J2EE

主讲：崔译

一、Servlet简介

在 jdk1.2 版本时，提出的 用于 扩展 web 服务功能的 组件技术

Servlet 解决的问题

封装了一些基础操作， 用于实现网站开发。

- 数据包解析
- 多线程操作
- Socket/ IO 操作

二、HelloWorld

1、准备工作

1-1 下载并安装`Tomcat`

在Tomcat官网下载,下载后解压缩。

1-2 为MyEclipse添加Tomcat

> window --- preferences --> tomcat ---> tomcat7.x ---> enable --> browse(定位到tomcat根目录)

2、配置版（J2EE 5.0/ Web2.5）

2-1 编写Java类

```
package day01;

public class HelloWorld1 extends  HttpServlet{
    @Override
    protected void service(HttpServletRequest request,
                           HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.print("<h1>HelloWorld</h1>");
    }
}
```

2-2 编写web.xml

> 项目 --> WebRoot --> WEB-INF --> web.xml

```
<servlet>
  <servlet-name>abc</servlet-name>
  <servlet-class>day01.HelloWorld1</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>abc</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

2-3 发布（部署）项目

一次性的工作，只需要部署一次

点击

. 选择要部署的项目，点击add，选择对应的Tomcat，finish

2-4 启动Tomcat

点击

。的箭头，选择对应tomcat，start

在MyEclipse关闭前，重新启动Tomcat，只需要点在  按钮上

2-5 访问Servlet

浏览器地址栏输入

> localhost:8080/05-web/URL-PATTERN
>
> 127.0.0.1:8080/05-web/URL-PATTERN 本地回环地址（本机）
>
> 192.168.7.8:8080/05-web/URL-PATTERN

3、注解版（J2EE 6.0 / web3.0）

取消了Web.xml的配置

3-1、编写java类

```
@WebServlet(urlPatterns="/hello2")
public class HelloWorld2 extends HttpServlet{

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.print("<a href='http://bbs.itany.com'>Hello Kitty</a>");
    }
}
```

三、再探HelloWorld

1、Tomcat

1-1 介绍

- 是一个Servlet容器（遵循Servlet规范的软件）
- 是一个运行在服务器上的软件
- 实现了基础服务（网络通信，IO，多线程）
- **Tomcat可以独立存在，并独立运行**

1-2 安装

1. 下载压缩包
2. 解压缩到合适位置
3. 配置环境变量
    1. tomcat 依赖的环境变量

        `JAVA_HOME` / `CLASSPATH`
    2. tomcat 自己的环境变量

        `path` 配置到 `TOMCAT安装目录/bin`

        `CATALINA_HOME` 配置到 `TOMCAT安装目录`
4. 启动Tomcat
    1. `catalina.sh/bat run` 会阻塞线程
    2. `startup.bat` | `./startup.sh` 会重新打开新的终端，当前线程不阻塞
5. 停止服务器
    1. `ctrl+c`
    2. `shutdown.bat` | `shutdown.sh`

1-3 Address already in use

地址被使用（端口被占用）

- 打开 Tomcat安装目录/bin
- 鼠标右键--打开终端
- 在终端中（只需要做一次）执行命令 `chmod 777 *`

- 将`shutdown.sh`拽入终端，（确保终端获取到焦点）回车

1-4 目录结构（重点）

```
tomcat安装目录
|--- bin 所有的可执行文件
|--- lib  tomcat 运行时所依赖的jar包
|--- conf  tomcat 的配置文件所在目录
    |--- server.xml  tomcat的配置文件（包括tomcat端口配置）
|--- webapps  所有发布的项目
    locahost:8080/文件路径或者urlPattern
    tomcat安装目录/webapps/文件路径或者urlPattern
|--- work
    |--- JSP 对应的java类和class文件
|--- temp
    |--- 临时文件夹
```

2、JavaWeb 项目

2-1 目录结构(重点)

```
项目
|---src                    java代码
|---JRE System Library     jdk
|---Java EE 6 Libraries    JavaWeb(Servlet组件) 对应的jar
|---WebRoot                web项目 根目录
    |---js                 放js文件
    |---css                放css文件
    |---imgs               放图片
    |---xxxx               插件
    |---WEB-INF            私有目录(该目录中内容无法通过浏览器直接访问)
        |---lib            jar包（不需要build path...）
        |---web.xml        项目的配置文件
```

2-2 发布到`webapps`后的目录结构

