Servlet规范

一。Servlet规范介绍：

1.servlet规范来自于JAVAEE规范中的一种

2.作用：

1）在Servlet规范中，指定【动态资源文件】开发步骤

2）在Servlet规范中，指定Http服务器调用动态资源文件规则

3）在Servlet规范中，指定Http服务器管理动态资源文件实例对象规则

二。Servlet接口实现类：

1.Servlet接口来自于Servlet规范下一个接口，这个接口存在Http服务器

提供jar包

2.Tomcat服务器下lib文件有一个servlet-api.jar存放Servlet接口（javax.servlet.Servlet接口）

3.Servlet规范中任务，Http服务器能调用的【动态资源文件】必须是一个Servlet接口实现类

例子：

class Student{

//不是动态资源文件，Tomcat无权调用

}

class Teacher implements Servlet{

//合法动态资源文件，Tomcat有权利调用

Servlet obj = new Teacher();

obj.doGet()

}

三。Servlet接口实现类开发步骤

第一步：创建一个Java类继承与HttpServlet父类，使之成为一个Servlet接口实现类

第二步：重写HttpServlet父类两个方法。doGet或则doPost

get

浏览器------》oneServlet.doGet()

post

浏览器------》oneServlet.doPost()

第三步：将Servlet接口实现类信息【注册】到Tomcat服务器

【网站】--->【web】---》【WEB-INF】--->web.xml

<!--将Servlet接口实现类类路径地址交给Tomcat-->

<servlet>

<servlet-name>mm</servlet-name> <!--声明一个变量存储servlet接口实现类类路径-->

<servlet-class>com.bjpowernode.controller.OneServlet</servlet-class><!--声明servlet接口实现类类路径-->

</servlet>

Tomcat String mm = "com.bjpowernode.controller.OneServlet"

<!--为了降低用户访问Servlet接口实现类难度，需要设置简短请求别名-->

<servlet-mapping>

<servlet-name>mm</servlet-name>

<url-pattern>/one</url-pattern> <!--设置简短请求别名,别名在书写时必须以"/"为开头-->

</servlet-mapping>

如果现在浏览器向Tomcat索要OneServlet时地址

http://localhost:8080/myWeb/one

三。Servlet对象生命周期:

1.网站中所有的Servlet接口实现类的实例对象，只能由Http服务器负责额创建。

开发人员不能手动创建Servlet接口实现类的实例对象

2.在默认的情况下，Http服务器接收到对于当前Servlet接口实现类第一次请求时

自动创建这个Servlet接口实现类的实例对象

在手动配置情况下，要求Http服务器在启动时自动创建某个Servlet接口实现类

的实例对象

<servlet>

<servlet-name>mm</servlet-name> <!--声明一个变量存储servlet接口实现类类路径-->

<servlet-class>com.bjpowernode.controller.OneServlet</servlet-class>

<load-on-startup>30<load-on-startup><!--填写一个大于0的整数即可-->

</servlet>

3.在Http服务器运行期间，一个Servlet接口实现类只能被创建出一个实例对象

4.在Http服务器关闭时刻，自动将网站中所有的Servlet对象进行销毁

四。HttpServletResponse接口

1.介绍:

1)HttpServletResponse接口来自于Servlet规范中，在Tomcat中存在servlet-api.jar

2)HttpServletResponse接口实现类由Http服务器负责提供

3)HttpServletResponse接口负责将doGet/doPost方法执行结果写入到【响应体】交给浏览器

4)开发人员习惯于将HttpServletResponse接口修饰的对象称为【响应对象】

2.主要功能:

1) 将执行结果以二进制形式写入到【响应体】

2) 设置响应头中[content-type]属性值，从而控制浏览器使用

对应编译器将响应体二进制数据编译为【文字，图片，视频，命令】

3) 设置响应头中【location】属性，将一个请求地址赋值给location.

