#_Toc4583)

[13.13 HttpSession接口中提供的常用方法 36](#_Toc13372)

[13.14 session的超时 37](#_Toc23137)

[13.15用户禁止cookie以后，如何继续使用session 37](#_Toc21299)

[13.16 url重写 37](#_Toc12009)

[13.17 session的优点 37](#_Toc29043)

[13.18 session的缺点 38](#_Toc28741)

[13.19案例：session验证和访问次数 38](#_Toc2611)

[13.20案例：验证码 39](#_Toc31788)

[13.21案例：购物车 41](#_Toc7716)

[十四、 过滤器 45](#_Toc27394)

[14.1什么是过滤器 45](#_Toc23753)

[14.2如何写一个过滤器 45](#_Toc27710)

[14.3案例：敏感字过滤器 45](#_Toc1462)

[14.4过滤器的优先级 46](#_Toc16180)

[14.5初始化参数 47](#_Toc10424)

[14.6优点 47](#_Toc29477)

[十五、 监听器 48](#_Toc25638)

[15.1什么是监听器 48](#_Toc12471)

[15.2容器会产生两大类事件 48](#_Toc31481)

[15.3如何写一个监听器 48](#_Toc8261)

[15.4 ServletContext（Servlet上下文） 48](#_Toc10327)

[15.5如何获得Servlet上下文 48](#_Toc23470)

[15.6 Servlet上下文的作用 48](#_Toc1201)

[15.7案例：统计在线人数 49](#_Toc3972)

[十六、 上传文件 51](#_Toc12458)

[16.1如何上传文件 51](#_Toc20170)

[16.2案例：上传文件 51](#_Toc23649)

[十七、 Servlet线程安全问题 53](#_Toc30123)

[17.1为何Servlet会有线程安全问题 53](#_Toc13775)

[17.2如何处理线程安全问题 53](#_Toc6857)

[十八、 Servlet小结 54](#_Toc24412)

[18.1 Servlet基础 54](#_Toc19471)

[18.2 Servlet核心 54](#_Toc22491)

[18.3状态管理 54](#_Toc31548)

[18.4数据库访问 54](#_Toc18303)

[18.5过滤器和监听器 54](#_Toc24434)

[18.6典型案例和扩展 54](#_Toc410)

[十九、 其他注意事项 55](#_Toc31886)

[19.1连接数据库的工具 55](#_Toc18716)

[19.2知名网站 55](#_Toc17956)

[19.3 C/S架构：Client/Server 55](#_Toc5670)

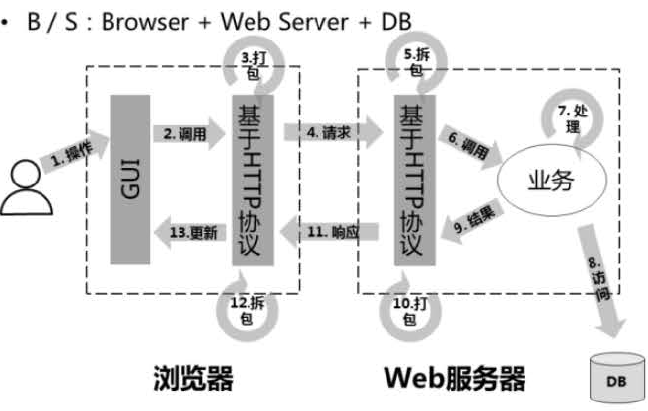
[19.4 B/S架构：Browser/Server 56](#_Toc17218)

# 一、Servlet概述

## 1.1 B/S架构（了解）

### 1.1.1 什么是B/S架构

客户端使用浏览器，服务端使用web浏览器，客户端跟服务器之间使用HTTP协议进行通讯。



1. 数据库只负责数据的管理

2. Web服务器负责业务逻辑的处理

3. 浏览器负责提供操作页面

### 1.1.2 B/S特点

1. 客户端不需要单独安装，因为使用浏览器。C/S架构比较麻烦的是需要单独安装每个客户端，并且一旦客户端版本发生改变，就需要再次安装。

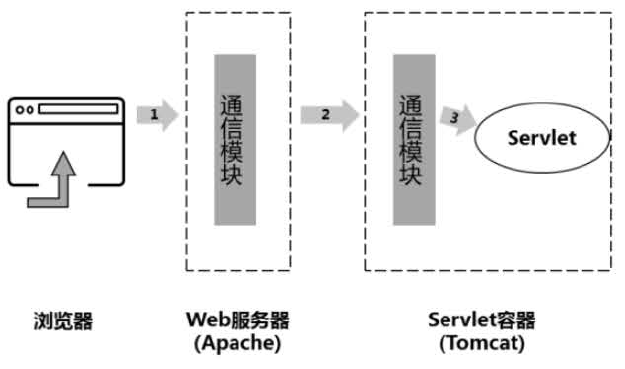
2. 开发相对简单，C/S架构需要我们分别在，客户端和服务器端编写相应的通信处理模块和自定义协议，而B/S架构使用标准的HTTP协议（即不再需要自定义协议），而且浏览器与Web服务器已经包含了相应的通信模块了。

## 1.2什么是Servlet

### 1.2.1 servlet概述

Sun公司制订的一种用来扩展Web服务器功能的组件规范。早期（2000年左右）的Web服务器：apache的web server、微软的iis。只能够处理静态资源，即需要事先将html文件写好，不能够处理动态资源的请求，即需要依据请求参数然后进行计算，生成相应的页面。

为了让这些web服务器能够处理动态资源的请求，需要扩展他们的功能。现在，可以使用Servlet来扩展。



当浏览器将请求发送给Web服务器，Web服务器会向Servlet容器发送请求，Servlet容器负责解析请求数据包。当然，也包括网络通讯相关的一些处理，然后将解析之后的数据交给Servlet来处理。因此Servlet只需要关注具体的业务处理，不用关心网络通讯相关的问题。

注意事项：可以不使用服务器，而直接向Servlet容器发送请求，因为Servlet容器里面也有个通信模块，所以也可直接把Servlet容器当作简单的Web服务器来使用。

### 1.2.2 组件规范

组件：是符合一定规范，并且实现部分功能的可以单独部署的软件模块。组建必须要部署到容器里面才能运行。

容器：也是符合一定规范，并且提供组件的运行环境的程序。

注意事项：单个的组件、单个的容器都是没意义的，都不能单独运行，需要放在一起才能运行。

## 1.3什么是Tomcat

### 1.3.1 Tomcat概述

Tomcat本身是一个Servlet容器，即可以提供Servlet运行环境的一个程序，但是Tomcat还提供了Web服务器所具有的所有功能，所以我们也称Tomcat是一个Web服务器。Tomcat的默认端口是8080。

### 1.3.2 Tomcat目录结构简介

bin目录：存放启动和关闭服务器的一些脚本（命令）。

common目录：共享(部署在该服务器上的所有程序都可以使用)的一些jar包。

conf目录：存放服务器的一些配置文件。

webapps目录：部署目录。

work目录：服务器运行时，生成的一些临时文件。

## 1.4如何写一个Servlet（不使用开发工具）

**step1：**先写一个**Java类**，实现Servlet接口或者继承HttpServlet抽象类。

1. Tomcat会自动调用service，自己不用再去写方法调用了

2. 设置消息头content-type，告诉浏览器返回的数据类型是html文档，以及编码格3. 式此外，还可以告诉服务器，在使用out输出时，使用指定的编码格式进行编码