```
项目(项目结构和MyEclipse中项目WebRoot文件夹的结构是一致的)
    |---js                 放js文件
    |---css                放css文件
    |---imgs               放图片
    |---xxxx               插件
    |---WEB-INF            私有目录(该目录中内容无法通过浏览器直接访问)
        |---classes        src下的java类对应的class文件（包括包结构和配置文件）
        |---lib            jar包（不需要build path...）
        |---web.xml        项目的配置文件
```

2-3 web项目的最终形态

- J2SE 项目的最终形态 `jar包`
- J2EE 项目的最终形态 `war包`

四、HttpServletRequest

1、基本信息

请求对象，该对象由服务器容器创建（Tomcat），封装了请求信息（浏览器F12--NetWork中查看）。

- 请求头（request Headers）
- 请求者信息（ip，端口，主机名等等）
- **请求参数**
- 获取项目名（项目的根目录）

```
String contextPath = request.getContextPath();
System.out.println(contextPath);
```

2、请求参数

2-1 请求发送方式

```
// 常用（记住）
1. 直接在浏览器地址栏中输入请求地址
2. a 标签的 href 属性
3. form 标签的action 属性
4. js 中的 location.href
// 其他
5. img src
6. script src
7. link href
8. 等等.....
```

2-2 参数的传递方式

1. 在请求地址中携带参数

   `localhost:8080/ums/add?参数名=参数值（不要引号）&参数名=参数值&.....`

2. `form`表单提交

   对于`form`表单，**所有的** 有`name`属性的表单组件都会作为请求参数传递

   除了radio 和 checkbox

   - radio 必须有一个默认选中
   - radio 和 checkbox 必须有value
   - 对于 checkbox Servlet中要做非空保护

2-3 Servlet中参数的获取方式

```
String param = request.getParameter("请求参数名");
// 用于获取同一个参数名对应多个参数值的情况（注意：非空保护）
String[] arr = request.getParameterValues("请求参数名");
```

3、请求方式

Http 请求方式有很多种，包括: `get,post,delete,test,head.....`

- post
  - 发送方式：`form method=post` 或者 `ajax`
  - 请求参数不可见 为 form data
  - 可以实现 文件上传
  - 请求参数长度没有限制
- get
  - 发送方式：除了post的两种，其他全部都是get
  - 请求参数在url上，`query string patameter` url 编码的参数
  - 请求参数的长度有限制
  - 请求参数的编码是 `iso-8859-1`('不支持'中文)

能post，坚决不get

五、乱码问题

- 项目编码（文件的编码）

  项目（文件）右键
- 页面的编码

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

- 请求参数
  - POST

```
// 在获取请求参数之前
request.setCharacterEncoding("utf-8");
```

  - GET
    - 在java代码中转码

```
new String(username.getBytes("iso-8859-1"), "utf-8");
```

    - 修改Tomcat 配置

```
<!--server.xml-->
 <Connector connectionTimeout="20000" port="8080"
            protocol="HTTP/1.1" redirectPort="8443"
            URIEncoding="UTF-8"
/>
```

- 数据库建表

```
create table t_xx()engin=Innodb default charset=utf8
-- show create table t_xxx 查看建表语句
```

- 导入原始数据有乱码
  - 使用程序导入数据
  - 使用第三方数据库管理软件（Navicat）
- JDBC URL

