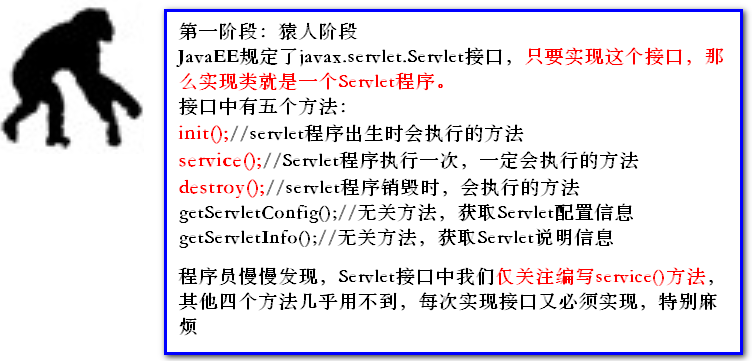
# 1、Servlet知识总结

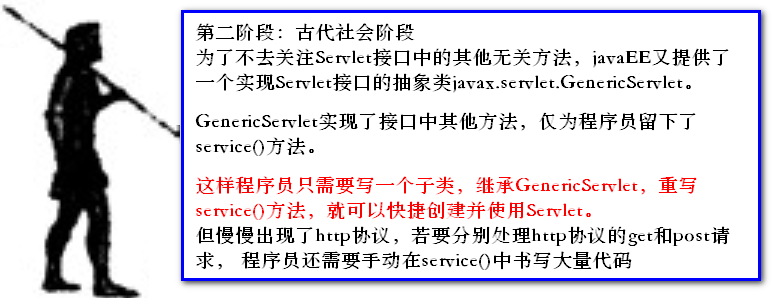
## 1.1、Servlet的发展史及生命周期

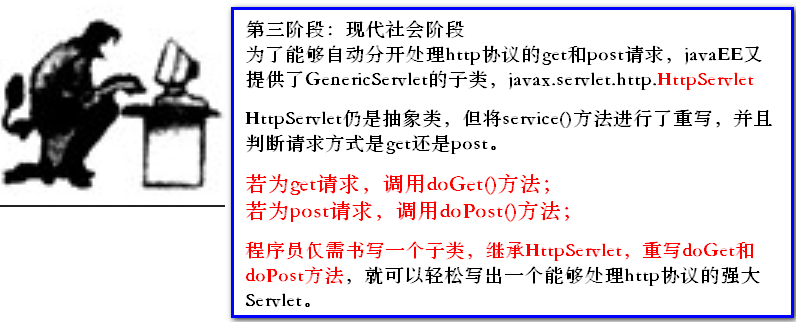
Servlet也是由复杂难用、功能单一，逐步发展到目前的操作简单，功能强大的。

Servlet的发展史，完全就是一部人类由猿人的进化史。

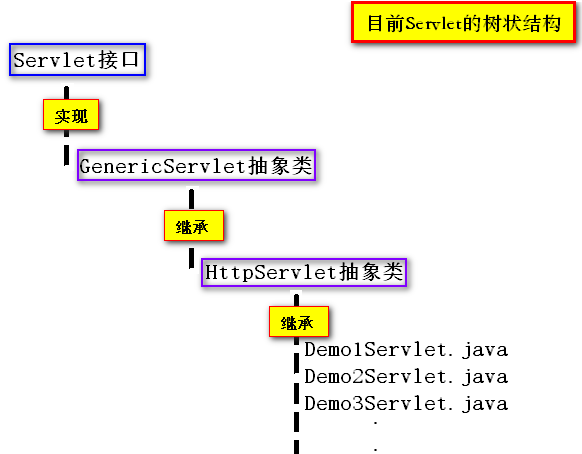
* 发展史







* 结构继承树



单例模式(只允许创建一个对象)、多例

Servlet在内存中是**单例**----单实例对象

一个Servlet类 在内存中最多有一个对象

Servlet的生命周期：

* Servlet创建：Servlet在第一次被访问时。 Servlet创建时，服务器会调用该Servlet的init()方法。
* Servlet销毁：Servlet在服务器正常关闭时。 Servlet销毁时，服务器会调用该Servlet的destroy()方法。
* Servlet工作：Servlet被调用一次，就会执行一次service()方法，service()根据请求方式，调用doGet()和doPost()方法。