4. 通过响应对象，获得一个输出流

5. 调用流的方法进行输出，其实质是将处理结果写到了response对象上

**step2：**执行**编译操作**：javac -d . HelloWorldServlet.java

**step3**：**打包**，即创建一个具有如下结构的文件夹：将编译好的.class文件放入classes文件夹

appname（文件夹名起应用名）

|－－WEB-INF（必须大写）

｜－－classes（文件夹，放.class文件）

｜－－lib（文件夹，放.jar文件，可没有）

｜－－web.xml（部署描述文件 url-pattern）

**step4**：**部署**，将step3创建的文件夹拷贝到Servlet容器特定的文件夹下面注意事项：也可以将step3创建的文件夹使用jar命令进行压缩，生成.war为后缀的文件，然后拷贝。

**step5**：**启动**Servlet容器，**访问**Servlet。

访问格式：http://ip:port/appname/url-pattern

比如在浏览器地址栏输入：http://localhost:8080/web01/hello

## 1.6 Servlet是如何运行的

比如，在浏览器地址栏输入：http://ip:port/web01/hello（

step1：浏览器依据ip和port建立与Servlet容器之间的连接。

step2：浏览器将请求数据打包，并发送给Servlet容器。

step3：**Servlet容器**解析请求数据包，并将解析数据放到已创建的request对象上。

同时，容器也已经创建好了一个response对象。

step4：**Servlet容器**依据请求资源路径找到Servlet的配置，然后创建Servlet对象

根据xml文件里配置的类而创建。

step5：**Servlet容器**接下来调用Servlet对象的service方法，并且会将事先创建好的

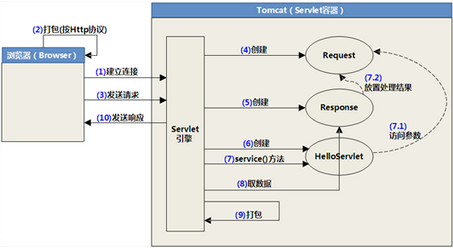
request对象和response对象作为service方法的参数传递给Servlet对象。

step6：Servlet可以通过request对象获得请求参数，进行相应的处理

然后将处理结果写到response对象上

step7：**Servlet容器**读取response对象上的数据，然后将处理结果打包并发送给浏览器

step8：**浏览器**解析响应数据包，将返回的数据展现给用户。



## 1.7常见错误及解决方式

### 1.7.1 错误404

**概述**：404，是一个状态码，是一个三位数字，由服务器发送给浏览器，告诉浏览器是否正确处理了请求，404的意思是说：服务器依据请求资源路径，**找不到对应的资源**。

**解决**：依据http://ip:port/appname/url-pattern检查你的请求地址是否正确。

仔细检查web.xml，特别要注意servlet-name是否一致。

### 1.7.2 错误500

**概述**：500，是服务器处理出错，一般是因为**程序运行出错**。

**解决**：检查程序的代码，比如：是否继承。

检查web.xml，类名要填写正确。

### 1.7.3 错误405

**概述**：405，是服务器**找不到对应的service方法**。

**解决**：检查service方法的签名（方法名、参数类型、返回类型、异常类型）。

## 1.8案例：根据请求次数显示结果和显示当前时间

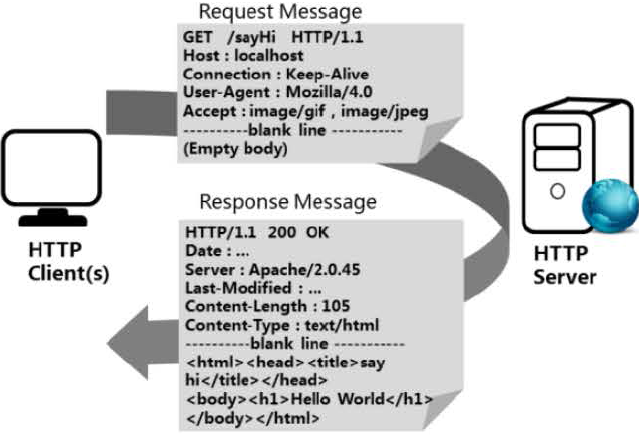
eg1：在地址栏输入http://localhost:8080/web01/hello?qty=5后，获得5个Hello World

eg2：在网页上显示当前时间

# 二、HTTP协议

## 2.1什么是HTTP协议

HTTP（Hypertext transport protocol）是超文本传输协议。是一种应用层协议，由W3C制定，它定义了浏览器与Web服务器之间通讯的过程及数据格式。



## 2.2通讯的过程

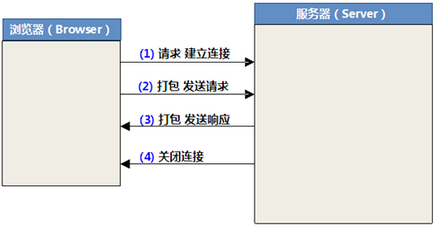
step1：浏览器建立与Web服务器之间的连接（Socket）。

step2：浏览器要将请求数据打包（请求数据包），然后发送给Web服务器。

step3：Web服务器将处理结果打包（响应数据包），然后发送给浏览器。

step4：Web服务器关闭连接。

注意事项：一次请求，一次连接。优点：Web服务器可以利用有限的连接个数为尽可能多的客户服务（效率高）。如果浏览器要再发请求，就必须重新建立一个新的连接。



## 2.3数据格式

### 2.3.1 请求数据包

1. **请求行**：请求方式+请求资源路径+协议描述。

2. **若干消息头**：消息头是一些键值对，一般由W3C定义，有特定的含义。浏览器和服务器之间，可以通过发送消息头来传递一些特定的信息，比如：浏览器可以通过user-agent消息头来通知服务器浏览器的类型和版本。

3. **实体内容**：只有当请求方式为post时，浏览器才会将请求参数添加到实体内容里面，如果请求方式为get，浏览器会将请求参数添加到请求资源路径的后面。



### 2.3.2 响应数据包

**状态行**：协议描述+状态码+状态描述。

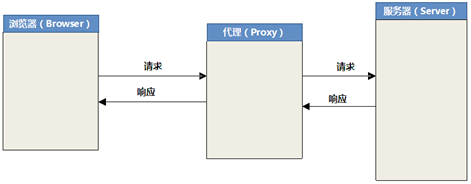
**若干消息头**：服务器也可以发送一些消息头给浏览器，比如content-type，告诉浏览器服务器返回的数据类型和编码格式（字符集，比如：UTF-8、ISO-8859-1）。

**实体内容**：程序处理之后，返回的结果。



### 2.3.3截取数据包

截获数据包，使用MyEclipse中的TCP/IP Monitor，TCP/IP Monitor相当于一个代理服务器，它的原理图如下：



## 2.4 请求方式

### 2.4.1 为什么区分请求方式

请求方式是客户端对话服务器时的意向说明，是区分请求种类的关键。

1. 不同的请求方式不仅仅在数据传输时会有所不同

2. 在表单提交及服务器端处理时都会采用不同的方式，

3. 而区分不同种类的请求方式也会使得浏览器采用不同的缓存方式处理后续请求

4. 从而提升响应速度



### 2.4.2 使用get请求方式

浏览器会使用get方式发请求，情况如下：

1. 直接在浏览器地址输入某个地址。

2. 点击链接地址。

3. 表单默认的提交方法：<form method="get(默认)/post">。

浏览器使用，get请求的特点如下：

1. get请求会将请求参数添加到请求资源路径的后面，因为请求行存放的数据大小有限，也就是地址栏的最长字节数，所以get请求只能提交少量的数据。

2. get请求会将请求参数显示在浏览器地址栏，不安全（比如，路由器会记录整个地址）。

### 2.4.3 使用post请求

浏览器使用post方法发请求，情况如下：

设置表单的method属性值为“post”。

浏览器使用，post请求的特点如下：

post请求会将请求参数添加到实体内容里面，所以，可以提交大量的数据。

post请求不会将请求参数显示在浏览器地址栏，相对安全一些。但是，post请求并不会对请求参数进行加密处理。用HTTPS协议进行加密处理。

注意：

服务器不关心是用浏览器还是Java程序发送的请求，只要符合协议格式，都会处理。

get在地址栏中传值，post在实体中传值

## 2.5读取请求参数

方法一：String request.getParameter(String paraName);

如果paraName（即参数名称）与实际的参数名称不一致，会获得null（不报错）。

在使用表单提交数据时，如果用户没有填写任何的值，会获得空字符串 "" 。

方法二：String[] request.getParameterValues(String paraName);

当有多个参数且名称相同时，使用该方法。比如：?city=bj&city=cs&city=wh

注意事项：getParameterValues方法也可用于只有一个参数的情况。

## 2.6访问数据库（MySql）

### 2.6.1 使用MySql数据库

**登录MySql**：mysql -uroot;//登录mysql，使用root用户权限

**查看**当前所有的**数据库**：show databases;

**创建**一个新的**数据库**：create database db\_chang default character set utf8;

**使用**某个**数据库**：use db\_chang;

**查看**当前数据库有哪些**表**：show tables; drop table tablname;//删表

建表：

create table chang\_emp(

id int primary key auto\_increment,

name varchar(50),

salary double,

age int );

insert into chang\_emp(name,salary,age) values("tom",10000,23);