从而控制浏览器向指定服务器发送请求

五。HttpServletRequest接口

1.介绍:

1)HttpServletRequest接口来自于Servlet规范中，在Tomcat中存在servlet-api.jar

2)HttpServletRequest接口实现类由Http服务器负责提供

3)HttpServletRequest接口负责在doGet/doPost方法运行时读取Http请求协议包中信息

4)开发人员习惯于将HttpServletRequest接口修饰的对象称为【请求对象】

2.作用:

1)可以读取Http请求协议包中【请求行】信息

2)可以读取保存在Http请求协议包中【请求头】或则【请求体】中请求参数信息

3)可以代替浏览器向Http服务器申请资源文件调用

六。请求对象和响应对象生命周期

1.在Http服务器接收到浏览器发送的【Http请求协议包】之后，

自动为当前的【Http请求协议包】生成一个【请求对象】和一个【响应对象】

2.在Http服务器调用doGet/doPost方法时，负责将【请求对象】和【响应对象】

作为实参传递到方法，确保doGet/doPost正确执行

3.在Http服务器准备推送Http响应协议包之前，负责将本次请求关联的【请求对象】和【响应对象】

销毁

\*\*\*【请求对象】和【响应对象】生命周期贯穿一次请求的处理过程中

\*\*\*【请求对象】和【响应对象】相当于用户在服务端的代言人

七。欢迎资源文件

1.前提：

用户可以记住网站名，但是不会记住网站资源文件名

2.默认欢迎资源文件:

用户发送了一个针对某个网站的【默认请求】时，

此时由Http服务器自动从当前网站返回的资源文件

正常请求： http://localhost:8080/myWeb/index.html

默认请求： http://localhost:8080/myWeb/

3.Tomcat对于默认欢迎资源文件定位规则

1）规则位置：Tomcat安装位置/conf/web.xml

2）规则命令：<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

4.设置当前网站的默认欢迎资源文件规则

1)规则位置: 网站/web/WEB-INF/web.xml

2)规则命令: <welcome-file-list>

<welcome-file>login.html</welcome-file>

</welcome-file-list>

3)网站设置自定义默认文件定位规则，此时Tomcat自带定位规则将失效

八。Http状态码

1.介绍:

1)由三位数字组成的一个符号。

2)Http服务器在推送响应包之前，根据本次请求处理情况

将Http状态码写入到响应包中【状态行】上

3)如果Http服务器针对本次请求，返回了对应的资源文件。

通过Http状态码通知浏览器应该如何处理这个结果

如果Http服务器针对本次请求，无法返回对应的资源文件

通过Http状态码向浏览器解释不能提供服务的原因

2.分类：

1）组成 100---599；分为5个大类

2）1XX :

最有特征 100; 通知浏览器本次返回的资源文件

并不是一个独立的资源文件，需要浏览器在接收

响应包之后，继续向Http服务器所要依赖的其他资源文件

3) 2XX：

最有特征200，通知浏览器本次返回的资源文件是一个

完整独立资源文件，浏览器在接收到之后不需要所要

其他关联文件

4）3xx:

最有特征302，通知浏览器本次返回的不是一个资源文件内容

而是一个资源文件地址，需要浏览器根据这个地址自动发起

请求来索要这个资源文件

response.sendRedirect("资源文件地址")写入到响应头中

location

而这个行为导致Tomcat将302状态码写入到状态行

5）4XX:

404: 通知浏览器，由于在服务端没有定位到被访问的资源文件

因此无法提供帮助

405：通知浏览器，在服务端已经定位到被访问的资源文件（Servlet）

但是这个Servlet对于浏览器采用的请求方式不能处理

6）5xx:

500:通知浏览器，在服务端已经定位到被访问的资源文件（Servlet）

这个Servlet可以接收浏览器采用请求方式，但是Servlet在处理

请求期间，由于Java异常导致处理失败

九。多个Servlet之间调用规则:

1.前提条件：

某些来自于浏览器发送请求，往往需要服务端中多个Servlet协同处理。

但是浏览器一次只能访问一个Servlet，导致用户需要手动通过浏览器

发起多次请求才能得到服务。

这样增加用户获得服务难度，导致用户放弃访问当前网站【98k,AKM】

2.提高用户使用感受规则：

无论本次请求涉及到多少个Servlet,用户只需要【手动】通知浏览器发起

一次请求即可

3.多个Servlet之间调用规则：

1）重定向解决方案

2）请求转发解决方案

十。重定向解决方案:

1.工作原理: 用户第一次通过【手动方式】通知浏览器访问OneServlet。

OneServlet工作完毕后，将TwoServlet地址写入到响应头

location属性中，导致Tomcat将302状态码写入到状态行。

在浏览器接收到响应包之后，会读取到302状态。此时浏览器

自动根据响应头中location属性地址发起第二次请求，访问

TwoServlet去完成请求中剩余任务

2.实现命令:

response.sendRedirect("请求地址")

将地址写入到响应包中响应头中location属性

3.特征:

1)请求地址：

既可以把当前网站内部的资源文件地址发送给浏览器 （/网站名/资源文件名）

也可以把其他网站资源文件地址发送给浏览器(http://ip地址:端口号/网站名/资源文件名)

2)请求次数

浏览器至少发送两次请求，但是只有第一次请求是用户手动发送。

后续请求都是浏览器自动发送的。

3) 请求方式：

重定向解决方案中，通过地址栏通知浏览器发起下一次请求，因此

通过重定向解决方案调用的资源文件接收的请求方式一定是【GET】

4.缺点:

重定向解决方案需要在浏览器与服务器之间进行多次往返，大量时间

消耗在往返次数上，增加用户等待服务时间

十一。请求转发解决方案:

1.原理： 用户第一次通过手动方式要求浏览器访问OneServlet。

OneServlet工作完毕后，通过当前的请求对象代替浏览器

向Tomcat发送请求，申请调用TwoServlet。

Tomcat在接收到这个请求之后，自动调用TwoServlet来

完成剩余任务

2.实现命令： 请求对象代替浏览器向Tomcat发送请求

//1.通过当前请求对象生成资源文件申请报告对象

RequestDispatcher report = request.getRequestDispatcher("/资源文件名");一定要以"/"为开头

//2.将报告对象发送给Tomcat

report.forward(当前请求对象，当前响应对象)

3.优点：

1）无论本次请求涉及到多少个Servlet,用户只需要手动通过浏览器发送一次请求

2) Servlet之间调用发生在服务端计算机上，节省服务端与浏览器之间往返次数

增加处理服务速度

4.特征:

1)请求次数

在请求转发过程中，浏览器只发送一次请求

2）请求地址

只能向Tomcat服务器申请调用当前网站下资源文件地址

request.getRequestDispathcher("/资源文件名") \*\*\*\*不要写网站名\*\*\*\*

3）请求方式

在请求转发过程中，浏览器只发送一个了个Http请求协议包。

参与本次请求的所有Servlet共享同一个请求协议包，因此

这些Servlet接收的请求方式与浏览器发送的请求方式保持一致

十二。多个Servlet之间数据共享实现方案：

1.数据共享：OneServlet工作完毕后，将产生数据交给TwoServlet来使用

2.Servlet规范中提供四种数据共享方案

1.ServletContext接口

2.Cookie类

3.HttpSession接口

4.HttpServletRequest接口

十三。ServletContext接口:

1.介绍：

1）来自于Servlet规范中一个接口。在Tomcat中存在servlet-api.jar

在Tomcat中负责提供这个接口实现类

2）如果两个Servlet来自于同一个网站。彼此之间通过网站的ServletContext

实例对象实现数据共享

3）开发人员习惯于将ServletContext对象称为【全局作用域对象】

2.工作原理：

每一个网站都存在一个全局作用域对象。

这个全局作用域对象【相当于】一个Map.

