# Servlet概念

## 1.1概念

1)Servlet是一个普通的java类,继承了HttpServlet类

2)其实实现了Servlet接口的java类,才是一个Servlet类

3)servlet程序需要交给tomcat服务器运行

# Servlet执行过程

1. 浏览器:http://localhost:8080/web/hello
2. 截取URL字符串,得到需要访问的资源名称/hello
3. 在web网站下的web.xml信息中搜索是否存在/hello匹配的url-pattern
4. 匹配到对应的url-pattern之后,取出当前映射信息servlet-name的名称,然后

在web.xml中搜索是否存在一个相同名称的servlet-name的配置信息

1. 找到对应的servlet配置信息后,取出当前配置信息的servlet-class的内容

字符串: com.company.HelloServlet

1. 创建HelloServlet对象,然后调用调用HelloServlet类中的方法
2. Servlet就把内容返回给浏览器用户

# 映射路径

|  |
| --- |
| <servlet>  <servlet-name>HelloServlet</servlet-name>  <servlet-class>com.company.HelloServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>HelloServlet</servlet-name>  <url-pattern>/hello</url-pattern>  </servlet-mapping> |

**精确匹配** urlpattern 浏览器访问

/hello <http://localhost:8080/web/hello>

/itcast/hello <http://localhost:8080/web/itcast/hello>

模糊匹配 /\* <http://localhost:8080/web/任意路径>

/itcast/\* <http://localhost:8080/web/itcast/任意路径>

\*.后缀名 <http://localhost:8080/web/任意路径.后缀名>

注意:

1. url-pattern要么以斜杠开头,要么以\*开头
2. 非法的url-pattern /web /\*.html
3. 当多个url-pattern同时被匹配的时候

精确匹配优先,以后缀名结尾的url-pattern的优先级最低

# 4.缺省路径

在tomcat服务器内部配置了一个缺省路径<url-pattern>/</url-pattern>,这个缺省路径在tomcat服务器内部被一个缺省Servlet(DefaultServlet)匹配

作用:处理所有网站中过的静态文件或静态网页

浏览器:http://localhost:8080/web/hello.html

一个路径在tomcat服务器的执行过程:

1. 搜索web下面的web.html文件信息,所有是否存在匹配的url-pattern
2. 匹配到对应的url-pattern,执行执行对应的Servlet程序
3. 如果没有匹配的url-pattern,就把这个请求交给tomcat服务器的DefaultServlet处理
4. DefaultServlet会在web的根目录下搜索是否存在一个名叫hello.html文件
5. 如果找到此文件,那么DefaultServlet读取该文件内容返回给用户
6. 如果找不到此文件,那么就给用户返回一个404状态码和404错误页面