## 1.2、Servlet的相关的配置:

### 1.2.1、【启动时创建Servlet】

需要将Servlet提前创建出来。

一个项目有多少功能，将来就有多少Servlet。

作用：就是可以在服务器启动时，创建Servlet对象

Servlet默认是在第一次访问的时候创建的.现在让Servlet在服务器启动的时候创建好.进行对Servlet的配置:

在web.xml中在<servlet></servlet>标签中配置:

\* <load-on-startup>2</load-on-startup> --- 传入正整数,整数越小,被创建的优先级就越高.

建议：使用时写大于等于2的正整数

生命周期：

创建：**默认是Servlet第一次访问。如果加入<load-on-startup>标签，提前到服务器启动时**

### 1.2.2、【url-pattern的配置】

某个Servlet的访问路径。

url-pattern可以设置无限多个。

url-pattern配置方式共有三种:

1. **完全路径匹配** ：地址栏请求路径，和web.xml设置的访问路径完全一致，才能访问

格式：以 / 开始 ，/代表的是当前的项目下

例如: /demo1 , /aaa/demo2 , /aaa/bbb/demo3

1. **目录匹配** ：地址栏请求路径的**目录名**，和web.xml设置的访问路径**目录名**完全一致，才能访问

**格式：**以 / 开始 需要以 \* 结束.

例如: **/\*** （所有） ,**/aaa/\*** （aaa目录下的所有） ,**/aaa/bbb/\***

1. **扩展名匹配** ：地址栏请求路径的**扩展名**，和web.xml设置的访问路径**扩展名**完全一致，才能访问

格式：不能以 / 开始 以 \* 开始的.

例如: \*.do , \*.action 、\*.jsp 、\*.jpg 后面框架中会用到

\*\*\*\*\* 错误的写法 ： /\*.do

4.缺省路径/缺省匹配 /

通常情况访问 html页面时，首先从当前web项目的web.xml文件寻找匹配路径，如果如果没有找到，再从tomcat默认的web.xml匹配，将使用缺省servlet

tomcat获得匹配路径时，优先级顺序：完全匹配 > 目录匹配 > 扩展名匹配 > 缺省匹配

if(完全匹配){

//完全匹配

}else if(目录匹配){

//目录匹配

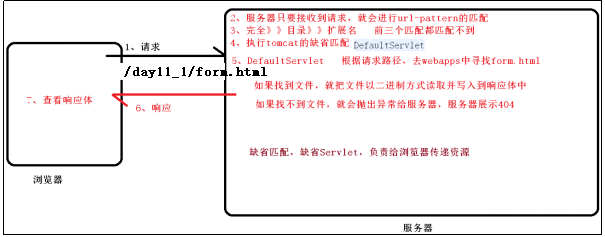
}else if(扩展名匹配){

//扩展名匹配

}else{

//缺省匹配

}



随堂练习：

有如下的配置:

<servlet-mapping>

<servlet-name>ServletDemo</servlet-name>

<url-pattern>/hehe/d4</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>ServletDemo5</servlet-name>

<url-pattern>/hehe/\*</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>ServletDemo6</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

如果访问地址:

[http://localhost:8080/day](http://localhost:8080/day13/ServletDemo4)11\_1/hehe/d4

[http://localhost:8080/day11\_1/hehe/nihao](http://localhost:8080/day34/hehe/nihao)

[http://localhost:8080/day11\_1/nihao.do](http://localhost:8080/day34/nihao.do)

[http://localhost:8080/day11\_1/demo1.html](http://localhost:8080/day34/demo1.html)