```
jdbc:mysql://localhost:3306/ums?useUnicode=true&characterEncoding=utf8
```

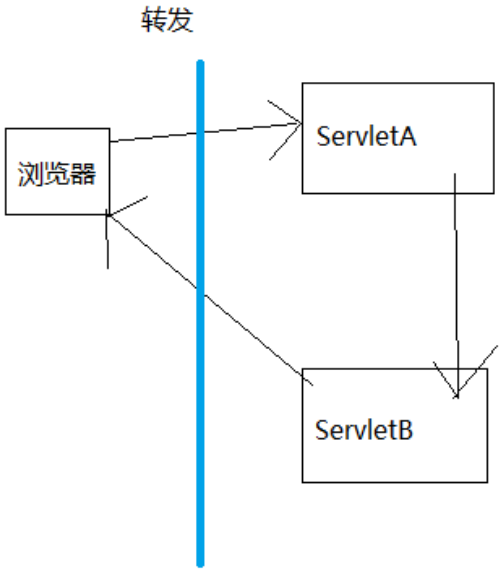
- 响应字符集

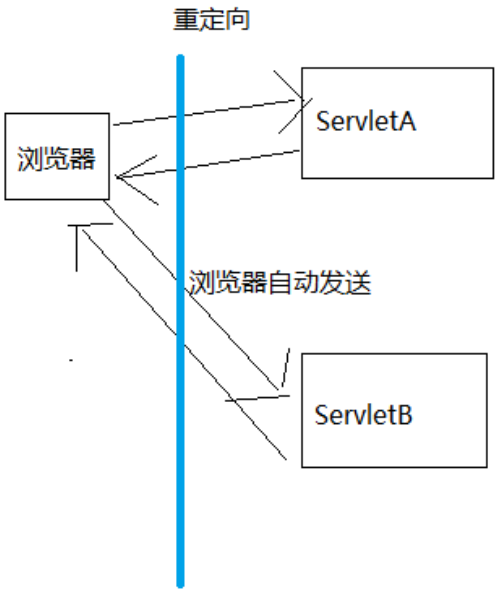
```
response.setCharacterEncoding("utf-8");
response.setContentType("text/html;charset=utf8");
```

六、请求转发

一个Servlet执行完，访问另一个Servlet

1、转发 和 重定向





2、区别

转发	重定向
对于浏览器而言，是一次请求	对于浏览器而言是两次请求
浏览器地址栏不改变	地址栏发生改变
两个Servlet 共享 request 和 response	每次请求都是新的request和response
可以访问WEB-INF中的内容	不可以访问WEB-INF

3、实现方式

```
// 转发到ServletB
request.getRequestDispatcher("/WEB-INF/a.html")
        .forward(request, response);
// 重定向到ServletB
response.sendRedirect(request.getContextPath()+"/s2");
```

4、转发 or 重定向

取决于事情是否做完，做完了 重定向，没做完转发

七、Http Status

Http 的响应状态码，是100~600的数字，不连续

- 200 成功响应
- 404 not found 资源未找到
- 500 服务器内部错误（代码出异常）
- 304 Not Modified 未修改（从浏览器缓存中获取的数据）
- 302 重定向

>400的状态都是非正常状态

八、路径问题

项目中所有的路径使用 绝对路径

在请求中，所有以 `/` 开头的路径都是绝对路径

- 在**HTML 代码**中的路径： 是相对于 `localhost:8080`
- 对于重定向： `localhost:8080`
- 对于转发： `localhost:8080/项目名`

九、Servlet API

1、API 方法

方法	作用	备注
service	处理请求	
doGet	处理get请求	当service方法同时存在，执行service方法
doPost	处理post请求	当service方法同时存在，执行service方法
init	初始化方法	Servlet 初始化时调用
destroy	销毁方法	当服务器正常关闭时，调用

2、关于init方法

- 该方法只会执行一次
- servlet 是 **单例**的
- servlet 线程不安全（**不要在Servlet中写属性，除非你确切知道在做什么**）
- 默认在 **第一次** 访问该Servlet 时，调用该方法

3、Servlet 创建时机的修改

方式1

```
<servlet>
  <servlet-name>abc</servlet-name>
  <servlet-class>day01.HelloWorld1</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

方式2

```
// 在服务器启动的时候，创建该Servlet
// loadOnStartup 值是一个数值，>0 都是在服务器启动的时候创建
// loadOnStartup 的数值 决定了 多个Servlet的创建顺序，值越小，越先创建
@WebServlet(urlPatterns="/api",loadOnStartup=1)
```

- loadOnStartup 值是一个数值，>0 都是在服务器启动的时候创建
- loadOnStartup 的数值 决定了 多个Servlet的创建顺序，值越小，越先创建

4、关于ServletConfig

4-1 配置文件

```
  <servlet>
    <servlet-name>api</servlet-name>
    <servlet-class>day02.ServletAPI</servlet-class>
    <!--配置初始化参数 -->
    <init-param>
        <param-name>abc</param-name>
        <param-value>123</param-value>
    </init-param>
    <init-param>
        <param-name>aaa</param-name>
        <param-value>456</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>api</servlet-name>
    <url-pattern>/api</url-pattern>
  </servlet-mapping>
```

4-2 注解

```
@WebServlet(urlPatterns="/api",loadOnStartup=1,
        initParams={@WebInitParam(name="abc",value="def"),
                    @WebInitParam(name="aaa",value="999")}
)
```