结论:先找动态网页,再找静态网页

# 5.Servlet生命周期

## 5.1概念

研究Servlet的声明周期,就是研究servlet这个在程序在什么时候创建对象,什么时候调用什么方法,在什么时候销毁对象

## 5.2Servlet的四个重要的生命周期方方法

**构造方法**:创建servlet对象时调用,只调用1次,证明servlet是单例

**init方法**:创建完对象之后调用,只调用1次,该方法用户初始化对象

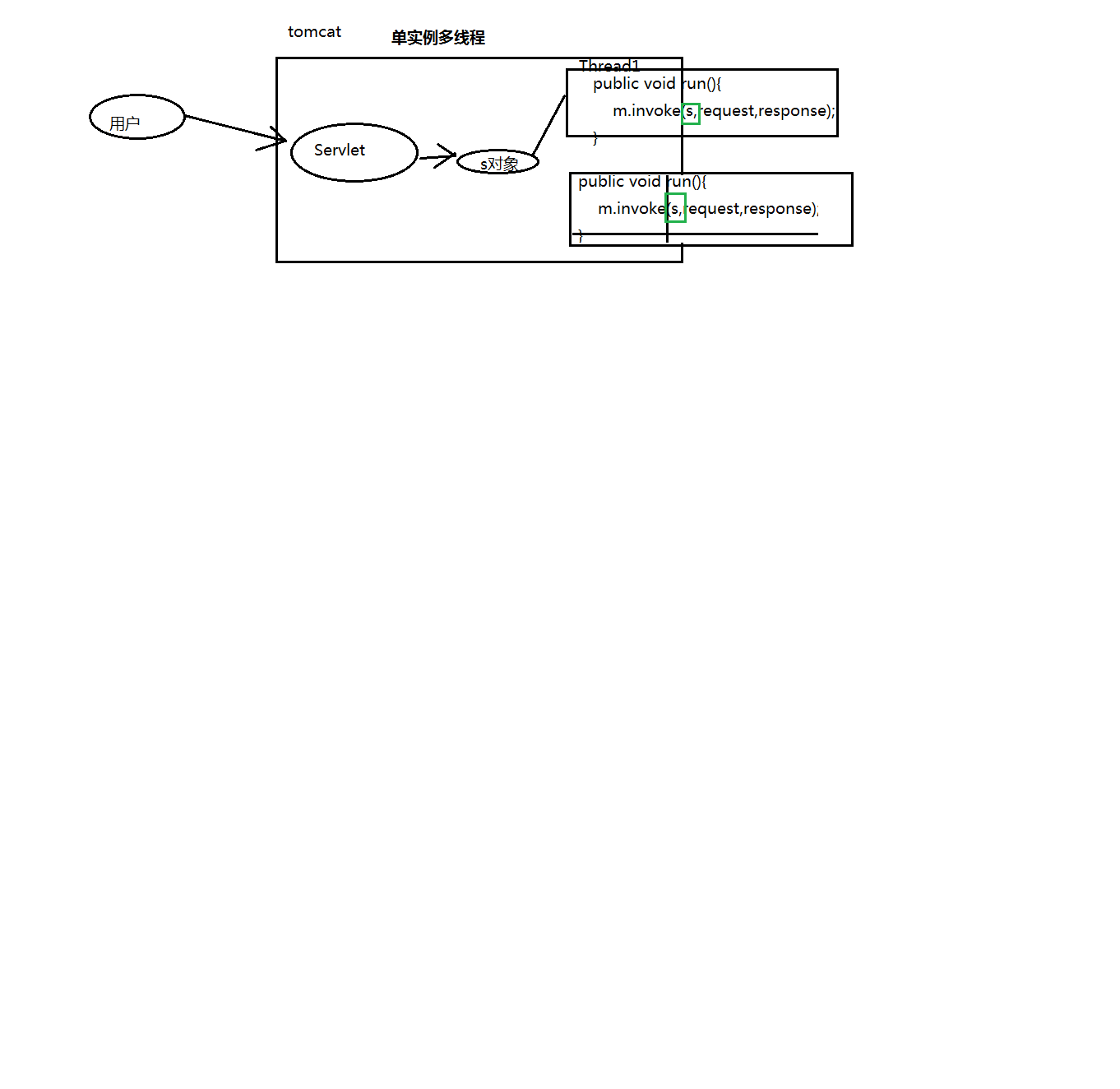
**service方法**:在每次请求时调用,调用n次,该方法是入门方法

**destroy方法**:在servlet对象销毁之后调用,只调用1次,重新部署网站或者停止服务器,servlet对象就被销毁了

# 6.Servlet线程并发问题

6.1引入

servlet在tomcat服务器中,是单实例多线程的



建议:

1. 尽量不要使用成员变量,或者静态成员变量
2. 必须要使用成员变量,要么给使用了成员变量的代码块加同步锁,加锁的代码块范围尽量缩小,因为有可能影响程序并发效率

# 7.Servlet的自动加载机制

## 7.1引入

默认情况下Servlet对象是在第一次访问的时候创建，如果这个servlet的构造方法或者init方法执行比较多的逻辑,那么第一次访问servlet的用户体验就不好

