# Javaweb

## Servlet

### 什么是servlet？

servlet是Java Web开发的基石，与平台无关的服务器组件，他是运行在Servlet容器/Web应用服务器(例：Tomcat),负责与客户端进行通信。

### Servlet的使用

Servlet在java中是一个接口，他有5个方法：

1. init():初始化
2. getServletConfig():获取servlet信息
3. service():处理客户端请求的方法**(核心)**
4. getServletInfo():返回servlet字符串信息
5. destory():释放资源

直接继承servlet的java类，即使写了service方法也无法被访问，因为被编译后的java类文件会在artificts下的WEB-INF包下，而web服务器是没有权利直接访问这个包的，要访问就需要在web.xml文件中添加标签：

**基于XML方式映射**

例：java类为test.java  
 <servlet>  
 <servlet-name>test</servlet-name>  
 <servlet-class>test</servlet-class>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>test</servlet-name>  
 <url-pattern>/test</url-pattern>  
 </servlet-mapping>

在service()方法中用println方法输出的字符串会显示在控制台上，但网页上不会显示，要想在网页上显示，则需要用response方法中的getWriter()方法下的write()方法进行输出，这样网页上才会显示。

有时网页响应为中文时可能会出现乱码问题，这时候就需要设置相应的类型：

servletResponce.setContontType(“text/html ; charset=UTF-8”);

这样响应为中文时就不会出现乱码。

Resquest()方法是获取接收到的信息，是一个请求方法。

**基于注解的方式映射**

不需要再在web.xml文件中写标签，直接在继承了servlet接口的java类上面添加注解：

**@WebServlet**

就可以实现用写web.xml实现的效果。

### Servlet的生命周期

* init()方法，初始化servlet对象，会在第一次访问网页时在内存中创建一个servlet对象，这个对象会被反复使用，即多次访问并不会创建多个对象（类似于字符串常量池，节省空间）。
* Service()方法，客户端方法，在每次访问网页时都会被调用，访问几次调用几次
* Destory()方法，在Tomcat启动时不会待用这个方法，作用是释放init方法所初始化的对象，在Tomcat服务器被关闭时自动调用，释放内存。
* 关于servlet的对象创建问题

Servlet的方法不是静态方法，所以调用他的方法一定是被创建了对象，这个对象是由Tomcat通过反射机制在访问网页时创建的，Tomcat调用了对象的构造方法创建对象之后，才能在访问网页时调用它的各种方法。

### Servlet的生命周期方法：无参构造、init、service()、destory

* 无参构造：只调用一次，用于创建对象。
* Init()方法：只调用一次，初始化对象。
* Service方法：调用N次，访问几次调用几次。
* Destory方法：调用一次，Tomcat服务器停止时调用，释放对象。

### Servlet的ServletConfig方法

ServletConfig是一个接口，用于描述servlet的基本信息。这里的获取参数的方法只适用基于xml创建的Servlet对象，在xml中的servlet中添加

<init-param>  
 <param-name></param-name>  
 <param-value></param-value>  
</init-param>

标签，这个标签本质上是一个map，name对应了键，value对应了值，只有这样设置，获取参数的方法才能获取到对象参数。

* getServletName()方法：返回Servlet(java类)的名字，全类名（带包名）
* getInitParameter(String key):获取init参数的值(web.xml)
* getInitParameteeNames():返回所有的initParameter的name值，一般用作遍历初始化参数(和getInitParameter(String key)方法结合使用)。
* getServletContext()方法：返回ServletContext对象，他是Servlet的上下文，整个Servlet的管理者。

ServletContext servletContext = servletConfig.getServletContext();

之后才能调ServletContext的方法。

ServletContext的两个常用方法：

1. getContextPath()方法：用于获取Tomcat部署的war项目下的路径
2. getServerInfo()方法：用11于获取Servlet的版本信息（例：Tomcat 9.0）

ServletConfig和ServletContext的区别：

ServletConfig作用于某一个实例，也就是说每个Servlet类都有对应的ServletConfig方法，他们之间是不同的对象。而ServletContext作用于整个web服务器，返回的都整体上的信息（抽象的），即多个Servlet实例可能对应一个ServletContext方法，而每一个ServletConfig方法都对应了不同的Servlet实例。