### 2.6.2 使用JDBC访问数据库

step1：将JDBC驱动放到WEB-INF\lib下

step2：编写JDBC代码，需要注意异常的处理！

# 三、编码问题

## 3.1 Java语言在内存当中默认使用的字符集

默认会用“Unicode”编码格式（字符集）来保存字符。

## 3.2编码

把Unicode这种编码格式对应的字节数组，转换成某种本地编码格式（比如GBK）对应的字节数组，从而保存。Unicode-->GBK。

## 3.3解码

把某种本地编码格式的字节数组转换成Unicode这种编码格式对应的字节数组。GBK-->Unicode。

服务器默认使用ISO-8859-1编码格式，使用1个字节保存，无法存中文！因此中文会出现乱码。

## 3.4 Servlet如何输出中文

需要调用：response.setContentType("text/html;charset=utf-8");其中charset=utf-8表示：

用来指定编码格式，只要支持中文即可，比如也可设置为charset=gbk。

作用两个：① 生成一个content-type消息头，告诉浏览器返回的数据类型和编码格式。

② 服务器在输出时，会使用指定的编码格式进行编码。

## 3.5如果表单有中文参数值，也需要注意编码问题

因为，当表单提交的时候，浏览器会对表单中的数据进行编码，会使用打开表单时的编码格式进行编码，而服务器默认情况下，会使用ISO-8859-1去解码，所以，会产生乱码问题。

**解决方式一：**

step1：先保证表单所在的页面按照指定的编码格式打开。即：

<meta http-equiv="content-type" content="text/html;charset=utf-8" />

已是一种规范模拟content-type消息头，告诉浏览器正在解析的数据类型和编码格式。

step2：调用request.setCharacterEncoding("utf-8");

意思是告诉服务器，使用指定的编码格式进行解码。

注意事项：该方法只能用于"post"请求！注意代码放置顺序，在request.getParameter()方法前。

**解决方式二：**

step1：同方式一的第一步。

step2：使用new String(str.getBytes("iso-8859-1"),"utf-8");

## 3.6将中文数据插入到数据库

程序从内存Unicode编码格式--->数据库中的某种本地格式。

step1：要确保数据库支持中文，即正确设置数据库的字符集。

例如，建MySql的数据库时设置编码：create database db\_chang default character set utf8;

step2：JDBC驱动必须能够正确地进行编码和解码。有些MySql的驱动不能够正确进行编码和解码（默认使用ISO-8859-1进行编码和解码），可以在JDBC的URL后添加：

“useUnicode=true&characterEncoding=utf8”。

例如：conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/db\_chang?" +

"useUnicode=true&characterEncoding=utf8","root","");

注意事项：

* MySql中utf不能写“-”，记得数据库名后有个“？”。
* MySql客户端也是一个程序，也会有编码问题，若你的程序中数据显示正常，而MySql客户端中查询的数据为乱码，则更改下编码，使用：set names gbk;命令。

# 四、重定向

## 4.1什么是重定向

服务器向浏览器发送一个302状态码及一个Location消息头，值是一个地址，称为重定向地址，通知浏览器立即向重定向地址发请求。

使用response.sendRedirect(String url);

## 4.2重定向特点

重定向之前，不要调用out.close();否则会报错

out.println("添加成功");//能看到这个输出！（如果不写这个输出，则响应为空白页）

out.close();

response.sendRedirect("list");//看不到重定向结果！

重定向之前，服务器会先清空response对象上缓存的数据。Servlet只允许同时发送一个响应。

out.println("添加成功");//看不到这个输出！缓存数据被清空（响应一）

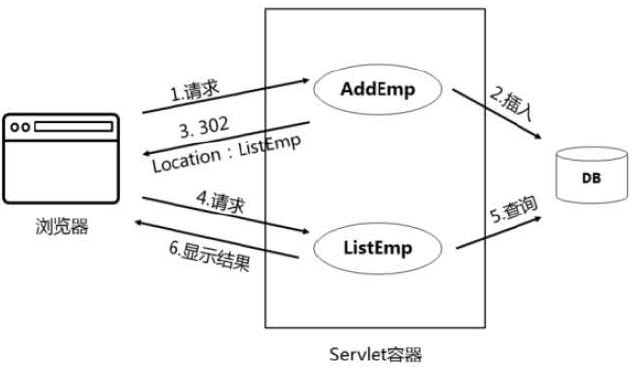
response.sendRedirect("list");//能看到重定向结果！（响应二）

out.close();

重定向的地址是任意的，前提要存在否则报404。

重定向之后，浏览器地址栏的地址会变成重定向地址。

## 4.5重定向原理图：以2.8案例为例



1. 访问AddEmp

2. 执行数据插入操作

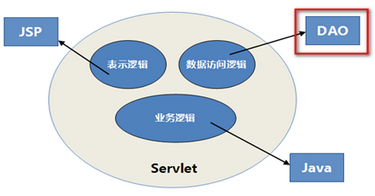
3. 执行结束后使用重定向代码发回—个数据包，里面包括

4. 302状态码和—个消息头Location

5. 浏览器收到后会立即向服务器的ListEmp发出请求五、DAO数据访问对象

## 5.1什么是DAO

Data Access Object，即封装了数据访问逻辑的一个模块。



## 5.2如何写一个DAO

**step1：实体类**

为了方便对数据库中的记录进行操作而封装的一个普通的Java类。比如，有一个表chang\_emp，可以根据表中的字段设计一个Employee类，属性名与表中的字段名相同，且有get/set方法。

**step2：声明DAO接口**

声明一系列方法，这些方法不应该涉及任何具体的数据库访问技术。

public ResultSet findAll();//错误的！因为ResultSet限制了只能是用JDBC来实现

public List<Employee> findAll() throws SQLException;//错误的！因为限制了只能是用JDBC来实现，SQLException是JDBC的错误

public List<Employee> findAll();//正确的！

public List<Employee> findAll() throws Exception;//正确的！

**step3：DAO实现**

使用具体的技术来实现DAO接口。比如，EmployeeDAOJdbcImpl类

**step4：DAO工厂**

Connection conn=DriverManager.getConnection...//DriverManager就是个工厂类，调用者通过DriverManager工厂类获得一个符合Connection接口要求的对象conn

## 5.3工厂类

封装了对象的创建细节，为调用者提供符合要求的对象。详细介绍见第六章。

# 六、工厂设计模式

## 6.1什么是工厂

工厂是一个设计模式，所谓设计模式，是为解决一类相同或者相似的问题提出的一套解决方案，并且会对这个解决方案命名，比如工厂就是一个常用的设计模式。

## 6.2使用工厂设计模式的好处

1）工厂为调用者提供符合接口要求的对象（对这对象一般称为产品），这样做的好处是，调用者不用了解对象的创建细节，这样，当对象创建过程发生改变，不会影响到调用者。

2）工厂也可以从配置文件来获得接口与其对应的实现类的配置信息。这样，当实现类发生改变时，也不用再去修改工厂类的源代码了。

## 6.3如何使用工厂模式

现在做这样一个假设，假如我们的底层数据访问不再使用JDBC，而采用Hibernate（框架技术，后续会学习，此处做了解）。

那么我们需要为EmployeeDAO增加一个实现EmployeeDAOHibernateImpl.java

并且修改之前的代码，DelEmpServlet、AddEmpServlet、ListEmpServlet、LoadEmpServlet、ModifyEmpServlet都要做相应修改（new的地方都要换成EmployeeDAOHibernateImpl对象）。

## 6.4案例：为2.8案例添加新功能，并使用DAO和工厂模式

step1：创建DBUtil.java类，专门用于连接和关闭数据库：

public static Connection getConnection() throws Exception{

Connection conn=null;

try { Class.forName("com.mysql.jdbc.Driver");

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/db\_chang?" +

"useUnicode=true&characterEncoding=utf8","root","");

} catch (Exception e) { e.printStackTrace(); throw e; }

return conn; }

public static void close(Connection conn){

if(conn!=null){ try { conn.close(); } catch (SQLException e) {

e.printStackTrace(); } } }

step2：创建Employee.java实体类：有id、name、salary、age四个属性，以及get/set方法。

step3：创建DAO接口EmployeeDAO

public List<Employee> findAll() throws Exception;

public void add(Employee e)throws Exception;

public void delete(long index)throws Exception;

public void modify(Employee e)throws Exception;

public Employee findById(long id) throws Exception;

step4：创建DAO接口的实现类EmployeeDAOJdbcImpl：

public List<Employee> findAll() throws Exception{//查找所有员工

List<Employee> employees=new ArrayList<Employee>();

Connection conn=DBUtil.getConnection();

Statement state=conn.createStatement();

ResultSet rs=state.executeQuery("select \* from chang\_emp order by id");

while(rs.next()){ int id=rs.getInt("id"); String name=rs.getString("name");

double salary=rs.getDouble("salary"); int age=rs.getInt("age");

Employee e=new Employee(); e.setId(id); e.setName(name);

e.setSalary(salary); e.setAge(age); employees.add(e); }

DBUtil.close(conn); return employees; }

public void add(Employee e) throws Exception{//添加员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement(