在这个网站中OneServlet可以将一个数据

存入到全局作用域对象，当前网站中其他

Servlet此时都可以从全局作用域对象得到

这个数据进行使用

3.全局作用域对象生命周期：

1）在Http服务器启动过程中，自动为当前网站在内存中创建

一个全局作用域对象

2）在Http服务器运行期间时，一个网站只有一个全局作用域对象

3）在Http服务器运行期间，全局作用域对象一直处于存活状态

4）在Http服务器准备关闭时，负责将当前网站中全局作用域对象

进行销毁处理

\*\*\*\*\*全局作用域对象生命周期贯穿网站整个运行期间\*\*\*

4.命令实现： 【同一个网站】OneServlet将数据共享给TwoServlet

OneServlet{

public void doGet(HttpServletRequest request,HttpServletResponse response){

//1.通过【请求对象】向Tomcat索要当前网站中【全局作用域对象】

ServletContext application = request.getServletContext();

//2.将数据添加到全局作用域对象作为【共享数据】

application.setAttribute("key1",数据)//只有高级程序员才有资格使用

}

}

TwoServlet{

public void doGet(HttpServletRequest request,HttpServletResponse response){

//1.通过【请求对象】向Tomcat索要当前网站中【全局作用域对象】

ServletContext application = request.getServletContext();

//2.从全局作用域对象得到指定关键字对应数据

Object 数据 = application.getAttribute("key1");

}

}

十四。Cookie

1.介绍:

1)Cookie来自于Servlet规范中一个工具类，存在于Tomcat提供servlet-api.jar中

2)如果两个Servlet来自于同一个网站，并且为同一个浏览器/用户提供服务，此时

借助于Cookie对象进行数据共享

3) Cookie存放当前用户的私人数据，在共享数据过程中提高服务质量

4) 在现实生活场景中，Cookie相当于用户在服务端得到【会员卡】

2.原理:

用户通过浏览器第一次向MyWeb网站发送请求申请OneServlet。

OneServlet在运行期间创建一个Cookie存储与当前用户相关数据

OneServlet工作完毕后，【将Cookie写入到响应头】交还给当前

浏览器。

浏览器收到响应响应包之后，将cookie存储在浏览器的缓存

一段时间之后，用户通过【同一个浏览器】再次向【myWeb网站】发送请求申请TwoServlet时。

【浏览器需要无条件的将myWeb网站之前推送过来的Cookie，写入到请求头】发送过去

此时TwoServlet在运行时，就可以通过读取请求头中cookie中信息，得到OneServlet提供的

共享数据

3.实现命令: 同一个网站 OneServlet 与 TwoServlet 借助于Cookie实现数据共享

OneServlet{

public void doGet(HttpServletRequest request,HttpServletResponse resp){

//1.创建一个cookie对象，保存共享数据（当前用户数据）

Cookie card = new Cookie("key1","abc");

Cookie card1= new Cookie("key2","efg");

\*\*\*\*cookie相当于一个map

\*\*\*\*一个cookie中只能存放一个键值对

\*\*\*\*这个键值对的key与value只能是String

\*\*\*\*键值对中key不能是中文

//2.【发卡】将cookie写入到响应头，交给浏览器

resp.addCookie(card);

resp.addCookie(card1)

}

}

浏览器/用户 <---------响应包 【200】

【cookie: key1=abc; key2=eft】

【】

【处理结果】

浏览器向myWeb网站发送请求访问TwoServlet---->请求包 【url:/myWeb/two method:get】

【

请求参数：xxxx

Cookie key1=abc;key2=efg

】

【】

【】

TwoServlet{

public void doGet(HttpServletRequest request,HttpServletResponse resp){

//1.调用请求对象从请求头得到浏览器返回的Cookie

Cookie cookieArray[] = request.getCookies();

//2.循环遍历数据得到每一个cookie的key 与 value

for(Cookie card:cookieArray){

String key = card.getName(); 读取key "key1"

Strign value = card.getValue();读取value "abc"

提供较好的服务。。。。。。。。

}

}

}

4.Cookie销毁时机:

1.在默认情况下，Cookie对象存放在浏览器的缓存中。

因此只要浏览器关闭，Cookie对象就被销毁掉

2.在手动设置情况下，可以要求浏览器将接收的Cookie

存放在客户端计算机上硬盘上，同时需要指定Cookie

在硬盘上存活时间。在存活时间范围内，关闭浏览器

关闭客户端计算机，关闭服务器，都不会导致Cookie

被销毁。在存活时间到达时，Cookie自动从硬盘上被

删除

cookie.setMaxAge(60); //cookie在硬盘上存活1分钟

十五。HttpSession接口:

1.介绍：

1）HttpSession接口来自于Servlet规范下一个接口。存在于Tomcat中servlet-api.jar

其实现类由Http服务器提供。Tomcat提供实现类存在于servlet-api.jar

2）如果两个Servlet来自于同一个网站，并且为同一个浏览器/用户提供服务，此时

借助于HttpSession对象进行数据共享

3）开发人员习惯于将HttpSession接口修饰对象称为【会话作用域对象】

2.HttpSession 与 Cookie 区别：【面试题】

1）存储位置: 一个在天上，一个在地下

Cookie：存放在客户端计算机（浏览器内存/硬盘）

HttpSession：存放在服务端计算机内存

2）数据类型：

Cookie对象存储共享数据类型只能是String

HttpSession对象可以存储任意类型的共享数据Object

3) 数据数量:

一个Cookie对象只能存储一个共享数据

HttpSession使用map集合存储共享数据，所以可以

存储任意数量共享数据

4）参照物:

Cookie相当于客户在服务端【会员卡】

HttpSession相当于客户在服务端【私人保险柜】

3.命令实现: 同一个网站（myWeb）下OneServlet将数据传递给TwoServlet

OneServlet{

public void doGet(HttpServletRequest request,HttpServletResponse response){

//1.调用请求对象向Tomcat索要当前用户在服务端的私人储物柜

HttpSession session = request.getSession();

//2.将数据添加到用户私人储物柜

session.setAttribute("key1",共享数据)

}

}

浏览器访问/myWeb中TwoServlet

TwoServlet{

public void doGet(HttpServletRequest request,HttpServletResponse response){

//1.调用请求对象向Tomcat索要当前用户在服务端的私人储物柜

HttpSession session = request.getSession();

//2.从会话作用域对象得到OneServlet提供的共享数据

Object 共享数据 = session.getAttribute("key1");

}

}

4.Http服务器如何将用户与HttpSession关联起来

cookie

5.getSession() 与 getSession(false)

1）getSession(): 如果当前用户在服务端已经拥有了自己的私人储物柜.

要求tomcat将这个私人储物柜进行返回

如果当前用户在服务端尚未拥有自己的私人储物柜

要求tocmat为当前用户创建一个全新的私人储物柜

2)getSession(false):如果当前用户在服务端已经拥有了自己的私人储物柜.

要求tomcat将这个私人储物柜进行返回

如果当前用户在服务端尚未拥有自己的私人储物柜

此时Tomcat将返回null

6.HttpSession销毁时机:

1.用户与HttpSession关联时使用的Cookie只能存放在浏览器缓存中.

2.在浏览器关闭时，意味着用户与他的HttpSession关系被切断

3.由于Tomcat无法检测浏览器何时关闭，因此在浏览器关闭时并不会

导致Tomcat将浏览器关联的HttpSession进行销毁（问题）

4.为了解决这个问题，Tomcat为每一个HttpSession对象设置【空闲时间】

这个空闲时间默认30分钟，如果当前HttpSession对象空闲时间达到30分钟

此时Tomcat认为用户已经放弃了自己的HttpSession，此时Tomcat就会销毁

掉这个HttpSession

7.HttpSession空闲时间手动设置

在当前网站/web/WEB-INF/web.xml

<session-config>

<session-timeout>5</session-timeout> <!--当前网站中每一个session最大空闲时间5分钟-->

</session-config>

十六。HttpServletRequest接口实现数据共享

1.介绍：

1） 在同一个网站中，如果两个Servlet之间通过【请求转发】方式进行调用，

彼此之间共享同一个请求协议包。而一个请求协议包只对应一个请求对象

因此servlet之间共享同一个请求对象，此时可以利用这个请求对象在两个

Servlet之间实现数据共享

2） 在请求对象实现Servlet之间数据共享功能时，开发人员将请求对象称为【请求作用域对象】

2.命令实现： OneServlet通过请求转发申请调用TwoServlet时，需要给TwoServlet提供共享数据

OneServlet{

public void doGet(HttpServletRequest req,HttpServletResponse response){

//1.将数据添加到【请求作用域对象】中attribute属性

req.setAttribute("key1",数据); //数据类型可以任意类型Object

//2.向Tomcat申请调用TwoServlet

req.getRequestDispatcher("/two").forward(req,response)