4-3 注解 + 配置

J2EE 允许 同时写 WebServlet 注解 和 配置 web.xml中的servlet节点

（**此时，web.xml中的servlet-name 必须写 对应Servlet的包名.类名**）

web.xml 中的配置会覆盖注解配置

```
<servlet>
  <!--servlet-name 必须写 对应Servlet的包名.类名-->
  <servlet-name>day02.ServletAPI</servlet-name>
  <servlet-class>day02.ServletAPI</servlet-class>
  <init-param>
    <param-name>abc</param-name>
    <param-value>aaaaaa</param-value>
  </init-param>
  <init-param>
    <param-name>aaa</param-name>
    <param-value>bbbbbbbbb</param-value>
  </init-param>
</servlet>
```

```
@WebServlet(urlPatterns="/api",loadOnStartup=1)
public class ServletAPI extends HttpServlet{}
```

4-4 参数获取方式

```
@Override
public void init(ServletConfig config) throws ServletException {
  // 获取初始化参数
  String abc = config.getInitParameter("abc");
  String aaa = config.getInitParameter("aaa");
  System.out.println(abc);
  System.out.println(aaa);
}
```

十、JSP

1、简介

- 本质上是 **Servlet** ，是Servlet的另一种形态
- **浏览器发送请求，访问的不是JSP，而是对应的Servlet**
- JSP 是运行在 **服务器** 端的
- 文件以 `.jsp` 结尾
- 位于 `WebRoot` 或者 `WEB-INF（常用）` 或其子目录中
- 使用 `Tomcat` 容器进行访问（localhost:8080/...）
- **不需要配置任何信息**
- HTML + CSS + JS + `java代码（实际开发不允许）` + 标签库标签 + 表达式（EL）

2、HelloWorld

```jsp
<%@ page language="java"  pageEncoding="UTF-8"%>

<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP '01.jsp' starting page</title>
  </head>

  <body>
    HelloWorld <br>
  </body>
</html>
```

3、HelloWorld-2

jsp 在`WEB-INF`中

```java
@WebServlet("/showHello")
public class Hello extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse
      response)throws ServletException, IOException {
        request.getRequestDispatcher("/WEB-INF/jsp/01.jsp")
                .forward(request, response);
    }

}
```

4、再探HelloWorld

对于`WebRoot/day03/01.jsp`,在`TOMCAT_HOME/work/catalina/localhost/项目名/org/apache/jsp/自己创建的目录/`

```
自己建的目录
|----- 01_jsp.java
|------01_jsp.class
```

5、组成

5-1 JSP 指令

指令： 以`<%@`开头 ， 以`%>`结尾的一种语法，在JSP引擎，将JSP转换为Java类的时候，要额外做的一些事情

```jsp
<!--语法是java，导入java.util.*,页面字符集是utf-8-->
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
```

后期会使用指令 导入 标签库

5-2 JSP 动作

动作：以`<jsp:`开头，以`>`结尾的一种语法

```
<!-- 将另一个JSP引入当前JSP ， page 对应jsp的路径，只能写相对路径
    引入的jsp 可以共享 父jsp中的css 和 js等
-->
<jsp:include page="template.jsp"></jsp:include>
<!-- 当遇到该动作时，浏览器会自动跳转到page对应的页面 -->
<jsp:forward page="template.jsp"></jsp:forward>
```

5-3 JSP 脚本

在JSP中写的java代码

转换规则

JSP	转换成的Servlet
HTML+CSS+JS	out.write(HTML+CSS+JS);
<% Java代码 %>	Java代码 原样
<%=Java表达式 %>	out.write(表达式的值)
<%! java代码 %>	用于声明Servlet属性

十一、JSP的九大内置对象

1、简介

内置对象	对应java类	作用
request	HttpServletRequest	请求对象，封装了请求数据（请求头和请求参数）
response	HttpServletResponse	响应对象
out	JspWriter	输出流
config	ServletConfig	获取Servlet初始化参数
page	Object	当前JSP对应的java类的this
exception	? extends Throwable	出现在isErrorPage=true中，异常对象
session	------	----
application	ServletContext	应用程序
pageContext	PageContext	页面上下文，用于获取其他内置对象