### Servlet的层次结构

Servlet ---> GenericServlet ---> HttpServlet

**HttpServlet:** HttpServlet是实例西城了GenericServlet，可以直接继承，常用的几个方法：

doGet():对应表单的GET操作，即读取

doPost():对应表单的POST操作，即保存

doPut():对应表单的PUT操作，即修改

doDelete():对应表单的DELETE操作，即删除

Java类可以继承HttpServlet类成为Servlet实例，实现他的方法就可以完成对应的操作。

**GenericServlet:**GenericServlet实现了Servlet接口的所有方法并进行了细分，让对应的操作进入对应的方法中。其中在service方法中可以将ServletRequest类型强转为HttpServletRequest类型，然后调用getMethod()方法获取操作，判断是否为POST/PUT/DELETE/GET...

## Jsp

JSP本质上就是一个Servlet，JSP主要负责和用户的交互，将最终的界面呈现给用户，相当于HTML+JS+CSS+JAVA的混合文件。

当服务器接收到一个后缀为JSP的请求时，将该请求交给JSP引擎去处理，每一个JSP页面第一次被访问的时候，JSP引擎会将它翻译为Servlet文件，再有Web容器调用Servlet完成响应。

从开发角度来看，JSP就是在HTML中嵌入java程序。

具体的嵌入方式有三种：

1. JSP脚本，执行Java逻辑代码  
    <% java代码 %>
2. JSP声明，定义Java方法  
    <%! 声明java方法 %>
3. JSP表达式，把Java对象直接输出到HTML页面中  
    <%=Java变量 %>

**例：**index.jsp

<%! public String test(){ return "方法声明"; } %>  
<% String str=test()+"-->变量声明"; %>  
<%=str+"-->调用输出" %>

## JSP的9个内置对象

* **request**:表示一次请求， HttpServletRequest类
* **response**:表示一次响应， HttpServletResponse类
* **pageContext**:页面上下文，获取页面信息， PageContext类
* **session**:表示一次绘画，保存用户信息， HttpSession类
* **application**:表示当前Web应用,全局对象,保存所有用户的共享信息， ServletContext类
* **config**:当前JSP对应的Servlet的ServletConfig对象，获取当前Servlet的信息
* **out**:向浏览器输出数据， JspWriter类
* **page**:当前JSP对应的Servlet对象， Servlet类
* **exception**:表示JSP页面发生的异常， Exception类

常用的对象：request、response、session、application、pageContext

**requset的常用方法：**

1. getParameter(String key)：String类型，获取客户端传来的数据
2. setAttribute(String key,Object value): void类型，通过键值对保存数据，生命周期比较短，随着对象的消失而消失
3. getAttribute(String key): Object类型，通过key取出value
4. getRequestDispatcher(String path): RequsetDispatcher类型，返回一个requestDispatcher对象，该对象的forward方法用于请求转发。
5. getParameterValues(String key): String[]数组类型，若是同一个属性有多个值，可以用String数组接收，然后输出所有的值，若是仍用第一个方法，则只会返回第一个值。
6. SetCharacterEncoding(String charset): void类型，当出现中文乱码时可以用这个方法指定请求转发的编码方式。  
    例：在<%%>中写java代码，用request的getParameter()方法通过“键”获取传过来的数据值，对其进行操作，之后可以用request的setAttribute()方法将数据以键值对的形式放入request中，最后用request的getRequestDispatcher(String url)方法，指定数据传入的url地址，再用getRequestDispatcher的forward(request，response)方法将请求和响应传入指定URL，而在定位的url地址的Servlet类中，则可以用request的getAttribute(String key)方法来获取传入的数据[注意这里不能用getParameter()方法，获取不到]。在同一个weib服务器中，类似这样的数据转发可以进行无数次。  
    getAttribute方法可以在一个web服务器中无限次的传数据，且返回值类型是Object（传过来的数据可以进行强转）。

**response的常用方法**

1. sendRedirect(String path):重定向，页面之间的跳转  
   转发getRequestDisoatcher和重定向sendRedirect的区别：