那么可以改变创建servlet的机制,可以让

## 7.2自动加载的配置

即:tomcat启动时,自动创建servlet对象,而不是等到用户第一次访问,tomcat才创建

|  |
| --- |
| <servlet>  <servlet-name>HelloServlet</servlet-name>  <servlet-class>com.company.HelloServlet</servlet-class>  <!--这个配置可以让tomcat服务器自动创建servlet对象-->  <load-on-startup>1</load-on-startup>  </servlet> |

注意:load-on-startup标签内容为数值,数值越小,越先被创建

# 8.Servlet继承结构图

构造方法

Servlet接口

init(ServletConfig config)

service(request,response)

destroy()

通用的不基于任何协议的

init(ServletConfig)

init()

GenericServlet类

Servlet实现

HttpServlet类

引入不同doXXX()

自己的Servlet

# 9.Servlet的init方法

init(ServletConfig)这个有参对的init方法的是Servlet的一个生命周期方法

init():这个无参的init方法作用是为了提供给开发者方便去覆盖,这个方法才是给

开发者做初始化逻辑的方法

**在init(ServletConfig)方法里,调用了init()方法,init()方法里是没有内容的,tomcat调用的是init(ServletConfig)方法,所以我们应该覆盖init()方法,而不是覆盖init(ServletConfig)方法**

# 10.Servlet涉及的类

HttpservletRequest:请求对象

HttpServletResponse:响应对象

//以上连个类的文档在http.docx中有讲解

ServletConfig:servlet配置对象

ServletContext:servlet上下文对象

# 11.ServletConfig接口

servletconfig是一个配置对象,主要是把servlet的初始化参数封装到这个对象中,

在一个网站中可能存在多个ServletConfig对象,一个ServletConfig就封装了一个servlet配置信息

tomcat服务器会把web.xml中每个<servlet>标签封装到ServletConfig对象中,然后将对象传入init(ServletConfig servletconfig)函数中

## 11.1配置初始化参数

|  |
| --- |
| <servlet>  <servlet-name>ConfigDemo</servlet-name>  <servlet-class>com.company.ConfigDemo</servlet-class>  <!--配置servlet的初始化参数-->  <init-param>  <param-name>path</param-name>  <param-value>F:/b.txt</param-value>  </init-param> //注意要在下面配置初始化参数,不能到<servlet>下一行配置  </servlet> |

servlet获取初始化参数

config.getInitParameter(“name”);//根据参数名称获取参数值

config.getInitParameterNames();//获取所有参数名称

# 12.ServletContext接口

## 12.1引入

ServletContext对象叫做servlet上下文对象,整个网站只会有一个

ServletContext对象,代表整个网站的环境信息

## 12.2获取ServletContext对象

this.getServletConfig().getServletContext();通过ServletConfig对象来获取到

ServletContext对象

## 12.3ServletContext的5大作用

1)获取web的上下文路径,指的是项目在tomcat服务器中运行的路径,例如/web

String getContextPath()

2)获取全局参数,如果整个网站的程序都要这个参数,那么设置为全局参数

String getInitParameter(String name);

Enumeration getInitParameterNames();

3)和域对象相关的

域对象是用来在不同的Servlet之间共享数据的,保存数据,获取数据

Servlet三种域对象:ServletContext,HttpServletRequest,HttpSeesion

void setAttribute(String name,Object object);

Object getAttribute(String name);

void removeAttribute(String name);

4)请求转发的

RequestDispatcher getRequestDispatcher(String path);