2、`exception`对象

只能出现在`isErrorPage=true`的jsp中

03.jsp

```
 <body>
    <%int i = 1 / 0; %>
 </body>
```

err.jsp

```
<%@ page language="java" pageEncoding="utf-8" isErrorPage="true"%>
<!DOCTYPE>
<html>
  <head>
    <title>My JSP 'err.jsp' starting page</title>
  </head>
  <body>
        <font color="red"><%=exception.getMessage() %></font>
  </body>
</html>
```

web.xml

```
<error-page>
  <error-code>500</error-code>
  <location>/day03/err.jsp</location>
</error-page>
```

十二、JSP的四大作用域对象

1、简介

所谓的作用域对象，指的是9个内置对象种的4个对象，用于解决在`Servlet`与`Servlet`

之间共享数据的问题

> jsp ---> servlet 表单提交，点击删除链接、修改，
>
> servlet ---> jsp 用于查询操作，转发
>
> jsp ---> jsp 点击添加按钮
>
> servlet ---> servlet 添加、删除、修改完成--》 查询servlet 重定向

要写java代码（访问数据库），进servlet ， 否则进jsp

2、相关方法

- `void setAttribute(String key,Object value)`

- `Object getAttribute(String key)`

这两个方法是4个作用域对象都具有的方法

3、`pageContext`

作用域范围： 当前jsp

一般不作为作用域对象使用，用于获取 其他8个内置对象

4、`request`

作用域范围：浏览器的一次请求响应

如果是转发操作，可以使用request 共享数据

5、`session`

作用域范围：一次会话，

6、`application`

作用域范围：整个应用程序，所用户共享

7、选择

能用小作用域，绝对不用大作用域

十三、EL表达式

1、简介

- Expression Language 表达式语言
- 是JSP中的一种语法（只能出现在JSP中）
- 可以出现在JSP的任意位置
- 语法`${xxxx }`**(在JSP中${}会被当做EL表达式，和ES6的模板字符串冲突)**

2、作用

2-1 显示和计算

```
<%@ page language="java"  pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>

    <title>My JSP 'el.jsp' starting page</title>
    <script type="text/javascript">
        //alert('${11+22}');
    </script>

  </head>
```

```
  <body>

        <h1>显示和计算</h1>
        <p>3 + 6 : ${3+6 }</p>
        <p>"22"+"11":${"22"+"11" }</p>
        <p>${1>3 }</p>
        <p>${"id=3" }</p>
        <p ${"id=3" } >aaa</p>

  </body>
</html>
```

2-2 获取作用域中的值

```
<%
 request.setAttribute("abc", "123");
session.setAttribute("abc", "222");
session.setAttribute("aaa", false);
application.setAttribute("def", new Date());
User u = new User();
u.setUsername("老王");
request.setAttribute("user", u);
%>

<h1>获取作用域中的属性</h1>
<p>abc:${abc }</p>
<p>aaa:${aaa }</p>
<p>def:${def }</p>


<h1>对于作用域中不存在的值，显示空字符串，不显示null</h1>
<p>user:${u }，<%=request.getAttribute("u") %></p>
<p>user:${user }</p>

<h1>获取对象属性</h1>
<p>username:${user.username }</p>
<font color="red">el表达式，获取对象属性，直接使用对象.属性名</font>
<font color="red">el表达式，访问对象属性，其实是在访问get方法</font>
<p>${user.abc }</p>

<h1>获取不同作用域中的同名属性</h1>
<font color="red">对于不同作用域中的同名属性，获取的是作用域范围小的</font>
<p>abc:${abc }</p>
<font color="red">可以使用 xxxScope.属性名 获取对应作用域中的属性</font>
<P>session:abc:${sessionScope.abc }</P>


<h1>绝对路径</h1>
<p>1: <%=request.getContextPath() %></p>

<p>2: ${pageContext.request.contextPath }</p>
```

```
${pageContext.request.contextPath
```

十四、JSTL

> java standard taglib java 标准标签库

1、使用方式

1. 导入标准标签库

```
<!--
    uri 标签库的uri地址
        定义在 jstl.jar/META-INF/c.tld中
    prifix
        c  表示标签的前缀  命名空间

 -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

2. 使用标签

```
<%@page import="java.util.ArrayList"%>
<%@page import="day03.User"%>
<%@page import="java.util.List"%>
<%@ page language="java"  pageEncoding="utf-8"%>
<!--
    uri 标签库的uri地址
        定义在 jstl.jar/META-INF/c.tld中
    prifix
        c  表示标签的前缀  命名空间