转发是将同一个请求传给下一个页面，也叫服务器跳转。重定向是创建一个新的请求给下一个页面，之前的请求结束生命周期，也叫客户端跳转。

直观上来看，转发将数据传入下一个页面之后地址栏不变，但重定向不会继承上一个页面的请求，地址栏会改变为path地址。

**Session的定义即常用方法** session是一个用户会话，由于服务器没办法识别每一次HTTP请求的出处（不知道来自于那个终端），他只会接受到一个请求信号，所以就存在一个问题：将用户的响应发送给其他人。必须有一种技术让服务器知道请求来自哪，这就是会话技术。

会话：就是客户端与服务器之间发生的一系列连续的请求和响应的过程，打开浏览器进行操作到关闭浏览器的过程。

会话状态：服务器和浏览器在会话过程中产生的状态信息，借助于会话状态，服务器能够把属于同一次会话的一系列请求和响应关联起来。

实现session的两种方式：

* session：session内置对象作用于服务器。
* cokie：cookie是作用于客户端（浏览器）。

**Session**

常用方法：

1. getId():String类型，session的getID()方法可以获取sessionID，这个ID在关闭浏览器之前都不会改变，但若是更换浏览器或者重新打开浏览器，这个ID就会改变。
2. setMaxInactiveInterval(int interval):void类型，设置session的失效时间，单位为秒。
3. getMaxInactiveInterval():int类型，获取当前session的失效时间。
4. invalidate():void类型，设置session立即失效。
5. setAttribute(String key,Object value):void类型，通过键值对存储数据。
6. getAttribute(String key):Object类型，通过键获取对应的数据。
7. removeAttribute(String key):void类型，通过键删除对应数据

**Cookie**

Cookie是服务器段=端在HTTP响应中附带传给浏览器的一个小的文本文件，一旦浏览器保存了某个Cookie，在之后的请求和响应过程中，会将此cookie来回传递，这样就可以通过cookie这个载体完成客户端和服务端的数据交互

* 创建Cookie

Cookie是一个类所以直接new

Cookie cookie = new Cookie(String key,String value);

Response.addCookie(cookie);

这个addCookie方法一定要加，不然不能将服务端创建的cooikie传回浏览器服务端中。

* 读取Cookie

可以用request的getCookies()方法获取到cookie，返回的是一个cookie数组

Cookie[] cookie = request.getCookies();

之后可以用cookie的getName()和getValue()方法，通过foreach循环遍历

Cookie的常用方法

1. setMaxAge(int age):void类型，设置cookie的有效时间，单位为秒
2. getMaxAge():int类型，获取cookie的有效时间（默认为-1，即网页关闭时失效）
3. getName():String类型，获取Cookie的name
4. getValue():String类型，获取Cookie的value

**Session和cookie的区别**

Session：保存在服务器中

保存的数据是Object

会随着会话的结束而销毁

保存重要信息

Cookie：保存在浏览器

保存的数据为String

可以长期保存在浏览器中，与会话无关

保存不重要信息

在存储用户信息方面

Session：setAttribute(key,value); 存

getAttribute(key); 取

生命周期：服务端：只要WEB应用（服务）关闭就销毁

用户端：只要浏览器关闭就销毁

退出登录：session.invalidate()；

Cookie：response.addCookie(new Cookie(key,value)); 存

foreach循环遍历结合getName和getValue取得值

生命周期：不随服务端的重启而销毁 客户端：默认在浏览器关闭时销毁，但可通过setMaxAge()方法设置有效期，有了有效期之后，cookie不随浏览器的关闭而销毁，而是过了有效期自动销毁

退出登录：setMaxAge(0);

## Http状态码

200：正常（一般看不到）

404：资源找不到  
400：请求类型不匹配  
500：java程序抛出异常

## JSP内置对象的作用域

4个内置对象：page、request、session、application

都有setAttribute()和getAttribute()方法

page作用域：对应的内置对象为pageContext。

request作用域：对应的内置对象为request。

session作用域：对应的内置对象为session。

application作用域：对应的内置对象为application

page < request < session < application

page只在当前页面有效

requset在一次请求内有效

session在一次会话内有效

application对应整个web项目

关于网页访问量：

Page肯定不行，request是每一次访问都会重置，因为每次刷新都是一次新的请求，session虽然刷新可以累计，但若是换一个浏览器或重启浏览器，产生了新的会话，那么也无法实现，只能用application，只要WEB服务器不关闭，就可以一直累计。