"insert into chang\_emp(name,salary,age) value(?,?,?)");

prep.setString(1, e.getName()); prep.setDouble(2, e.getSalary());

prep.setInt(3, e.getAge()); prep.executeUpdate(); DBUtil.close(conn); }

public void delete(long index) throws Exception{//删除员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement("delete from chang\_emp where id=?");

prep.setLong(1, index); prep.executeUpdate(); DBUtil.close(conn); }

public void modify(Employee e) throws Exception{//修改员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement(

"update chang\_emp set name=?,salary=?,age=? "where id=?");

prep.setString(1, e.getName()); prep.setDouble(2, e.getSalary());

prep.setInt(3, e.getAge()); prep.setLong(4, e.getId()); prep.executeUpdate();

DBUtil.close(conn); }

public Employee findById(long id) throws Exception {//根据id查找员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement("select \* from chang\_emp where id=?");

prep.setLong(1, id); ResultSet rs=prep.executeQuery(); Employee e=null;

if(rs.next()){ e=new Employee(); e.setId(id); e.setName(rs.getString("name"));

e.setSalary(rs.getDouble("salary")); e.setAge(rs.getInt("age")); }

DBUtil.close(conn); return e; }

step5：创建Factory.java工厂类：

public static Object getInstance(String type){ Object obj=null;

if("EmployeeDAO".equals(type)){

obj=new EmployeeDAOJdbcImpl();

/\*\* obj=new EmployeeDAOHibernateImpl();当不再使用JDBC连接数据库时，如HibernateI连接时，就需要更改每个new EmployeeDAOJdbcImpl();不方便维护。使用工厂模式就可以统一更改了 \*/ }

return obj; }

step6：修改AddEmpMySql.java中的try块，finally块也不需要了（DAO中调用了DBUtil的关闭）

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=new Employee(); e.setName(name); e.setSalary(salary);

e.setAge(age); dao.add(e); response.sendRedirect("list"); }

step7：修改ListEmpServlet.java中的try块，finally块也不需要了（DAO中调用了DBUtil的关闭）

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

List<Employee> Employees=new ArrayList(); Employees=dao.findAll();

out.println("<style>tr:hover{color:red;background:silver;}</style>"+

"<table border='1' width='60%' cellpadding='1' cellspacing='1'>");

out.println("<tr><td>id</td><td>姓名</td><td>薪水</td><td>年龄</td>" +

"<td>操作</td></tr>");

for(int i=0;i<Employees.size();i++){ Employee e=Employees.get(i);

out.println("<tr><td>"+e.getId()+"</td><td>"+e.getName()+"</td>" +

"<td>"+e.getSalary()+"</td><td>"+e.getAge()+"</td>"+

"<td><a href='del?id="+e.getId()+">删除</a>&nbsp;" +

"<a href='load?id="+e.getId()+">修改</a></td></tr>"); }

out.println("</table>");

out.println("<a href='addEmpMySql.html'>再次添加新员工</a>"); }

step8：增加DelEmpServlet.java类用于删除员工，其中service方法内容为：

response.setCharacterEncoding("utf-8"); PrintWriter out=response.getWriter();

long id=Long.parseLong(request.getParameter("id"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

dao.delete(id); response.sendRedirect("list");

}catch(Exception e){ e.printStackTrace(); out.println("删除失败！请稍后再试！"); }

step9：增加LoadEmpServlet.java类用于读取员工，其中service方法内容为：

response.setCharacterEncoding("utf-8"); PrintWriter out=response.getWriter();

long id = Long.parseLong(request.getParameter("id"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=dao.findById(id); response.setContentType("text/html;charset=utf-8");

out.println("<form action='modify?id="+id+"' method='post'>");

out.println("id："+id+"<br />");

out.println("姓名：<input name='name' value='"+e.getName()+"' /><br />");

out.println("薪水：<input name='salary' value='"+e.getSalary()+"' /><br />");

out.println("年龄：<input name='age' value='"+e.getAge()+"' /><br />");

//或out.println("<input type='hidden' name='id' value='"+id+" />");

out.println("<input type='submit' value='确定' />"); out.println("</form>");

}catch(Exception e){ e.printStackTrace(); out.println("读取失败！稍后再试！"); }

step10：增加ModifyEmpServlet.java类用于修改员工，其中service方法内容为：

response.setCharacterEncoding("utf-8"); request.setCharacterEncoding("utf-8");

PrintWriter out = response.getWriter(); long id = Long.parseLong(request.getParameter("id"));

String name=request.getParameter("name");

double salary=Double.parseDouble(request.getParameter("salary"));

int age=Integer.parseInt(request.getParameter("age"));

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=new Employee(); e.setId(id); e.setName(name); e.setSalary(salary);

e.setAge(age); dao.modify(e); response.sendRedirect("list");

} catch (Exception e) { e.printStackTrace(); out.println("修改失败！稍后重试"); }

step11：在浏览器地址栏输入http://localhost:8080/应用名/list则访问出员工列表页面。

# 七、Servlet容器处理细节

## 7.1 Servlet容器处理请求资源路径以及匹配

比如：我们在浏览器地址栏输入：http://ip:port/appname/abc.html浏览器会：

**step1**：依据ip和port建立与servlet容器之间的连接

"/appname/abc.html"作为请求资源路径发送给Servlet容器。

**step2**：Servlet容器会先假设访问的是一个Servlet，会依据应用名appname找到应用所在的文件夹，然后找到web.xml文件。

**step3**：匹配<url-pattern>

1. 精确匹配（完全匹配）：“/”、大小写、名字完全一样。

2. 通配符匹配：使用 “\*” 来匹配任意长度的字符串，比如：<url-pattern>/\*</url-pattern>

3. 后缀匹配：使用“\*.”开头，后接任意的字符串，比如：<url-pattern>\*.do</url-pattern>

注意事项：\*.do表示匹配所有以.do结尾的请求，注意不写“/”。

**step4**：如果**都不匹配**，则容器认为访问的是一个静态资源文件（比如html文件），然后容器会查找该文件，如果找到则返回，否则会返回404。

## 7.2一个Servlet处理多种请求

—般情况下，Servlet的主要作用为充当控制器的角色，即接受请求并分发给不同的资源，这时只要有—个Servlet就可以完成分发的过程，所以需要将Servlet合并。实现合并的步骤：

step1：使用后缀匹配模式修改web.xml文件，比如<url-pattern>\*.do</url-pattern>

step2：分析请求资源路径，然后依据分析的结果分别进行不同的处理

注意事项：是URI不是URL。

# 八、Servlet的生命周期

## 8.1 Servlet的生命周期的含义

Servlet容器如何去创建Servlet对象，如何给Servlet对象分配资源，如何调用Servlet对象的方法来处理请求，以及如何去销毁Servlet对象的整个过程。

Servlet生命周期的四个阶段如下：

1. 实例化

2. 初始化

3. 就绪

4. 销毁

## 8.2 Servlet的生命周期四个阶段

### 8.2.1Servlet实例化

概述：实例化指的是容器调用Servlet的构造器，创建Servlet对象。

**情况1**：

1. 容器收到请求之后才创建Servlet对象。

2. 在默认情况下，容器只会为Servlet创建唯一的一个实例

3. 多线程，有安全问题。每次请求创建一个线程，由线程去调用方法。

**情况2**：

1. 容器事先容器启动时将某些Servlet对象创建好

2. 但需要配置load-on-startup参数

3. 其中load-on-startup参数值：必须是>=0的整数，越小，优先级越高即先被实例化

4. 参数加在web.xml配置文件里的某个<servlet>标签里

注意事项：注意标签的先后顺序，是有要求的，要符合.xsd文档要求。

<servlet>

<servlet-name></servlet-name>

<servlet-class></servlet-class>

<load-on-startup>1</load-on-startup> //容器启动创建servlet实例，数越小级别高

</servlet>

### 8.2.2 Servlet初始化

**概述**：初始化指的是容器在创建好Servlet对象之后，会立即调用Servlet对象的init方法。

**init方法**：

1. init方法只会执行一次，GenericServlet已实现init方法

2. 该方法会将容器创建好的ServletConfig对象作为参数传入

3. ServletConfig对象提供了一个getInitParameter方法来访问Servlet的初始化参数

**step1**：在web.xml文件里，使用<init-param>来配置初始化参数，加在某个<servlet>标签里。

<servlet>

<init-param>

<param-name>name</param-name>

<param-value>常</param-value>

</init-param>

</servlet>

**step2**：使用String getInitParameter(String paraName)获取初始化参数。

如果GenericServlet的init方法提供的初始化操作不满足需要，可以覆盖（override） init()方法，覆盖的是不带参数的，对有参数的无影响。有参数的init方法执行时，同时无参的也被执行。有参的init方法里调用了无参的init方法，叫钩子方法。

