Servlet

一 Servlet 概念

1 servlet 是什么

1.A servlet 是 sun 公司提供的用于开发动态 web 资源的技术

1.B 按照 servlet 的规范开发的 servlet 可以放在 servlet 容器(Tomcat)中运行

2 开发 servlet 步骤

2.A 写一个类，实现 servlet 接口，并实现其中的方法

2.B 在 web.xml 中为 servlet 配置对外访问路径

获取通过注解配置对外访问路径

二 开发 Servlet

1 Servlet 继承结构

1.A Servlet 接口

1.A.a Servlet 接口提供一个 Servlet 的基本功能

1.A.b void init(ServletConfig var1)

初始化 servlet

1.A.c `void destroy();`

销毁 servlet

1.A.d `void service(ServletRequest var1, ServletResponse var2)`

处理请求的核心方法

1.B 抽象类 GenericServlet `implements` Servlet

1.B.a 实现了 Servlet 接口，并实现了大部分方法，但 Service 方法没有实现，这个方法是处理请求的核心方法

1.C 抽象类 HttpServlet `extends` GenericServlet，

1.C.a 实现了 service 方法，在 servlice 方法中根据不同的请求方式调用不同的 doXxx 方法

1.C.b (例如：doPost，doGet),因此我们开发的时候继承了 HttpServlet，并重写了 doGet 和 doPost 方法

1.D 自定义 Servlet `extends` HttpServlet

1.D.a 根据业务需要实现 doXxx()等方法

2 开发步骤

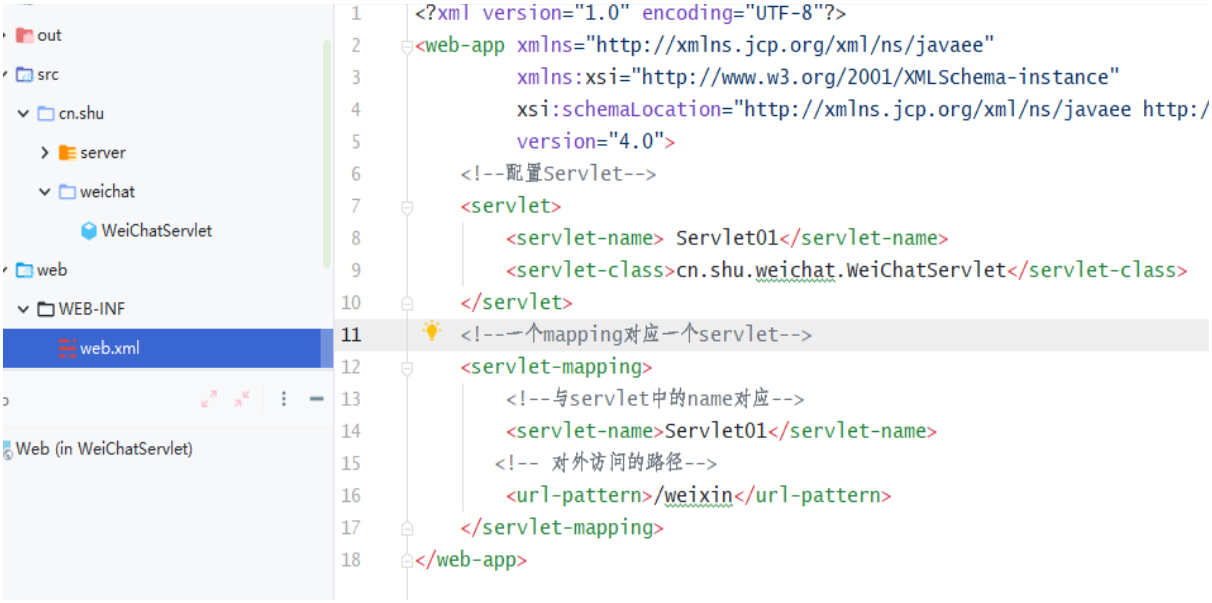
2.A 创建 web 项目

2.B 写一个类，实现 servlet 接口，实现其方法

2.B.a 实现 doGet 和 doPost

2.C 配置对外访问路径

2.C.a 方式一：web.xml



(2.C.a.1) 执行流程：浏览器访问地址时根据/weixin 匹配，

(2.C.a.2) 然后找到 mapping 中的 servlet-name

(2.C.a.3) 匹配 servlet 中的 name

(2.C.a.4) 找到 servlet-name 对应的包

2.C.b 方式二：注解

2.C.c 注意：

(2.C.c.1) 二种方式共存时，对同一个 Servlet 注册 URL 不能相同，否则服务器无法启动

三　Servlet 生命周期

1　加载(启动 tomcat 时)和实例化 Servlet

1.A Servlet 容器负责加载和实例化 Servlet。加载和实例化可以发生在容器启动时，或者延迟初始化直到容器决定有请求需要处理时(可配置加载时实例化,默认有请求时才实例化)