## EL表达式

只能在JSP文件中使用

Expression Language表达式语言，替代JSP页面中数据访问时的复杂代码，可以非常便捷地取出JSP内置对象中保存的数据，前提是一定要先setAttribute，EL就相当于再简化getAttribute。

语法：${变量名}

例：//存数据：以session为例

session.setAttribute(“name”,”mcl”);

//用getAttribute方式取出数据并输出：

out.print(session.getAttribute(“name”).toString());

EL表达式：${name};

1. 默认情况下：若是有同一个key值但使用的对象不同时，优先级为：

pageContext --> request --> session --> application

与域对象的作用域相反，且找到一个就停止，找不到就返回null。

1. 也可以指定作用域查找

pageContext: ${pageScope.name}

request: ${requestScope.name}

session: ${sessionScope.name}

application: ${application.name}

1. 还可以查找类

例：User类

<%  
 User user=new User("mcl",11);  
 application.setAttribute("user",user);  
%>  
${user}

或 ${user[“name”]} 或 ${user.name}

这个方式相当于默认使用get方法获取值，并不是直接访问属性值。

而且EL表达式能用的前提时数据被提前setAttribute之后才能使用

1. EL执行表达式

EL表达式中可以使用判断和运算符：==、&&、||、<=、>=、!=

empty：${ empty 变量 }:判断是否为空（变量null,字符串length为0,集合size为0）

## JSTL标签库

JSP Standard Tag Library ：JSP标准标签库，JSP为开发者提供的一系列的标签，使用这些标签可以完成一些逻辑上的处理，比如遍历集合，让代码更加简洁，不会出现JSP脚本穿插的情况。

实际开发中EL和JSTL结合起来使用，JSTL侧重于逻辑处理，EL负责展示数据。

JSTL的使用：

//首先导入jar包，并加入到JSP文件中

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

//代替for循环的java语句和setAttribute的穿插重复使用

//但要使用存入list中的数据，集合仍然要setAttribute放入。

//var是变量名，类似User user中的user。

requset.setAttribute(“user”,User);

<table>  
<tr>  
 <th>id</th>  
 <th>name</th>  
</tr>  
<c:forEach items="${list}" var="user">  
 <tr>  
 <td>${user.id}</td>  
 <td>${user.name}</td>  
 </tr>  
</c:forEach>

Foreach标签中还有begin，end，step和varStatus标签，分别表示为开始序号，结束序号，每次输出间隔和获取默认变量（默认变量中有些默认方法，可以根据需要使用）

JSTL的优点：

1. 提供统一的标签
2. 可以用于编写各种动态问题

常用标签：

set、out、remove、catch

**Set：向域对象中添加数据**

等同于：<% requset.setAttribute() %>

使用方法：

<c:set var=”name” value=”tom” scope=”requset”></c:set>

${requestScope.name}

不指定scope就是默认pageContext，这里的var代表setAttribute的key值,value就是值

set标签也可以修改对象内的数据（不能修改对象，set只能修改具体的数据）

<% User user = new User(1,”zz”);

Request.setAttribute(“user”,user)

%>

<c:set taget=”user” var=”name” value=”cc”></c:set>

${user.name}

Target代表指定的对象，var代表对象的属性，value表示属性值。

**Out：输出域对象中的数据**

使用方法

<c:out value=”${name}” default=”未定义”></c:out>//out的value值必须写成EL表达式。

在有数据或者没有defalut标签的情况下，这种写法与${name}这种EL标签没有区别，但在有default标签的情况下，若是name没有定义，则可以输出default的值，给用户反馈。

**Remove：删除域对象中的数据**

使用方法：

<c:remove var=”name” scope=”request”></c:remove>

删除方法比较紧剩，需要指定Scope类型，相同key值不同响应的误删。

**Catch：捕获异常、**

使用方法：

<c:catch var=”error”>

<%

Int a=10/0;

%>

</c:catch>

${error};