### 8.2.3 Servlet就绪

概述：就绪指的是Servlet对象可以接受调用了，容器收到请求之后，会调用Servlet对象的service方法来处理，且可以被执行多次。

HttpServlet已经实现了service方法，该方法会依据请求类型（get/post）分别调用doGet或doPost方法。所以我们在写一个Servlet时，有两种选择：

选择一：覆盖 HttpServlet的doGet，doPost方法

选择二：覆盖 HttpServlet的service方法

### 8.2.4 Servlet销毁

销毁指的是Servlet容器在销毁Servlet对象之前，会调用destroy方法，且只会执行一次。容器根据自身的算法，删除servlet对象，可以被垃圾回收器回收。

## 8.3 Servlet生命周期图

服务器加载Servlet

服务器创建Servlet实例

调用Servlet实例的init()方法

Servlet

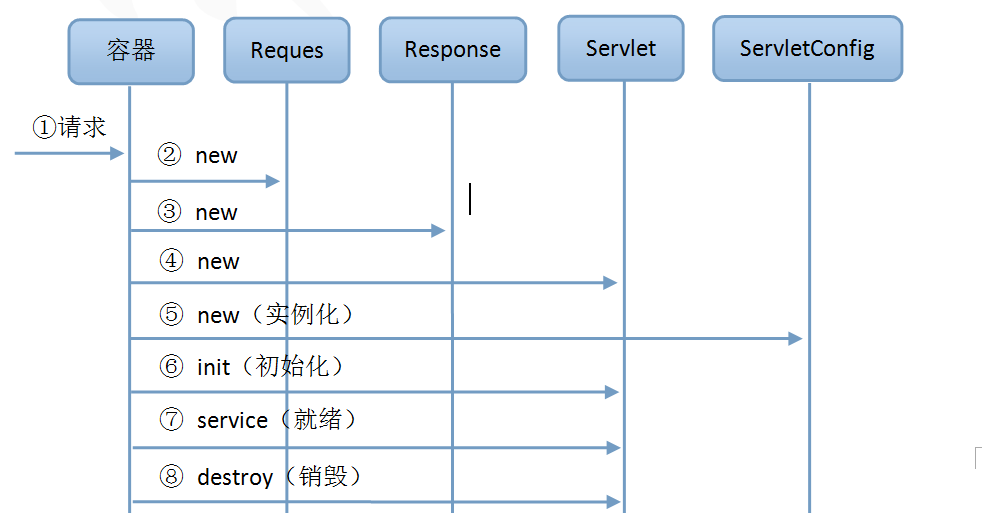
收到请求

调用service()方法

Service()方法处理请求并将输出结果返回客户端

Servlet等待下一个请求或由服务器卸载

Servlet在服务器调用destroy()方法后被卸载



## 8.4 Servlet生命周期相关的几个接口与类

Servlet接口：

1. init(ServletConfig config)//有参的init方法

2. service(ServletRequest req,ServletResponse res)

3. destroy()

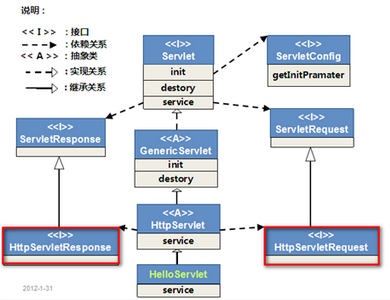
GenericServlet抽象类：实现了Servlet接口中的init，destroy方法

HttpServlet抽象类：继承了GenericServlet抽象类，实现了service方法

ServletConfig接口：String getInitParameter(String paraName)

ServletRequest接口：HttpServletRequest的父接口

ServletResponse接口：HttpServletResponse的父接口



## 8.5 Servlet上下文

### 8.5.1 ServletContext文概述

容器启动之后，会为每—个Web应用创建唯—的—个符合ServletContext接口要求的对象，该对象就是servlet上下文对象。上下文对象特点：

1. 唯—性：—个Web应用对应—个servlet

2. 一直存在：只要容器不关闭，应用没有被卸载删除servlet上下文就一直存在

### 8.5.2 获取ServletContext方式

通过GenericServlet提供的getServletContext()

通过ServletConfig提供的getServletContext()

通过HttpSession提供的getServletContext()

通过FilterConfig提供的getServletContext()

### 8.5.3 ServletContext作用及特点

ServletContext的主要作用如下：

使用setAttribute绑定数据

使用removeAttribute移除绑定数据

使用getAttribute获取绑定数据

ServletContext的主要特点如下：

servlet上下文绑定的数据可以被整个应用上的所有组件共享，并且一直可以访问

## 8.6 servlet线程安全

### 8.6.1 Servlet线程安全问题

当容器收到一个请求之后，会启动一个线程来处理该请求，如果有多个请求到达容器，并且这多个请求要访问的是同一个Servlet，则会发生多个线程调用同一个Servlet的情况，此时，就需要考虑线程安全问题了，比如，这多个线程都要修改Servlet的属性值。

### 8.6.2处理线程安全问题

加锁：使用synchronized关键字对方法或者代码块加锁。加锁会影响一些性能

SingleThreadMode接口：让一个Servlet实现SingleThreadMode接口：容器会为这样的Servlet

创建多个实例（一个线程一个实例）。因为有可能会产生过多的Servlet实例，所以，在比较大型的应用当中，尽量少用。

九、请求转发

## 9.1什么是转发

一个Web组件（Servlet/JSP）将未完成的工作通过容器转交给另外一个Web组件继续完成。常见的情况是：一个Servlet将数据处理完毕之后，转交给一个JSP去展示。

## 9.2如何转发

**step1**：**绑定数据**到requset：request里有个HashMap。

request.setAttribute(String name,Object obj);//name：绑定名。obj：绑定值。

另一个方法获取绑定值：

Object request.getAttribute(String name);//如果绑定对象的值不存在，会返回null

**step2**：获得一个转发器：url：要转发给哪一个Web组件

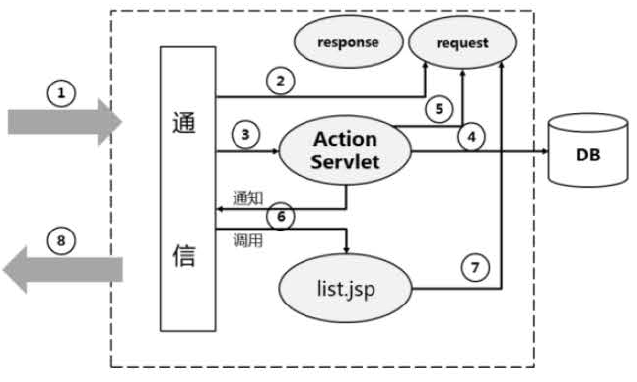
RequestDispatcher rd=request.getRequsetDispatcher(String url);

**step3**：**转发**

rd.forward(request,response);//JSP和Servlet会共享相同的请求和响应对象

**step4**：在转发的目的地，可以使用request.getAttribute方法获得绑定的数据，然后进行处理。

## 9.3 转发原理



1. 请求到达容器

2. 容器创建request和response对象

3. 将请求交给ActionServlet

4. ActionServlet通过dao获取数据

5. 将数据绑定到request当中

6. 转发到jsp页面显示，即通知容器调用jsp显示

7. 从request中获取绑定数据

8. 将jsp页面显示结果，交给浏览器解析

## 9.4转发的特点

1. 转发的目的地只能够是同一个应用内部的某个组件的地址。

2. 转发过程发生在服务器端内部的，浏览器并不知道，浏览器地址栏的地址不变

3. 转发所涉及的各个Web组件可以共享同一个request对象和response对象

## 9.5 转发和重定向

### 9.5.1 转发与重定向的区别

**重定向**是浏览器发送请求并收到响应以后，再次向一个新地址发送请求

**转发**是服务器收到请求后，为了完成响应转到一个新地址

1. 转发的目的地只能是同一个应用内部某个组件的地址，而重定向的目的地是任意的。

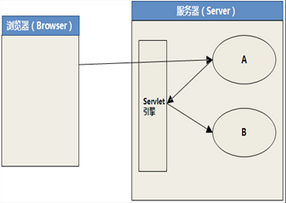
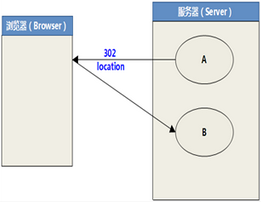
2. 转发之后，浏览器地址栏的地址不变，而重定向会变。