以上四个路径，分别会执行哪些Servlet？？？

## 1.3、ServletConfig:获得Servlet的配置信息(了解)

每个Servlet都有一个<Servlet>标签对应，<Servlet>标签的内容，可以理解为某个Servlet独一无二的配置信息。

ServletConfig对象，将<servlet>标签中所有的信息进行了封装。

一个Servlet就有一个ServletConfig对象。

ServletConfig对象就是某个Servlet独一无二的身份证

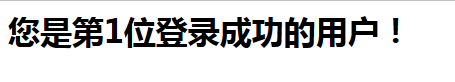
获取ServletConfig对象：doGet()/doPost()方法中，

this.getServletConfig();

|  |  |  |
| --- | --- | --- |
| 方法名 | 返回值类型 | 描述 |
| getInitParameter(name) | String | 获得Servlet的初始化参数的.  用来替代某个Servlet中的常量的 |
| getServletContext() | ServletContext | 获得ServletContext对象的引用  项目启动时，将web.xml中所有的配置信息进行封装。  ServletContext相当于某个项目独一无二的身份证 |
| *getServletName()* | *String* | *获得Servlet在web.xml中配置的name的值。*  *及<servlet-name>标签配置内容。* |

# 2、案例1：记录网站的登录成功人数

## 2.1、需求说明及原型导入



登录成功后,5秒后跳转到某个页面,在页面中显示您是第x位登录成功的用户.

分析：

1. 如何去存取 x 位 ServletContext
2. 发送给浏览器 response

## 2.2、知识讲解-ServletContext

ServletContext是某个项目独一无二的身份证对象。

服务器启动的时候,为每个WEB应用创建一个单独的ServletContext对象,将web.xml中的配置信息进行封装。

生命周期：

创建：服务器正常启动时

销毁：服务器正常关闭时

作用：

* + 获取项目配置信息；
  + 项目内进行信息共享；

Servlet中的获取方式（必须写在某个Servlet内部）：**this.getServletContext();**

只要是在同一个项目中，任意Servlet获取到的都是同一个ServletContext对象。

ServletContext对象---域对象

域对象：在域作用范围内进行数据共享。相当于一个map

|  |  |
| --- | --- |
| 方法名 | 描述 |
| **setAttribute(String name,Object object)** | 向ServletContext中存数据  相当于map.put(key,value); |
| **getAttribute(String name)** | 从ServletContext中取数据  相当于map.get(key) |
| **removeAttribute(name)** | 从ServletContext中移除数据  相当于map.remove(key) |

作用域---作用范围：整个项目的所有Servlet

servlet的初始化参数, 通过ServletConfig对象获得

整个项目的初始化参数（全局初始化参数）, 通过ServletContext对象获得

//获取项目名

## 2.3、知识讲解-response

在Servlet API中，定义了一个HttpServletResponse接口，它继承自ServletResponse接口，专门用来封装HTTP响应消息。

ServletResponse接口，普通响应接口。不能处理http协议

HttpServletResponse接口，专门处理http协议的响应接口。

实现类不同的服务器，有不同的实现类，不需要程序员关注。

Response对象封装了响应行、头、体三部分信息，我们可以通过调用response对象的方法，来轻松控制服务器向浏览器写响应信息

### 2.3.1、response对象操作响应行

响应状态码

200 响应成功

302 重定向

404 路径找不到

500 服务器内部异常

* **setStatus(int status);**

该方法用于设置HTTP响应消息的状态码，并生成响应状态行。由于响应状态行中的状态描述信息直接与状态码相关，而HTTP版本由服务器确定，因此，只要通过setStatus(int status)方法设置了状态码，即可实现状态行的发送。