**条件标签**

If和Choose

例：

<c:set var=”num1” value =”1”></c:set>

<c:set var=”num2” value=”2”></c:set>

<c:if test=”${num1>num2}”>OK</c:if>

<c:if test=”${num1<num2}”>fail</c:if>

<c:choose>

<c:when test=”${num1>num2}”>ok</c:when>

<c:otherwise>fail</c:otherwise>

</c:choose>

这里的if语句没有else标签。

**Fmt数据格式化**

这里用到的就不是core核心库，而是fmt数据库。注意区分

对日期：

<% request.setAttribute(“date”,new Date()) %>

<fmt:formatDate value=”${date}” pattern=”yyyy-MM-dd HH:mm:ss”></fmt:formatDate>

对数字：

<fmt:formatDate value=”1254.125454” maxIntegerDigits=”2” maxFractionDigits=”3”>

</fmt:formatDate>

maxIntegerDigtis表示获取小数点前n位

MaxFractuinDigits表示获取小数点后n位

所以以上数据为：54.125

**函数标签库常用标签**

这里用的时function包，对应了String的各种用法

注意这里的与之前的使用方法不同，格式为${}

<% request.setAttribute(“info”,”Java,C”) %>

${fn : contains ( info , ” python ”)} //查询info中有没有”python”字符串

${fn : startsWith ( info ,” Java ”)} //查询info是否以Java开头

${fn : endsWith ( info, “ C ” )} //查询info是否以C为结尾

${fn : indexOf ( info, “ va ” )} //查询info中v字符的地址

${fn : replace ( info , “C”,”python”)} //将字符串中的C字符变成python并输出

${fn : substring ( info , 2 , 3)} //分割字符串，这里的2，3都是指下标

${fn : split ( info ,” , ” )[0]} ${fn : split( info , “ , ”)[1]} //将字符串以“，”为标志分割一个数组， //之后可以通过下标获取

## Filter过滤器

功能：

1. 拦截过滤传入的请求和传出的响应。
2. 修改或以某种方式处理正在客户端和服务端之间交换的数据流。

如何使用?

和servlet创建方法类似，实现Filter(Javax包下的)接口即可。

**例：**

public void doFilter(ServletRequest servletRequest, ServletResponse

servletResponse, FilterChain filterChain) throws IOException, ServletException {

servletRequest.setCharacterEncoding("UTF-8");

filterChain.doFilter(servletRequest, servletResponse);

}

doFilter是主体方法，没有default修饰，Filter是接口，他还有其他两个方法init和destory，但这两个方法有default修饰，所以可以不用写。

**XML设置**

<filter>  
 <filter-name>charactor</filter-name>  
 <filter-class>Parctice.Filter.CharacterFilter</filter-class>  
</filter>  
<filter-mapping>  
 <filter-name>charactor</filter-name>  
 <url-pattern>/Login</url-pattern>  
</filter-mapping>

这样就对Login这个bean做了拦截器，若是想要拦截多个页面，则再添加一个<url-pattern>即可。

FilterChai n的作用：

一个服务器可以有多个过滤器，而filterChain的用处就是将所有的过滤器和服务器连接起来，每次过滤后将过滤后的请求响应发送给下一个过滤器或服务器，之后就可以将过滤后的内容显示出来。

**Filter的生命周期**

当tomcat启动时，通过反射机制Filter的无参构造创建实例化对象，同时调用init方法初始化，doFilter在每次有请求响应时都会被调用，当tomcat服务关闭的时候，调用destory来销毁Filter对象。（前提是在xml中配置过了，不配置不执行）

无参构造：只调用一次，当Tomcat启动时调用（Filter要配置）

Init方法：只调用一次，当filter的实例化对象被创建时调用。

doFilter方法：调用多次，访问Filter的业务逻辑都写在Filter中

Destory方法：只调用一次，在tomcat关闭时调用。

Filter过滤器的顺序与xml里的位置有关系。

Filter和servlet一样都可以用注解的方式添加。

@WebFilter(“/login”)

用注解的方式简化web.xml中的配置，但对于一个网页的多个过滤器，注解的方式无法决定过滤器的先后顺序，注解的先后顺序是随机的，而且若是要初始化参数，注解也无法实现。