3. 转发所涉及各个Web组件可以共享request对象，而重定向两次请求对象数据不共享。

4. 转发是一件事情未做完，而重定向是一件事情已经做完。

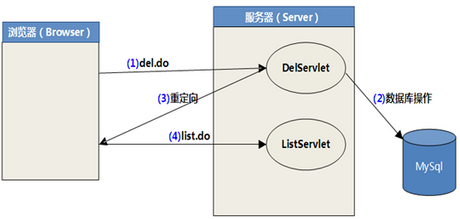
转发

重定向



### 9.5.2何时用重定向

比如用户作删除操作时，删除操作已做完，重定向访问list.do



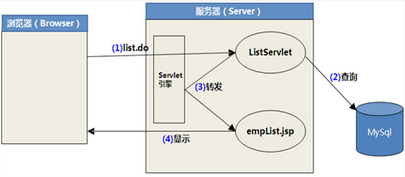
### 9.5.3何时用转发

1. 用户调用list.do

2. 由ListServlet到数据库查询数据

3. 由ListServlet将查询结果通过Servlet引擎（通信模块）转发给负责显示的empList.jsp

4. 由empList.jsp将数据通过友好的界面显示给用户，比如用户作删除操作时，删除操作已做完，重定向访问list.do



## 9.6 编程需要注意的两个问题

转发之前，先清空response对象中缓存的数据。

out.println("hello");//看不到结果

rd.forward(request, response);//能看到结果

转发之前，不能调用out.close()或着out.flush只能同时处理一个响应

out.println("hello");//看到结果

out.close();

rd.forward(request, response);//看不到结果

# 十、异常的处理

## 10.1用输出流out

catch(Exception e) {

e.printStackTrace();

out.println("系统繁忙，稍后再试！");

}

## 10.2用转发的方式

使用转发的方式，就不用输出流out了

**step1**：Servlet中catch块

catch(Exception e) {

request.setAttribute("syserror", "系统繁忙，稍后重试！");

request.getRequestDispatcher("error.jsp").forward(request, response); }

**step2**：error.jsp页面

<body style="font-size:30px;font-style:italic;">

<% String msg = (String)request.getAttribute("syserror"); %>

<%=msg%>

</body>

## 11.3让容器处理系统异常

**step1**：将异常抛给容器

catch(Exception e) {

e.printStackTrace();//写throw e;报错，子类异常个数不能大于父类

throw new ServletException(e);//这里将异常转化

}

**step2**：编写一个错误处理页面，比如system\_error.jsp

<body style="font-size:30px;font-style:italic;color:red;">

系统繁忙，稍后重试。

</body>

**step3**：配置错误处理页面（让容器知道，当捕获到了相应的异常后，应该调用哪一个页面）

<!-- 配置错误处理页面 -->

<error-page>

<exception-type>javax.servlet.ServletException</exception-type>

<location>/system\_error.jsp</location>

</error-page>

# 十二、路径问题

路径问题：即链接地址、表单提交地址、重定向、转发这四种情况下，如何写正确的路径

<a href="list.do"></a>

<form action="add.do">

response.sendRedirect("list.do");

request.getRequestDispatcher("emplist.jsp");

## 12.1 路径概述

### 12.1.1相对路径

相对路径：不以"/"开头的路径，从当前文件出发，到达目标文件所经过的路径。

1. ../ 的用法为：向上跳一级

2. 相对路径较易出错，在实际开发中建议使用绝对路径。

### 12.1.2什么是绝对路径

绝对路径：以“/”开头的路径。不以当前文件位置为起始，而是以一个固定位置为起始到达目标文件所经过的路径。

## 12.2如何写绝对路径

### 12.2.1 绝对地址分类

从**应用名开始**写：链接地址、表单提交地址、重定向地址，即”/”表示应用名前

从**应用名之后开始**写：转发地址，即”/”表示应用名后

### 12.6四种绝对路径写法

**链接地址**：从“应用名”开始写

<a href=/app01/a1.jsp">点击，访问a1.jsp</a>

表单提交地址：从“应用名”开始写

<form method="post" action="<%=request.getContextPath() %>/add">……</form>

重定向地址：从“应用名”开始写

response.sendRedirect(request.getContextPath()+"/app01/a1.jsp");

转发地址：从“应用名之后”开始写

request.getRequestDispatcher("app01/a1.jsp").forward(request, response);

## 12.4如何防止硬编码

所谓硬编码就是固定、写死在某个文件中，如JSP页面中。如果Web应用的应用名被更改，则应用会因为路径问题而无法正常运行。

解决此问题需要使用String request.getContextPath()方法来即时获取应用实际部署名。

# 

# 十二、状态管理

## 13.1状态管理

将浏览器与Web服务器之间多次交互当作一个整体来看待，并且将多次交互所涉及的数据保存下来。目前状态管理有如下两种模式：

1. 客户端状态管理技术，即将状态（也就是多次交互所涉及的数据）保存在客户端（浏览器）

2. 服务端状态管理技术，即将状态保存在Web服务器端

## 12.2 cookie

COOKIE是一种客户端的状态管理技术。当浏览器访问服务器的时候，服务器可以将少量的数据以set-cookie消息头的方式发送给浏览器，浏览器会将这些数据保存下来。当浏览器再次访问服务器时，会将之前保存的这些数据以cookie消息头的方式发送给服务器。

注意事项：有几个cookie，就有几个set-cookie消息头。

### 12.2.1创建cookie

**step1**：**创建Cookie**

Cookie c = new Cookie(String name,String value);//name：cookie的名称。value：cookie的值

**step2**：**添加Cookie**：response.addCookie(c);//调用响应对象的addCookie方法

例如：AddCookieServlet类中的service方法中写

response.setContentType("text/html;charset=utf-8");

PrintWriter out=response.getWriter();

Cookie c1=new Cookie("username","tom");

response.addCookie(c1);

Cookie c2=new Cookie("city","beijing");

response.addCookie(c2);//消息头中将会有两个set-cookie

out.println("<h1>添加Cookie成功</h1>");

out.close();

### 12.2.2查询cookie

方法：Cookie[] request.getCookies();//如果没有任何的cookie，则返回null

String cookie.getName();//返回cookie的名称

String cookie.getValue();//返回cookie的值

案例：FindCookieServlet类中的service方法中写

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter();

Cookie[] cookies=request.getCookies();

if(cookies!=null){

for(int i=0;i<cookies.length;i++){

Cookie c=cookies[i]; String name=c.getName();

String value=c.getValue();

out.println("name："+name+"，value："+value+"<br />");

}

}else{ out.println("没有cookie"); }

### 12.2.3修改Cookie

1. 获取客户端发送的所有Cookie

2. 根据name找到要修改的Cookie

3. 调用Cookie的setValue( String newValue)方法，修改该Cookie的值

4. 将修改后的Cookie加入到response发送回客户端

### 12.2.4 Cookie原理

1. 浏览器向服务器发送请求

2. 服务端执行response.addCookie语句后，会以set-cookie消息发回一段文本信息给浏览器

3. 浏览器保存信息在硬盘或内存中

4. 当再次发送请求时，会以Cookie消息头的形式，自动将保存的文本信息发送到服务器

5. 服务器使用request.getCookies()方法获取请求中可能携带的Cookies的值

### 12.2.5 cookie的生存时间

Cookie默认情况下，浏览器会将cookie保存在内存里，只要浏览器不关闭，cookie会一直存在。

**Cookie的生成与销毁**

1. 启动FireFox后，操作系统会为浏览器开辟一块内存空间

2. 浏览器（FireFox）访问服务器

3. 服务器生成消息头setCookie发回给浏览器

4. 浏览器将服务器生成的消息头保存到内存区域

5. 当浏览器关闭，系统回收为浏览器分配的内存，cookie也相应消失

**设置生存时间**

使用cookie.setMaxAge(int seconds)方法，来设置生存时间

1. 当seconds>0时，表示cookie在硬盘上保存的最大时间，

2. 当seconds<0时，缺省值，浏览器会将cookie保存在内存里，浏览器关闭即删除

3. 当seconds=0时，立即删除cookie

### 12.2.6 编码问题

cookie的值或者名称只允许合法的ASCII码字符串，如果是中文，需要将中文转换成ASCII码字符串。

需要用到的两个方法

String URLEncoder.encode(String str,String code);

String URLDecoder.decode(String str,String code);