若未设置状态码，且服务器运行正常，Web服务器会默认产生一个状态码为200的状态行。

* **sendError(int sc,String errorMsg);**

该方法用于发送表示错误信息的状态码。

建议该方法仅发送4xx和5xx状态码。

### 2.3.2、response对象操作响应头

当Servlet向客户端发送响应消息时，由于HTTP协议的响应头字段有很多种，为此，在HttpServletResponse接口中，定义了一系列设置HTTP响应头字段的方法，如表4-1所示。

1. 设置响应消息头字段的方法

|  |  |
| --- | --- |
| **方法声明** | **功能描述** |
| void addHeader(String name, String value) | 这两个方法都是用来设置HTTP协议的响应头字段，其中，参数name用于指定响应头字段的名称，参数value用于指定响应头字段的值。不同的是，addHeader()方法可以增加同名的响应头字段，而setHeader()方法则会覆盖同名的头字段 |
| **void setHeader(String name, String value)** |
| **void** **setCharacterEncoding(String charset)** | 该方法用于设置输出内容使用的字符编码，对HTTP 协议来说，就是设置Content-Type头字段中的字符集编码部分。  （后面响应体中讲解） |
| **void setContentType(String type)** | 该方法用于设置Servlet输出内容的MIME类型，对于HTTP协议来说，就是设置Content-Type响应头字段的值。例如，如果发送到客户端的内容是jpeg格式的图像数据，就需要将响应头字段的类型设置为“image/jpeg”。需要注意的是，如果响应的内容为文本，setContentType()方法的还可以设置字符编码，如：text/html;charset=UTF-8  （后面响应体中讲解） |
| *void addIntHeader(String name,*  *int value)* | *这两个方法专门用于设置包含整数值的响应头。避免了使用addHeader()与setHeader()方法时，需要将int类型的设置值转换为String类型的麻烦* |
| *void setIntHeader(String name,*  *int value)* |
| void setContentLength(int len) | 该方法用于设置响应消息的实体内容的大小，单位为字节。对于HTTP协议来说，这个方法就是设置Content-Length响应头字段的值  **无需手动设置，由服务器自行设置** |