 -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE HTML >
<html>
  <head>

    <title>My JSP 'jstl.jsp' starting page</title>


  </head>

  <body>
    <!-- if标签
        test中的值是布尔类型或者布尔类型表达式
        如果test值为true，那么则显示if标签中间的内容，否则不显示
     -->
    <c:if test="true">
        <div>asdljadsjdlaskj</div>
    </c:if>
    <h3>和EL表达式一起使用</h3>
```

```
    <%
        request.setAttribute("age", 3);
    %>
    <c:if test="${age > 2 }">
        <p>年龄大于2</p>
    </c:if>

    <!-- 类似于switch-case 自带break效果
        从上到下，找第一个test为true的
        如果所有的test都为false，则匹配otherwise
     -->
    <c:choose>
        <c:when test="${age > 1 }">>1</c:when>
        <c:when test="${age > 2 }">>2</c:when>
        <c:otherwise>default</c:otherwise>
    </c:choose>

    <!-- for(int m = 1 ; m <= 7 ; m+=2){m} -->
    <c:forEach begin="1" end="7" step="2" var="m">
        <h4>${m }</h4>
    </c:forEach>


    <%
        List<User> list = new ArrayList<User>();
        for(int i = 0 ; i < 20;i++)
        {
            User u = new User();
            u.setUsername("name"+i);
            u.setPassword("pwd"+i);
            list.add(u);
        }

        request.setAttribute("list", list);
     %>

     <ul>
        <!-- for(User item:list){} -->
        <c:forEach items="${list }" var="item" varStatus="st">
            <li>第${st.index+1}项是：${item.username }---${item.password }</li>
        </c:forEach>
     </ul>
    <br><br><br><br><br><br><br><br><br><br><br>
  </body>
</html>
```

十五、HttpSession

1、Cookie

1-1、简介

- 是一种保存在客户端（浏览器）中的键值对（Map<String,String>）
- 浏览器每次发送请求，会在request中（请求头）携带cookie中的键值对
- 浏览器中能保存的cookie大小有限制，具体值和浏览器相关，大约4k
- **不是JSP内置对象**
- **是一种 客户端（浏览器）的 会话跟踪技术**

1-2、特点

1. cookie 是由 服务器 端 产生（创建）的
2. 通过响应对象（response）响应给客户端浏览器
3. 浏览器每次请求，会通过请求对象（request）传递给服务器
4. **保存在 浏览器中**

1-3、 Java中访问Cookie

```
package day04;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/cookie/*")
public class CookieServlet extends HttpServlet{

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String uri = request.getRequestURI();
        if(uri.endsWith("add"))
        {
            add(request,response);
        }

    }

    private void find(HttpServletRequest request, HttpServletResponse response)
    {
        Cookie[] cookies = request.getCookies();

    }

    private void update(HttpServletRequest request, HttpServletResponse response)
    {
        // 添加同名cookie
    }
```

```
    private void delete(HttpServletRequest request, HttpServletResponse response)
    {
        // 将cookie超时时间设置为0
    }

    private void add(HttpServletRequest request, HttpServletResponse response)
            throws ServletException,IOException{
        // 创建cookie，参数是 key 和value
        Cookie c = new Cookie("dd", "12345" + Math.random());

        // 当浏览器发送请求的时候，会将相同域和相同path（或者其子path） 的cookie 传递给服务器
        // 当 cookie中的domain+path 路径 是请求路径 或者 请求路径的父路径时

        // 域的值默认是 域名（ip/网址/localhost/127.0.0.1）
        // 设置cookie 的域  无法设置 跨域（不同域）的cookie
        // 必须同域
//      c.setDomain("www.baidu.com");
//      c.setDomain("localhost:8080");

        // 设置path值
        c.setPath("/");

        //为cookie 设置 超时时间（最大生存时间）
        //默认值是-1s 永不超时
        c.setMaxAge(5);

        response.addCookie(c);
    }

}
```

1-4、 JS中访问cookie

```
// 取值
var value = $.cookie("dd");
console.log(value);
// 赋值
$.cookie("dd","aaaaaa");
```

2、session

2-1、简介

- 是 JSP 内置对象之一，对应的Java类`HttpSession`
- 是保存在 服务器端的 会话跟踪技术
- 是键值对（Map<String,Object>）
- 理论上 大小 无限制

2-2、获取方式

```
HttpSession session = request.getSession();
// 参数为true ， 行为和无参一样
//HttpSession session = request.getSession(false);