例如：上例代码修改如下

Cookie c1=new Cookie("username",URLEncoder.encode("常","utf-8"));

out.println("name："+name+"，value："+URLDecoder.decode(value,"utf-8")+"<br />");

### 12.2.7 cookie的路径问题

**路径问题**指的是：浏览器在向服务器上的某个地址发请求时，会比较请求地址与cookie的路径是否匹配，只有匹配的cookie才会发送。

1. cookie有一个默认的路径，值等于创建该cookie的组件的路径，比如：/web06\_Cookie/

app01/addCookie.jsp创建了一个cookie，则该cookie默认的路径为：/web06\_Cookie/app01。

2. 匹配规则：只有当访问的地址是cookie的路径或着是其子路径时，浏览器才会将这个cookie进行发送。

3. 修改cookie的默认路径，可以调用cookie.setPath(String path)方法，一般我们会设置cookie.setPath("/appname")，这样，应用内部的某个组件所添加的cookie可以被该应用内部的其他组件访问到。

### 12.2.8 cookie的限制

1. cookie可以被用户禁止。

2. cookie保存在浏览器端，所以cookie不安全。对于一些敏感的数据，需要加密处理。

3. cookie只能够保存少量的数据（大约是4K左右）。

4. cookie的个数也有限制

5. cookie只能够保存字符串，对象、集合、数组都不能保存。

### 12.2.9案例：写一个CookieUtil

## 12.3 session

SESSION是服务器端的状态管理技术:

1. 当浏览器访问服务器时，服务器会创建一个session对象

2. 该对象有一个唯一的id号，称之为sessionId

3. 接下来，服务器在默认情况下，会使用set-cookie消息头将这个sessionId发送浏览器

4. 浏览器会将这个sessionId保存下来，内存中，因为指定生存时间

5. 当浏览器再次访问服务器时，会将sessionId使用cookie消息头发送给服务器

6. 服务器依据这个sessionId就可以找到之前创建的session对象

7. 用户与服务器之间的多次交互叫一次会话。

### 12.3.1获取session对象

**方式一**：HttpSession s = request.getSession(boolean flag);

HttpSession是一个接口，返回一个符合要求的对象。

当**flag=true**时：

服务器会先检查请求当中是否有sessionId

1. 如果没有sessionId则创建一个session对象

2. 如果有sessionId，则服务器会依据sessionId查找对应的session对象，

2.1 如果找到了，则返回session对象

2.2 根据sessionId找不到，则服务器会创建一个新的session对象。

当**flag＝false**时：

服务器会先检查请求当中是否有sessionId

1. 如果没有sessionId则返回null。

2. 如果有sessionId，则服务器会依据sessionId查找对应的session对象

2.1 如果找到了，则返回session对象。

2.2 根据sessionId找不到，则服务器返回null。

**方式二**：HttpSession s=request.getSession();等价于HttpSession s=request.getSession(true);

### 12.3.2 HttpSession接口中提供的常用方法

1. String getId()：获得sessionId

2. setAttribute(String name,Object obj)：绑定一个对象到session对象上

3. getAttribute(String name)：获得绑定对象，如果不存在则返回null

4. removeAttribute(String name)：解除绑定（request也有）

### 12.3.3 设置session的超时

服务器会将空闲时间过长的session对象删除，大部分服务器默认的session超时限制是30分钟。修改超时限制：可以修改这个默认的超时限制。

**方式一**：通过TomCat的web.xml文件

在Tomcat下的web.xml配置信息，影响Tomcat服务器下的所有应用，修改之后，需要重新启动Tomcat服务器。也可以，修改某个应用的web.xml。

<session-config>

<session-timeout>30</session-timeout>

</session-config>

**方式二**：通过编程方式修改

也可以通过编程的方式来修改超时的限制：setMaxInactiveInterval(int seconds);

**方式三**：立即删除session：invalidate();

上面操作，实际是清空session对象的内容，然后可以给其他人继续使用该session对象

### 12.3.4用户禁止cookie

如果用户禁止cookie，服务器仍然会将sessionId以cookie的方式发送给浏览器，但是，浏览器不再保存这个cookie（即sessionId）。

如果想要继续使用session，需要采取其他方式来实现sessionId的跟踪，如可以使用url重写来实现sessionId的跟踪。

### 12.3.5 url重写

**什么是url重写**

浏览器在访问服务器上的某个地址时，不能够直接写这个组件（Servlet/JSP）的地址，而应该使用服务器生成的包含有sessionId的地址。

当一个新页面有个链接时<a href="SomeServlet">someServlet</a>是错误的，点刷新是不会计数的，因为cookie被禁止了。

应该是<a href="<%=response.encodeURL("SomeServlet")%>">someServlet</a>此时去点击刷新count会自动加1的。

**注意事项**：当用户禁止cookie时，encodeURL方法会在“SomeServlet”后面添加sessionId，不禁止不会有。

**如何进行url重写**

response.encodeURL(String utl);//encodeURL方法用在链接地址、表单提交地址。

例如：<form action="<%=response.encodeURL("SomeServlet")%>">

注意事项：还有一个encodeUrl()是个老方法，内部实现和encodeURL一模一样！

response.encodeRedirectURL(String url);//用于重定向

例如：response.sendRedirect(response.encodeRedirectURL("list.do"));

转发不用考虑，是服务器内部，不是浏览器访问服务器。

### 12.3.6 session特点

**(1) session的优点**

1. session比较安全（相对于cookie）因为将所有状态都写在服务器端。

2. session能够保存的数据类型更加丰富（cookie只能保存字符串）。

3. session能够保存的数据大小更大（cookie只能保存大约4K左右的数据）

4. session没有被禁止的问题，也就不需要url重写（cookie可以被用户禁止）。

**(2) session的缺点**

1. session将数据放在服务器端，占用服务器资源，如果用户量过大会影响服务器性能

可以将session存到数据库或文件中（钝化），采取最近最少使用原则。当再次使用session时，从数据库或文件中激活

2. session默认情况下，会将sessionId以cookie的方式发送给浏览器，浏览器会将session保存到内存中，如果浏览器关闭，浏览器发请求时就没有sessionId，服务器端的session对象就找不到了。

### 12.3.6案例：session验证和访问次数

**session验证**：session验证经常用于保护一些需要登录之后才能访问的资源。只有登录成功以后，才能访问main.jsp

step1：登录成功以后，绑定一些数据到session对象上。

例如：session.setAttribute("user",user);

step2：对于需要保护的资源，添加session验证的代码。例如：为main.jsp设置session验证

<% Object obj=session.getAttribute("user");

if(obj==null){

//没有登录成功，或因为session超时，服务器已经删除了之前的session对象

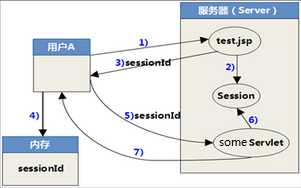
response.sendRedirect("login.jsp"); return;

}

%>

注意事项：若不写return，则该页面中body标签中的内容，虽然看不到，但还是会执行！

**session访问次数**



SomeServlet中service方法

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter();

HttpSession session=request.getSession();//获得一个session对象

//设置超时限制，注意：若不到10秒刷新则又重新开始计时

//session.setMaxInactiveInterval(10);

String sessionId=session.getId();//获得sessionId

System.out.println("sessionId："+sessionId);

Integer count=(Integer)session.getAttribute("count");//绑定必须是个对象，所以用包装类

if(count==null){

count=1;//是第一次访问，则设置count=1

}else{

count++;//不是第一次访问，则原有值基础上加1

}

session.setAttribute("count", count);

out.println("<h1>你是第："+count+"次访问</h1>");

//session.invalidate();//立即删除session对象

out.close();