页面的跳转有两种方式:

a请求重定向

1. 地址栏会发生改变,变成重定向到的地址
2. 可以跳转到项目内的资源,也可以跳转到其他项目的资源
3. 是浏览器行为
4. 浏览器发出两次请求,如果使用请求作为域对象来共享数据

b.请求转发

1) 地址栏不改变

2) 可以跳转到项目内的资源,不可以跳转到其他项目的资源

3) 服务器行为

4) 浏览器只发出一次请求,可以使用请求作为域对象来共享数据

5)读取web项目的资源文件

由于不能使用绝对路径,而相对路径又是tomcat下的bin目录,因此要使用下面的方法

String getRealPath(String path);

InputStream getResourceAsStream(String path);

java.net.URL getResource(String path);

范例1):web上下文路径

|  |
| --- |
| public class Context extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  /\*\*  \* 获取web上下路径  \*/  //1.获取  //ServletContext context = this.getServletConfig().getServletContext();  //上面的下发太麻烦,用下面的简写  ServletContext context = this.getServletContext();  //获取path  String path = context.getContextPath();  System.out.println(path);  /\*\*  \* web上下文路径主要用于路径问题  \* 请求重定向  \*/  response.sendRedirect(path+ "/hello.html");  }  } |

范例2:全局参数

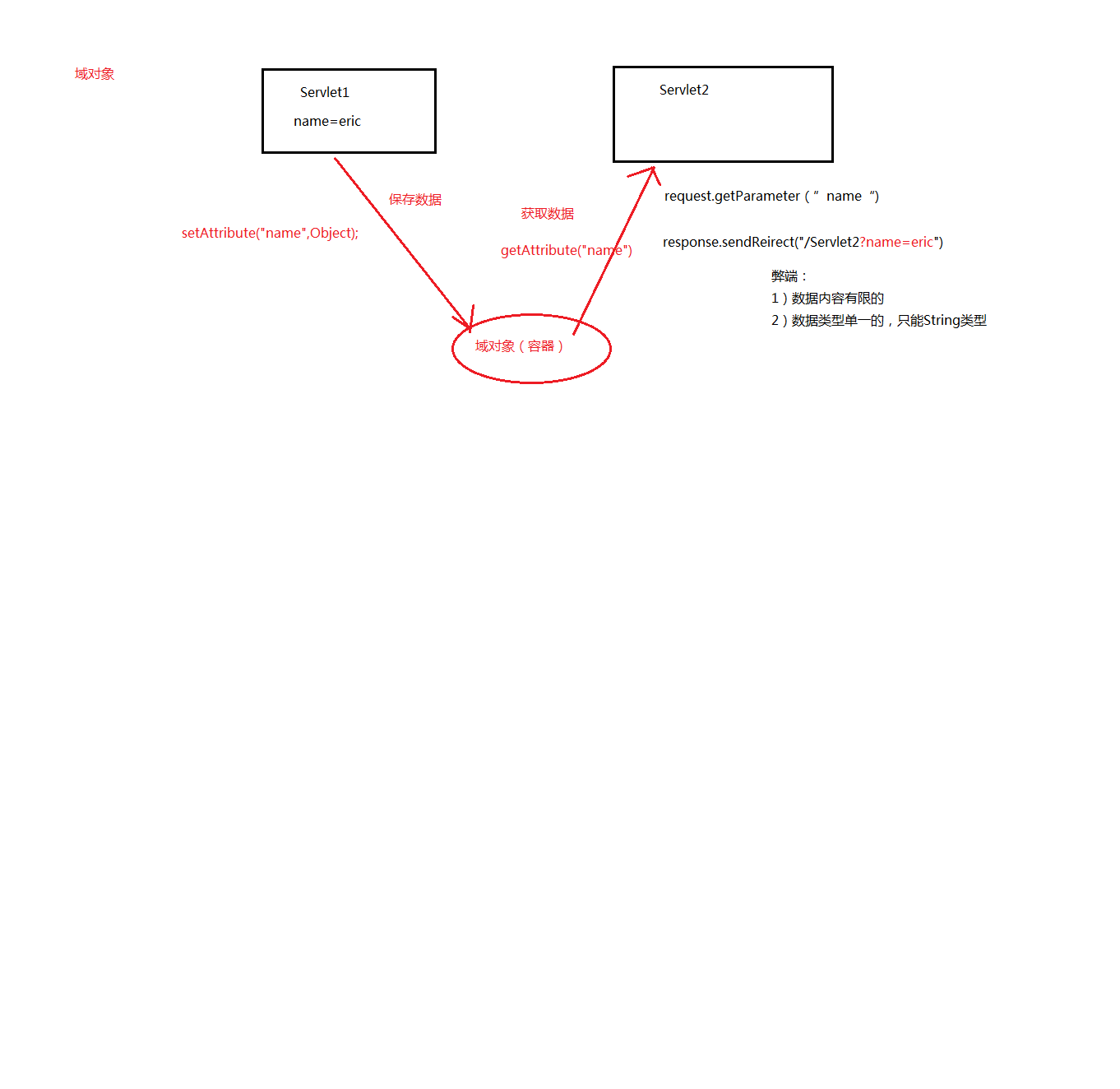
配置全局参数是到web.xml文件中配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  version="3.1">  <!--全局配置-->  <context-param>  <param-name>AAA</param-name>  <param-value>AAA'vlaue</param-value>  </context-param>  <context-param>  <param-name>BBB</param-name>  <param-value>BBB'vlaue</param-value>  </context-param>  </web-app> |