2　初始化(init)

2.A 实例化(new Servlet)后，调用 init 方法进行初始化

3　服务（service）

3.A init 方法 此后该实例一直贮存在服务器内存中，为后续请求提供服务

4　销毁(destroy)

4.A 销毁之前调用 destroy()方法

4.B 当服务器关闭时，servlet 容器销毁时，servlet 实例会随之销毁，销毁之前调用 destroy

5　卸载

四　HttpServletRequest 接口

代表请求的对象

1 继承结构

1.A 接口 ServletRequest

1.B 接口 HttpServletRequest extends ServletRequest

在父类基础上增加了一些与 HTTP 有关的方法

2 常用方法

2.A request.getContextPath() String

获取当前 web 应用的名称，如果省略 web 应用名称返回空白

2.B request.getHeader("HOST") String

获取请求头信息，根据请求头 key，可以不区分大小写

2.C 获取请求参数

2.C.a request.getParameter("username");

根据 key 获取 value 返回 数组的第一个值

2.C.b request.getParameterValues("username");

返回数组，因为一个前端的表单 name 或者 get 参数里的 name 可以重复可以对应多个 value

2.C.c request.getParameterMap(); Map<String,String[]>

返回 key-value 的 map，因为一个前端的表单 name 可能对应多个 value (String[])

2.C.d 注意中文乱码问题

浏览器提交的编码是 utf-8 的，服务器(tomcat)默认使用的编码为 iso8859-1(这个编码不支持中文)

(2.C.d.1) 解决 POST 提交乱码：

解决

如果为 post 提交，请求参数通过请求实体提交，可以设置请求实体内容的编码格式

```
request.setCharacterEncoding("utf-8");
```

(2.C.d.2) 解决 GET 提交乱码

如果为get提交，需要手动编解码解决获取二进制(解码)

```
byte[] bytes= "username".getBytes("iso8859-1");
```

然后根据某个编码(utf-8)再编码，该方法对 post 也有效

```
String name=new String(bytes,"utf-8");
```

2.D 请求转发

2.D.a 特点

(2.D.a.1) 一次请求一次响应

(2.D.a.2) 地址栏不会发生变化，也就是对浏览器没有任何影响，操作只是在服务器端完成

(2.D.a.3) 请求转发只能在同一 tomcat 服务器内的同一个 web 应用(一个 tomcat 对应多个应用)进行转发

(2.D.a.4) 可以多级转发 A -> B -> C...

2.D.b 方法

(2.D.b.1) 获取转发器

```
request.getRequestDispatcher("/weixin")
```

(2.D.b.2) 转发 request 和 response

```
.forward(request,response);
```

(2.D.b.3) 注意

“/weixin”为 servlet 的 urlPatterns 值，或者是页面(例/index.jsp)即对外访问路径

“/”可以省略

转发后之前 response.getWriter().write 的内容是在缓冲区中，会清空，即转发之前不会输出

2.E 作为域对象使用

2.E.a 域对象

如果一个对象具有一个可以被看见的范围，利用该对象的 map 可以在整个范围内共享数据，那么这个对象就是域对象。

2.E.b 方法

(2.E.b.1) 请求转发前设置，将姓名放到域中

```
request.setAttribute("name","shu");
```

(2.E.b.2) 目的 Servlet 接收或 jsp 页面

```
request.getAttribute("name")
```

(2.E.b.3) 从 域 中 删 除

```
request.removeAttribute("name");
```

(2.E.b.4) 只是请求转发，针对请求的数据，所以 response 没有这二个方法

3 生命周期及作用范围

3.A 生命周期：请求开始时创建，请求结束时销毁

3.B 作用范围：整个请求链, 即包括请求转发的下一个 servlet

五 HttpServletResponse 接口

1 概述

1.A 代表响应的对象

2 继承结构

2.A 接口 ServletResponse

2.B 接口 HttpServletResponse extends ServletResponse

在父类基础上增加了一些与 HTTP 有关的方法

3 功能

3.A 设置响应状态码

```
response.setStatus(666);
```

3.B 设置响应头

```
response.setHeader("set-cookie","pord=phone");

response.setHeader("Content-Type","text/html;charset=utf-8");
```

3.C 设置响应实体内容

3.C.a 方法

```
// 字符流 需要设置请求头编码，不然可能乱码
response.getWriter().write("你好 Writer");
// 字节流 需要设置编码，不然可能乱码
response.getOutputStream().write("你好 OutPutStream".getBytes("utf-8"));
```

3.C.b 注意

字节流和字符流不能同时使用,网页可能会出现错误码 500

3.C.c 乱码问题

由于服务器发送的编码和浏览器的解码不一致，字符流服务器(tomcat)默认使用的是iso8859-1，浏览器读取数据时默认使用的是平台(系统)码，即 GBK。

(3.C.c.1) **字符流：字符流不能设置编码，只能通过设置请求头。**

可以指定浏览器读取数据时使用 utf-8 读取，同时使用 utf-8 发送数据。
告诉浏览器发送的数据格式和编码，设置了这个头，服务器发送数据时也会默认此编码 发送 。

```
response.setHeader("Content-Type","text/html;charset=utf-8");
```