Filter在实际开发中的应用

1. 统一中文乱码
2. 屏蔽敏感词

（使用replaceAll(获取的数据，替换的数据)将获取到数据中的敏感词转换）

首先，使用注解方式使用过滤器，过滤器对象应该是接收信息的servlet而不是发送信息的servlet，其次，虽然Filter可以用getParameter接收数据，但是在转发数据时不能使用getParameter获取，不然获取的就仍然是未处理过的数据。比较好的方式是将数据使用setAttribute存放数据，再用getAttribute获取数据。

1. 控制资源的访问权限

比如下载资源时，判断用户是否登录，若登录则继续，未登录则重定向到登陆界面。逻辑上是将之前写的登陆页面中的用户信息存放在session中，然后再这里获取session信息，若是登陆成功则session里有数据，否则就没有数据。

但在这里的servletRequest和servletResponse都是request和response的父类，没有getSession方法，也没有response的重定向方法，所以都需要向下转型。

ServletRequest request = （HttpSerevletRequset）ServletRequest；

Response类似。

## 文件上传下载

1. Jsp
   1. 将input的type设置为file
   2. Form表单的method设置为post，若使用get，则获取到的是文件的名字而不是文件本身。
   3. 但若只是用post，传过去的数据仍然只是文件名字，只有将form表单的enctype设置为”multipart/from-data”,以二进制形式传输信息。

**Servlet**

1. Servlet端接收数据，若使用getParameter方法获取，则获取的只是字符串，返回的结果自然是null，因为文件上传下载本质上是对流的操作，所以应该使用流的操作类。

例： InputStream is = req.inputStream(); //获取流。

对于文件，即使是Text文件，只是这样也一样读不出来，因为inputStream是字节输入流，对于任何文件都是以二进制读取，所以仍然需要一些其他操作。

Reader reader = new InputStreamReader(is);//这里是将字节流转化为字符流读取。

BufferReader br = new BufferReader(reader);//这里是将读取到的字节放入缓冲区。

//若是想要用输出流输出到到一个本地文件中，则需要用OutputStream这样的输出流，输出到文件中，但注意这里的OutputStream读取文件是需要获取到文件的绝对路径,而这个绝对路径通常是tomcat工程的绝对路径，因此仍然需要以下操作。

//通过context的读取路径方法获取绝对路径，file在本地工程文件中

String path = req.getServletContext().getRealPath(“file/copy.txt”);

String str=””;

OutputStream os = new FileOutputStream(path);//通过绝对路径定位到文件

Writer writer = new OutputStreamWriter(os);//将字节输出流转化为字符输出流

BufferWriter bw = new OutputStreamWriter(writer);//获取输出流准备将数据写入文件。

While((str=os.readLine())!=null){

Sout(str);

Bw.write(str);//将数据写入到path所指路径文件中

}

....//最后关闭所有流

**但真正开发并不使用以上方法，而是使用Apache提供的的类：fileupload**

Fileupload将一个文件转化为一个类，对文件操作就相当于对类进行操作，当然即使不是文件，服务器端的上传类型是text，fileupload也能获取到这个请求的基本信息。

使用这个方法需要导入两个包：commons-fileupload-1.3.1、commons-io-2.5

例：

DiskFileItemFactory dfif = new DisFileItemFactory();//创建工厂

ServletFileUpload sfu = new ServletFileUpload(dfif);//上传下载文件类给工厂

//通过parseRequset方法，将上传的文件类转变成一个list集合（几个文件几个Item）。

List<FileItem> list = sfu.parseRequest(req);

for(FileItem fi : list){

Sout(fi.getName());//获取文件名

Sout(fi.getFileName());//获取请求端上传的文件name

}

若是获取到的list集合中有不是文件类型的，可以用fileItem的isFormFiled()方法判断，true为文本，false为文件。

若是要对这个文件进行写操作，则仍然需要使用之前的方法

InputStream is = fileItem.getInputStream();

String path = req.getServletContext().getRealPath(“file/”+fileItem.getName());