### 2.3.3、response对象操作响应体

响应体信息无限，所以一般用来传递大量数据。

* **getOutputStream()**

向响应体输出二进制字节流信息。【文件下载】

* **getWriter()**

向响应体输出字符流信息。【字符串及前端代码输出】

若输出前端代码，则会被浏览器解析执行

两个方法我们都使用其write()方法进行数据输出。

* 乱码解决：
  + **setContentType("text/html;charset=UTF-8");**

设置响应体的编码字符集为utf-8

设置浏览器的解码字符集为utf-8

告知浏览器响应体里是html代码，需要进行解析

注意事项：

1. 乱码解决的设置代码，必须放在获取字节流/字符流之前。
2. 一山难容二虎。**一个响应体不可以同时被两种流操作。**

以下代码因为一次仅操作一种流，没问题

If(){

//字节流

}else{

//字符流

}

3、两种流，因为是被服务器管理的，所以**无需程序员关闭**

## 2.4、知识讲解-资源跳转-重定向

javaEE资源跳转方式有两种：

1. 重定向---请求重定向
2. 请求转发

今天我们来学习重定向方式进行资源跳转

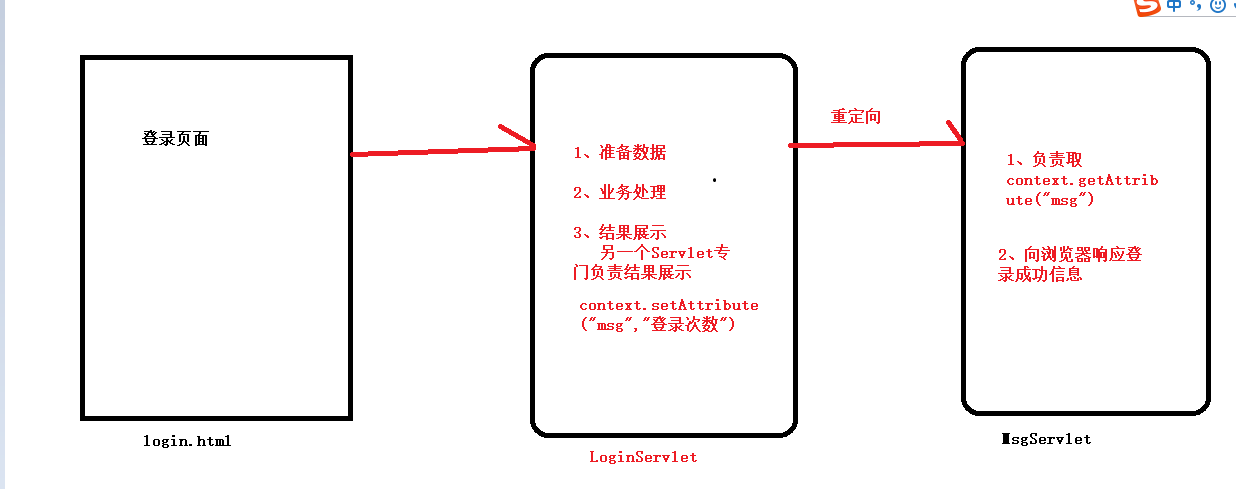


重定向严重依赖浏览器，路径是客户端路径。

|  |
| --- |
| **response.setStatus(302);**  **response.setHeader("Location","路径");** |

* 浏览器立即重定向：**response.sendRedirect("客户端绝对路径");**
* 浏览器延迟X秒重定向：response.setHeader("refresh","X秒;url=客户端绝对路径");