System.out.println(session.hashCode());
```

2-3、原理

- 浏览器发送请求到服务器
- 服务器查询请求头中的cookie，看是否存在一个cookie值，叫做`jsessionId`
  - 不存在
    - 无参或者为true的情况下，在getSession的时候，会创建新的session
    - 参数为false的请情况下，返回null
  - 存在: 返回该sessionId 对应的session对象

浏览器从第一次访问服务器到访问结束（cookie 中的jsessionId消失），服务器中存在同一个session对象

2-4、生命周期

- 初始化: cookie 中不存在 `jsessionid`的时候
- 使用
- 销毁：当session 超过最大生存时间，或者调用了`invalidate`方法之后，session处于失效状态，此时，如果再次访问session，那么处于失效状态的session 被销毁

注意：

1. session 超过最大生存时间，或者调用了`invalidate`方法 只是让session失效
2. 浏览器关闭，只是cookie中jsessionId丢失
3. 使用`setMaxInactiveInterval`设置的是session 的 最大不活跃周期，即：从从最后一次访问sesion 开始计时，超过对应值（单位是秒）后，session处于无效状态

十六、文件上传

1、要求

1. `form`表单的`method`必须是 `post`
2. `form`表单的 `enctype` 必须是 `multipart/form-data`

2、原生Servlet实现

servlet实现文件上传的唯一方式

```
ServletInputStream is = request.getInputStream();
byte[] b = new byte[102400];
int i = is.read(b);
String str = new String(b,0,i);
System.out.println(str);
```

3、使用第三方jar包

对servlet 原生文件上传的封装

添加jar包

- commons-fileupload 文件上传的jar
- commons-io fileupload的依赖包

编写代码

```
try {
  // 创建磁盘类型的文件项 工厂对象
  FileItemFactory fileItemFactory = new DiskFileItemFactory();
  // 创建Servlet文件上传对象
  ServletFileUpload upload = new ServletFileUpload(fileItemFactory);

  request.setCharacterEncoding("utf-8");

  // 将request交给文件上传对象
  // 返回文件项 的 集合
  List<FileItem> fileItemList = upload.parseRequest(request);

  for (FileItem item : fileItemList) {
    // 如果item是表单域，(普通表单值)
    if(item.isFormField())
    {
      String fieldName = item.getFieldName();
      System.out.println("参数名:"+fieldName);
      String value = item.getString();
      System.out.println("参数值:"+value);
    }
    if(!item.isFormField())
    {
      String fieldName = item.getFieldName();
      System.out.println("参数名:"+fieldName);
      String name = item.getName();
      System.out.println("文件名"+name);
      item.write(new File("C://" + name));
    }

  }

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

4、项目中的封装

无论是自己封装的框架，或者其他的第三方框架（Struts，SpringMVC），都是对commons-fileupload 的封装

```
@RequestMapping("/upload")
public String upload( HttpServletRequest request,HttpServletResponse
response,List<CommonsMultipartFile> file) throws Exception
{
  String username = request.getParameter("username");
  String password = request.getParameter("pwd");
  System.out.println(username +","+password);
  String fileName = file.get(0).getOriginalFilename();
  file.get(0).transferTo(new File("c:/" + fileName));
  return null;
}
```

5、存在问题和解决方案

5-1 文件保存位置

```
// 获取application对象
//      ServletContext application = request.getServletContext();
//      ServletContext application = request.getSession().getServletContext();

// getServletContext 是继承自HttpServlet的方法
ServletContext application = getServletContext();

// 获取的是TOMCAT_HOME/webapps/项目名/参数对应的文件或者文件夹的 绝对路径
String realPath = application.getRealPath("upload");
String pathname = realPath + File.separator + item.getName();
try {
  item.write(new File(pathname));
} catch (Exception e) {
  e.printStackTrace();
}
```

5-2 文件名

```
// 当前时间毫秒数
//      long timestamp = System.currentTimeMillis();
// UUID
String uuid = UUID.randomUUID().toString();
String fileName = uuid +
            item.getName().substring(item.getName().lastIndexOf("."));
String pathname = realPath + File.separator + fileName;
```

5-3 数据库数据和文件的对应关系

一条数据对应一个文件

- 数据库中添加一个字段：保存文件地址（路径）
- 数据库中添加一个字段：保存文件原始名字（给用户看的，选加）

一条数据对应多个文件

- t_tableA : id xxxxx
- t_annex: id path oriName a_id

多张表中的一条数据对应多个文件

- t_tableA : id xxxx grou_id
- t_tableB : id xxxx group_id
- t_annex_group id name
- t_annex id path oriName group_id