OutputStream os = new FileOutputStream(path);//通过绝对路径定位到文件

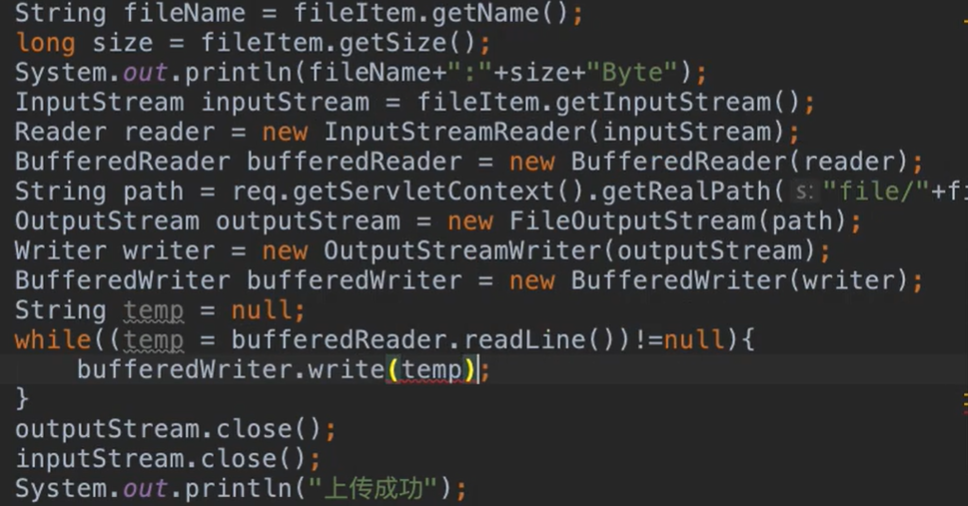
Int temp=0;

If((temp=inputStream.read())!=-1){

os.write(temp);//这是字节流，字符流与上面差不多

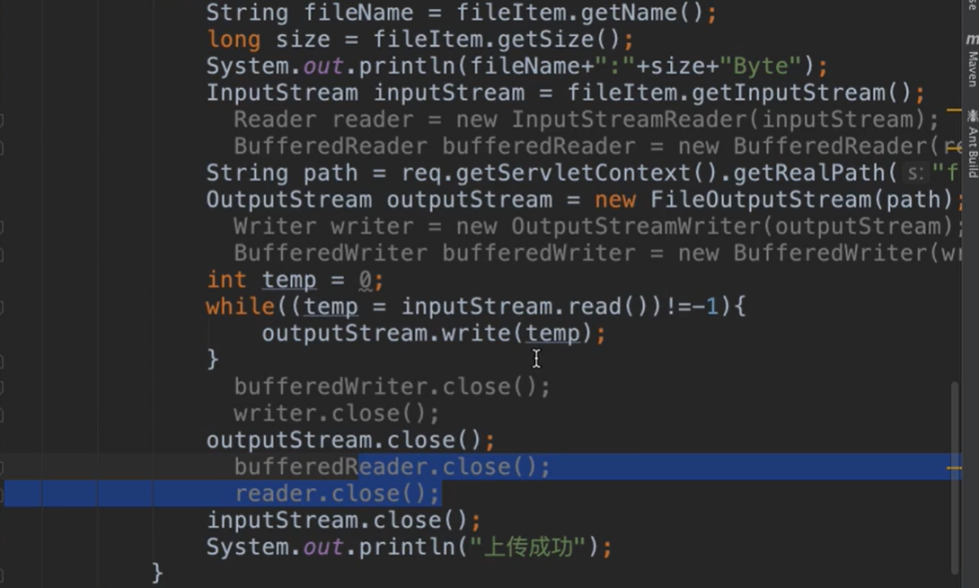
}

...//关闭数据流



对于图片，不能用字符流读，不然会打不开，直接用字节流就可以了。

即：



**下载**

下载的原理：将文件读取到服务器，之后若点击下载，则将文件用输出流输出到客户端。

对于下载的文件，一般jsp用超链接，所以servlet要用doGet方法，而doGet方法中，首先应该设置servlet的响应方式为下载，所以第一步应该设置相应方式，之后则应该设置下载时的文件名，最后通过输入流将数据下载下来。

resp.setContextType(“application/x-msdownload”);//设置servlet的类型

Resp.setHeader(“Context-Disposition”,”attachment;filename=”+filename);//设置下载时的文件名

OutputStream os = new OutputStream();

String path=req.getServletContext().getRealPath(“file/”+filename);

InputStream is = new FileInputStream(path);

Int temp = 0;

while((temp=is.read)!=-1){

os.write(temp);

}

//关闭流