## 2.5、流程分析



## 2.6、案例代码实现

# 3、案例3：文件下载

## 3.1、需求说明

分别使用超链接和Servlet程序，进行网站资源的下载。

## 3.2、需求分析

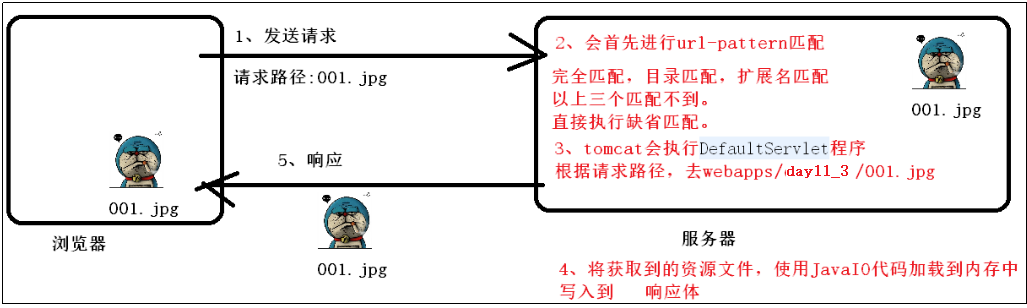
我们在上网时看到的所有数据，本质都是下载到本地进行查看。

超链接下载，

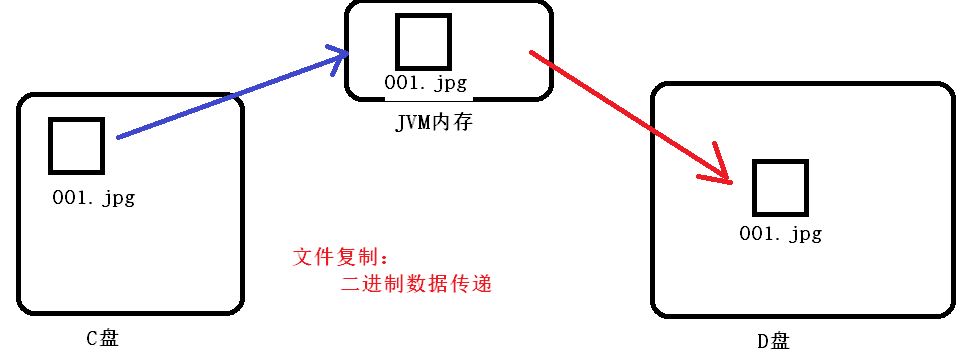
下载后，能打开，就打开查看

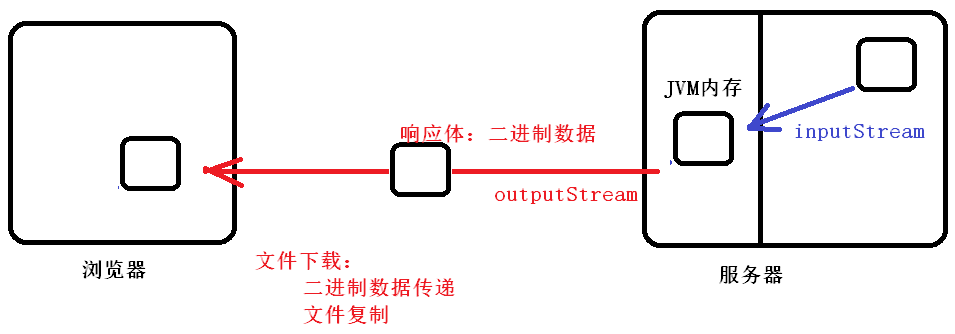
打开不了的格式，提供下载窗口

超链接下载方式，其实就是依赖了缺省Servlet进行下载。



文件下载本质：





## 3.3、案例代码实现

### 3.3.1、强制下载响应头设置

// 必须写在getOutputStream()之前 正文-配置 filename是下载名

//下载名会直接显示在下载的窗口中

response.setHeader("Content-Disposition", "attachment;filename="+filename);

### 3.3.2 文件下载中文文件名乱码问题

|  |
| --- |
| IE URL编码方式解决  URLEncoder.encode(fileName,"UTF-8")  firefox：  利用String的构造方法  new String(fileName.getBytes(),"ISO-8859-1") |

# 4、案例4：点击切换验证码

## 4.1、需求说明

在访问登录页面时，需要生产验证码。从而防止用户使用程序恶意操作。



验证码本质：

图片

验证码作用：

用户输入验证码浪费的时间，就是服务器喘口气的时间。

验证码的难易：

取决于 网站的流量

## 4.2、案例原型准备

生成图片验证码工具：（无需手写，直接使用）

public class VerifyCodeServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

//生成图片

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//1 高和宽

int height = 30;

int width = 60;

String data = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz";

Random random = new Random();

//2 创建一个图片

BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

//3 获得画板

Graphics g = image.getGraphics();

//4 填充一个矩形

// \* 设置颜色

g.setColor(Color.BLACK);

g.fillRect(0, 0, width, height);

g.setColor(Color.WHITE);

g.fillRect(1, 1, width-2, height-2);

// \* 设置字体

g.setFont(new Font("宋体", Font.BOLD|Font.ITALIC, 25));

//5 写随机字

for(int i = 0 ; i < 4 ; i ++){

// 设置颜色--随机数

g.setColor(new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)));

// 获得随机字

int index = random.nextInt(data.length());

String str = data.substring(index, index + 1);

// 写入

g.drawString(str, width/6 \* (i + 1), 20);

}

//6 干扰线

for(int i = 0 ; i < 3 ; i ++){

// 设置颜色--随机数

g.setColor(new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255)));

// 随机绘制先

g.drawLine(random.nextInt(width), random.nextInt(height), random.nextInt(width),random.nextInt(height));

// 随机点

g.drawOval(random.nextInt(width), random.nextInt(height), 2, 2);

}

//end 将图片响应给浏览器

ImageIO.write(image, "jpg", response.getOutputStream());

}

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

## 4.3、案例代码实现

当浏览器解析到<img>标签时，

会自动根据src发请求。 Get请求。

最终被缺省Servlet匹配到，把图片写入响应体，发回浏览器

浏览器根据响应体图片，替代<img/>标签