十七、文件下载

1、使用物理路径下载

```
<a href="${pageContext.request.contextPath }/abc.zip">下载地址-abc.zip</a>
<a href="${pageContext.request.contextPath }/J2EE.pdf">下载地址-j2ee.pdf</a>
<img alt="" src="${pageContext.request.contextPath }/download/xxx.jpg">
```

2、使用Servlet下载

```
package day05;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Calendar;
import java.util.List;
import java.util.Map;
import java.util.UUID;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

import util.RequestUtil;

@WebServlet("/download")
public class DownloadServlet extends HttpServlet{
```

```
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        File f = new File("C:\\Users\\User\\Desktop\\一阶段\\JDK_API_1.6_zh_CN.CHM");

        FileInputStream is = new FileInputStream(f);

        byte[] b = new byte[is.available()];

        is.read(b);

        // text/html;charset=utf8
        // 响应类型是二进制流
        response.setContentType("application/octet-stream");

        String cd = "attachement;filename=java帮助手册.CHM";
        // 对中文进行转码操作
        cd = new String(cd.getBytes("utf-8"),"iso-8859-1");

        // 设置响应头
        response.setHeader("Content-Disposition", cd);

        ServletOutputStream os = response.getOutputStream();
        os.write(b);

        os.flush();
        is.close();
    }
}
```

3、特性

谁请求，响应给谁

```
<%@ page language="java"  pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>My JSP 'upload.jsp' starting page</title>
    <script type="text/javascript" src="${pageContext.request.contextPath }/download">
    </script>
  </head>

  <body>
    <button onclick="doAdd()">add</button>
  </body>
</html>
```

```
PrintWriter out = response.getWriter();
out.print("function doAdd(){alert(1)}");
```

十八、监听器

1、简介

`listener`

用于监听对象的创建和销毁

当对象被创建或者销毁的时候，自动地执行某段代码

- ServletRequestListener
- HttpSessionListener
- ServletContextListener

2、代码

```
public class AppListener implements ServletRequestListener,
    HttpSessionListener,ServletContextListener{

    @Override
    public void contextDestroyed(ServletContextEvent arg0) {
        System.out.println("服务器正常关闭.....................");
    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
        System.out.println("服务器启动.....................");

    }

    @Override
    public void sessionCreated(HttpSessionEvent arg0) {
        System.out.println("cookie中不存在jsessionId");
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent arg0) {
        System.out.println("session处于失效状态：invalidate/超时：默认30min");
    }

    @Override
    public void requestDestroyed(ServletRequestEvent arg0) {
        System.out.println("请求响应结束");
    }

    @Override
    public void requestInitialized(ServletRequestEvent arg0) {

        System.out.println("发出请求");
```

```
    }

}
```

3、基于web.xml配置

```
<listener>
    <listener-class>day05.AppListener</listener-class>
</listener>
```

4、基于注解

```
@WebListener
public class AppListener implements ServletRequestListener,
    HttpSessionListener,ServletContextListener{

}
```

十九、过滤器

1、简介

所谓的过滤器，指的是 对 请求 进行拦截和过滤，由过滤器代码决定是否继续向下执行对应的Servlet，并且在 执行Servlet 之前，之后 执行其他代码

2、实现方式

```
public class SomeFilter implements Filter{
    @Override
    public void destroy() {

    }
    @Override
    // interface HttpServletRequest extends ServletRequest
    public void doFilter(ServletRequest req, ServletResponse res,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("SomeFilter.doFilter()");
        HttpServletRequest request = (HttpServletRequest) req;
        request.getSession();
        // 继续执行对应的Servlet
        chain.doFilter(req, res);
    }
    @Override
    public void init(FilterConfig arg0) throws ServletException {

    }

}
```

3、基于web.xml配置

```
<filter>
  <filter-name>f</filter-name>
  <filter-class>day05.SomeFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>f</filter-name>
  <url-pattern>/s/*</url-pattern>
</filter-mapping>
```

4、基于注解

```
@WebFilter(urlPatterns="/s/*")
public class SomeFilter implements Filter{}
```

5、作用

- 字符集设置
- 记录系统日志
- 记录异常日志
- 权限控制

对多个Servlet中的相同代码片段可以提取到Filter中