}

}

TwoServlet{

public void doGet(HttpServletRequest req,HttpServletResponse response){

//从当前请求对象得到OneServlet写入到共享数据

Object 数据 = req.getAttribute("key1");

}

}

十七。Servlet规范扩展-----------------监听器接口

1.介绍：

1）一组来自于Servlet规范下接口，共有8个接口。在Tomcat存在servlet-api.jar包

2）监听器接口需要由开发人员亲自实现，Http服务器提供jar包并没有对应的实现类

3）监听器接口用于监控【作用域对象生命周期变化时刻】以及【作用域对象共享数据变化时刻】

2.作用域对象：

1）在Servlet规范中，认为在服务端内存中可以在某些条件下为两个Servlet之间提供

数据共享方案的对象，被称为【作用域对象】

2）Servlet规范下作用域对象:

ServletContext： 全局作用域对象

HttpSession : 会话作用域对象

HttpServletRequest:请求作用域对象

3.监听器接口实现类开发规范：三步

1）根据监听的实际情况，选择对应监听器接口进行实现

2）重写监听器接口声明【监听事件处理方法】

3）在web.xml文件将监听器接口实现类注册到Http服务器

4.ServletContextListener接口：

1）作用：通过这个接口合法的检测全局作用域对象被初始化时刻以及被销毁时刻

2）监听事件处理方法：

public void contextInitlized（） ：在全局作用域对象被Http服务器初始化被调用

public void contextDestory(): 在全局作用域对象被Http服务器销毁时候触发调用

5.ServletContextAttributeListener接口:

1)作用：通过这个接口合法的检测全局作用域对象共享数据变化时刻

2）监听事件处理方法：

public void contextAdd():在全局作用域对象添加共享数据

public void contextReplaced():在全局作用域对象更新共享数据

public void contextRemove():在全局作用域对象删除共享数据

6.全局作用域对象共享数据变化时刻

ServletContext application = request.getServletContext();

application.setAttribute("key1",100); //新增共享数据

application.setAttribute("key1",200); //更新共享数据

application.removeAttribute("key1"); //删除共享数据

十八：Servlet规范扩展-----------------Filter接口(过滤器接口)

1.介绍：

1）来自于Servlet规范下接口，在Tomcat中存在于servlet-api.jar包

2）Filter接口实现类由开发人员负责提供，Http服务器不负责提供

3)Filter接口在Http服务器调用资源文件之前，对Http服务器进行拦截

2.具体作用：

1）拦截Http服务器，帮助Http服务器检测当前请求合法性

2）拦截Http服务器，对当前请求进行增强操作

3.Filter接口实现类开发步骤：三步

1）创建一个Java类实现Filter接口

2）重写Filter接口中doFilter方法

3）web.xml将过滤器接口实现类注册到Http服务器

4.Filter拦截地址格式

1) 命令格式:

<filter-mapping>

<filter-name>oneFilter</filter-name>

<url-pattern>拦截地址</url-pattern>

</filter-mapping>

2) 命令作用：

拦截地址通知Tomcat在调用何种资源文件之前需要调用OneFilter过滤进行拦截

3）要求Tomcat在调用某一个具体文件之前，来调用OneFilter拦截

<url-pattern>/img/mm.jpg</url-pattern>

4）要求Tomcat在调用某一个文件夹下所有的资源文件之前，来调用OneFilter拦截

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

5）要求Tomcat在调用任意文件夹下某种类型文件之前，来调用OneFilter拦截

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

6)要求Tomcat在调用网站中任意文件时，来调用OneFilter拦截

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