|  |
| --- |
| public class ContextDemo2 extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  ServletContext context = this.getServletContext();  String str1 = context.getInitParameter("AAA");  System.out.println(str1);  Enumeration<String> enums = context.getInitParameterNames();  while(enums.hasMoreElements()){  String paramName = enums.nextElement();  String paramValue = context.getInitParameter(paramName);  System.out.println(paramName + "=" + paramValue);  }  }  } |

范例3:域对象

用重定向来共享数据的弊端:



范例4:域对象

1)存入数据

|  |
| --- |
| public class ScopeDemo1 extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  /\*\*  \* 把数据存储到ServletContext域对象  \*/  ServletContext context = this.getServletContext();  context.setAttribute("name","eric");  System.out.println("保存成功!");  }  } |

2)取出数据

|  |
| --- |
| public class ScopeDemo2 extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  /\*\*  \* 从ServletContext域对象中取出数据  \*/  ServletContext context = this.getServletContext();  String name = (String)context.getAttribute("name");  System.out.println("取出来内容:" + name);  }  } |

范例5:转发

转发端:

|  |
| --- |
| public class ForwardDemo extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  /\*\*  \* 转发  \*/  request.setAttribute("name","linwei");  // this.getServletContext().getRequestDispatcher("/getDataServlet").forward(request,response);  //上面一行太繁琐,简写如下  request.getRequestDispatcher("/getDataServlet").forward(request,response);;  }  } |

转到的另一个程序中:

|  |
| --- |
| public class GetDataServlet extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //从request域对象中取出数据  String name = (String)request.getAttribute("name");  System.out.println("取出的数据:" + name);  }  } |

范例6:web资源文件

|  |
| --- |
| public class ResourceDemo extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  /\*\*  \* 读取资源文件  \*/  //方式1  String path = this.getServletContext().getRealPath("/WEB-INF/classes/news.properties");  //方式2  // InputStream in = this.getServletContext().getResourceAsStream("/WEB-INF/classes/news.properties");  //方式3  // URL url = this.getServletContext().getResource("/WEB-INF/classes/news.properties");  FileInputStream in = new FileInputStream(path);  Properties prop = new Properties();  prop.load(in);  System.out.println(prop.getProperty("name"));  System.out.println(prop.getProperty("passwd"));  }  } |

# 13.路径问题

服务器行为 / 代表从当前项目的根目标开始

浏览器行为 / 代表从当前站点的根目录开始

相对路径:当前路径相对于java命令运行目录

在web项目中:java命令运行的目录就是在tomcat的bin目录