## 12.3.8 案例：验证码

**验证码介绍**

**验证码作用**：防止恶意攻击，不停的注册，不停的登陆来占用服务器资源

**验证码原理**：只有发送请求时，才将随机产生的字符组合绘制到图片，返回

**验证码生成**：绘制一张从白纸到有内容的图片过程

1. 指定大小区域，获取一张空白图片

2. 针对这张纸获取画笔

3. 设置画笔颜色

4. 绘制背景

5. 更改颜色

6. 绘制内容

6. 绘制完成，保存图片

**验证码请求原理**

1. 客户端请求JSP页面

2. 容器自动创建session

3. 容器将img标记的页面返回给客户端，但图片不会返回给客户端

4. 浏览器收到页面，逐行解析。遇到img标记时，根据src地址发出请求

5. 到达codeServlet

5.1 绘制图片

5.2 产生随机字符串

5.3 将验证码的随机自负床存储到session对象，返回图片给客户端

6. 用户填写表单，点击提交

7. 在login.do中的处理逻辑，获取用户填写的表单数据

8. 将用户输入的验证码与session中存储的验证码比较

8.1 比对失败，绑定提示信息转发login.jsp

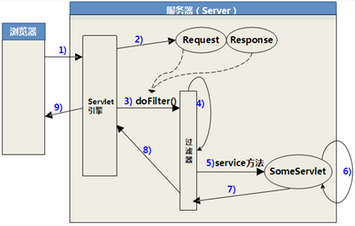
8.2 比对成功，比对用户名和密码

# 十三、过滤器

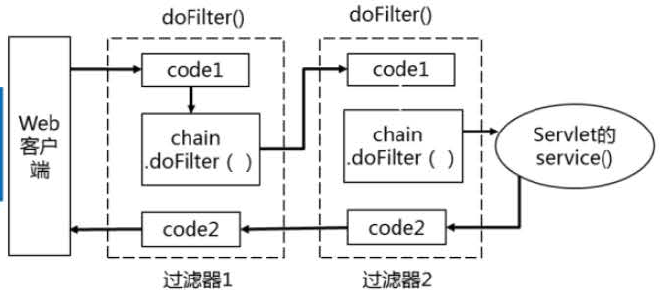
## 13.1什么是过滤器

### 13.1.1过滤器介绍

Servlet规范当中定义的一种特殊的组件，可以拦截Servlet容器的调用过程并进行相应的处理。即过滤器是一种可以拦截请求和响应，并对请求做出一些修改或判断功能的可拔插组件。某个过滤器只有一个实例，即单例模式。



### 13.1.2 过滤器原理



1. 浏览器发送请求到服务端

2. 服务端接收到求情后，会创建request和response对象

3. 当请求的资源路径包含有过滤器时，则由过滤器对请求做出相应处理

3.1 当不满足过滤器放行条件，则由过滤器直接响应客户端

3.2 当满足过滤器放行条件，则将response和request传给链对象，将请求转给后续组件

4. 当所有的过滤器都满足放行条件，则将请求传给servlet处理

5. 当最后的servlet处理结束后，分别执行每个过滤器链后面的一段代码

7. 之行结束后，将最终的结果返回给客户端

## 13.2编写过滤器

1. 写一个Java类，实现Filter接口

2. 在doFilter方法里面，编写处理逻辑

3. 配置过滤器

注意事项：Filter的xml配置在Servlet的xml配置前

## 13.3过滤器的优先级和初始化参数

### 13.3.1 过滤器优先级

若有多个过滤器都满足过滤的条件，则容器会依据**<filter-mapping>的先后顺序**来调用过滤器。

### 13.3.2 过滤器初始化参数

1. 在web.xml中的filter节点内，使用<init-param>配置初始化参数

<filter>

<filter-name>CommentFilter</filter-name>

<filter-class>web.CommentFilter</filter-class>

<init-param>

<param-name>word</param-name>

<param-value>共党</param-value>

</init-param>

</filter>

2. 调用FilterConfig.getInitParameter(String parameter)

**public** **void** init(FilterConfig arg0) **throws** ServletException {

keyword = arg0.getInitParameter("word");

}

## 13.4过滤器优点

1. 可以实现代码的“可插拔性”，增加或减少某个模块，不会影响程序的正常运行。

2. 可以将多个组件相同的处理逻辑集中写在过滤器里面，方便代码的维护。

如：四个Web组件都需要session验证，把验证写在一个过滤器即可。配置中写“/\*”，拦截所有请求。

# 十四、监听器

## 14.1监听器概述

Servlet规范当中定义的一种特殊的组件，用来监听容器产生的事件并进行处理。容器会产生两大类事件：

1. **生命周期**相关的事件：

容器在创建或者销毁Requset对象时产生的事件，ServletRequestListener

容器在创建或者销毁Session对象时产生的事件，HttpSessionListener

容器在创建或者销毁ServletContext对象时产生的事件，ServletContextListener

2. **绑定**相关的事件：

容器调用了Request对象的setAttribute、removeAttribute ServletRequestAttributeListener

容器调用了Session对象的setAttribute、removeAttribute HttpSessionAttributeListener

容器调用了ServletContext的setAttribute、removeAttribute ServletContextAttributeListener

## 14.2编写监听器

1. 写一个Java类，实现监听器接口，依据监听的事件类型来选择对应的接口2. 在监听器接2. 口定义的方法里面，编写处理逻辑

3. 配置监听器

## 14.3 ServletContext Servlet上下文

容器在启动的时候，会为每一个应用创建唯一的一个符合ServletContext接口要求的对象（Servlet上下文），该对象一直存在，只有非容器关闭时对象被销毁。

### 14.3.1获得Servlet上下文

GenericServlet.getServletContext();

通过GenericServlet抽象类获得上下文,其实也是调用了ServletConfig的getServletContext方法

HttpSession.getServletContext();//通过Session获得上下文

ServletConfig.getServletContext();//通过ServletConfig接口获得上下文

FilterConfig.getServletContext();//通过过滤器获得上下文

### 14.3.2 Servlet上下文作用

1. **绑定数据**：setAttribute、removeAttribute、getAttribute

Request、Session、ServletContext都提供了绑定数据的相关的三个方法，如果都满足使用的条件，应该优先使用生命周期短的：Request < Session < ServletContext

**Request对象上**绑定的数据只有同一个请求所涉及的各个Web组件可以共享，例如：一个Servlet将数据绑定到Request，然后转发到一个jsp。请求先交给过滤器来处理，然后调用Servlet。

**Session对象上**绑定的数据是同一个会话所涉及的各个Web组件可以共享。

**ServletContext对象**绑定的数据是公开的，谁都可以访问，而且随时可访问。

2. **访问全局的初始化参数**

使用**<context-param>**配置的初始化参数，调用String getInitParameter(String paramName)方法，可以被同一个应用中的所有的Servlet、Filter共享。

3. 依据逻辑路径（path）获得实际部署时的物理路径

String getRealPath(String path);

<context-param><!-- **全局初始化参数**，放在servlet前面 -->

<param-name>company</param-name>

<param-value>chang全局</param-value>

</context-param>

<servlet><!-- **局部初始化参数**，放在某个servlet里面 -->

<servlet-name>A</servlet-name>

<servlet-class>web.A</servlet-class>

<init-param>

<param-name>company</param-name>

<param-value>chang局部</param-value>

</init-param>

</servlet>

# 

# 15、上传文件

## 15.1如何上传文件

1. 给表单设置enctype="multipart/form-data"，这是http协议的要求。并且，表单只能使用post方式来提交。

2. 在服务器端，不能使用request.getParameter方法来获得参数值。此时，需要调用request.getInputStream获得一个InputStream字节流。然后分析这个流来获得数据，一般使用一些工具来分析这个流，比如：apache提供的commons-fileupload-1.2.1.jar。

## 15.2案例：上传文件

step1：将commons-fileupload-1.2.1.jar和commons-io-1.4.jar包导入

step2：创建form.jsp页面

<form action="fileupload" method="post" enctype="multipart/form-data">

<fiedset>

<legend>上传文件</legend>

用户名：<input name="username" /><br />

照片：<input type="file" name="file1"/>

<input type="submit" value="确定" />

</fieldset>

</form>

step3：FileUploadServlet类中service方法

DiskFileItemFactory dfif = new DiskFileItemFactory();

ServletFileUpload sfu = new ServletFileUpload(dfif);

try { List<FileItem> items = sfu.parseRequest(request);

//只需要遍历items集合就可以访问表单中的每一个表单域的数据。

for(int i=0;i<items.size();i++){ FileItem curr = items.get(i);

if(curr.isFormField()){//如果是普通表单域

String fieldname = curr.getFieldName();

System.out.println("fieldname:" + fieldname);

String username = curr.getString();

System.out.println("username:" + username);

}else{//如果是上传文件域

ServletContext sctx = getServletContext();

//依据逻辑路径获得实际部署时的物理路径

String path = sctx.getRealPath("upload");

System.out.println("path:" + path);

String filename = curr.getName();//获得文件名

File file = new File(path + "//" + filename);

curr.write(file);//注意，windows用"\\"，Linux用“//”

}

}

} catch (Exception e) { e.printStackTrace(); }

1. 创建一个DiskFileItemFactory工厂类对象，该对象为解析器提供了解析时的缺省配置

2. 创建解析器

3. 使用解析器来解析，解析器会调用request.getInputStream获得一个流

4. 然后分析这个流，并且将分析的结果封装到FileItem对象里面

5. 一个FileItem对象封装了一个表单域中的所有数据