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

(3.C.c.2) **字节流：字节流可以设置发送数据的编码**

在字符流设置 header 基础上多一步设置” 字符串编码 ”, 和发送编码无关

```
response.getOutputStream().write("你好 OutPutStream".getBytes("utf-8"));
```

字节流输出中文： (实际中不会用)

```
// 使用字节流的方式输出中文:
ServletOutputStream outputStream = response.getOutputStream();
// 设置浏览器默认打开的时候采用的字符集
response.setHeader("Content-Type", "text/html;charset=UTF-8");
// 设置中文转成字节数组字符集编码
outputStream.write("中文".getBytes("UTF-8"));
```

3.D 请求重定向

3.D.a 方法

```
response.sendRedirect(request.getContextPath()+"/weixin2");
```

3.D.b 特点：

(3.D.b.1) 二次请求，二次响应

(3.D.b.2) 浏览器地址栏会发生变化

(3.D.b.3) 在浏览器端完成

(3.D.b.4) 可以跳转到当前 web 应用外的网页(即所有网页)

(3.D.b.5) 重定向不能通过 request 对象共享数据，第一次请求响应 request 已销毁

3.E 定时刷新

3.E.a 方法

```
response.setHeader("refresh","5;url="+request.getContextPath()+"/weixin2");
```

延时 5 秒请求重定向

```
response.setHeader("refresh","5;url=http://www.baidu.com");
```

跳转到外部，需加 http 协议，否则会认为是当前 web 应用的目录,会自动加上当前网页的目录，可以设置 0 秒

3.E.b 特点

只是延迟的重定向，因为是通过设置请求头的方式，没有违反一次请求一次响应原则，所以重定向之前可以通过字节流或字符流输出内容

六　ServletContext 接口

1　概述

1.A　代表整个 Web 应用的对象，即一个 web 应用只有一个 ServletContext

2　生命周期

2.A　启动

2.A.a　服务器启动时，web 应用加载时创建 ServletContext，这个对象唯一代表 web 应用

2.B　销毁

2.B.a　直到服务器关闭，web 应用销毁时，随着其销毁

3　获取 ServletContext 对象

3.A　在当前 web 应用的任意 Servlet 中获取

3.A.a　this.getServletContext()

4　ServletContext 功能

4.A　获取 WEB 应用初始化参数

4.A.a　配置初始化参数

(4.A.a.1)　在 web.xml 中配置初始化参数

RegistServlet.java × web.xml ×

```xml
<?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.
         version="4.0">


    <context-param>
        <param-name>myKey1</param-name>
        <param-value>myValues</param-value>
    </context-param>
    <context-param>
        <param-name>encode</param-name>
        <param-value>utf-8</param-value>
    </context-param>

</web-app>
```

4.A.b Servlet 中获取参数

(4.A.b.1) 如果希望在整个 WEB 应用中配置一些参数，可以通过 ServletContext 对象获取这些参数

```java
ServletContext sc= this.getServletContext();
System.out.println(sc.getInitParameter( s: "myKey1"));//myValues
 System.out.println(sc.getInitParameter( s: "encode"));//utf-8
```

4.B 作为域对象

4.B.a 特点：

(4.B.a.1) 生命周期：和 web 应用的生命一样长

Web 应用启动时创建，关闭时销毁

(4.B.a.2) 作用范围：整个 web 应用

(4.B.a.3) 功能：在整个 web 应用内共享数据

4.B.b API:

和其他域的方法一样的
setAttribute(“key”,”value”)
getAttribute(“key”)
removeAttribute(“key”)
等…

4.C 获取 WEB 资源文件路径

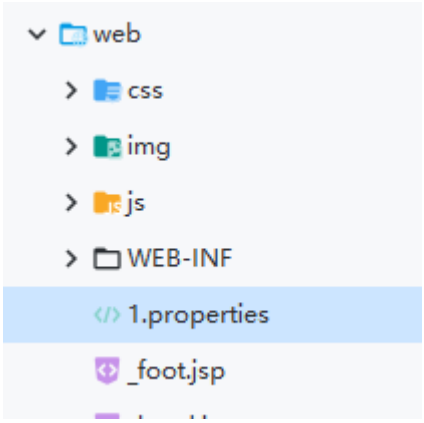
4.C.a String str=sc.getRealPath("1.propertiies")

// E:\kaifa\JAVA\project_idea\EasyMall\out\artifacts\EasyMall_war_exploded*1.propertiie*

4.C.b 返回当前项目下的路径,返回值会自动在参数前面补全 web 项目全路径

4.C.c 即参数为当前项目的相对路径，返回绝对路径

4.C.d

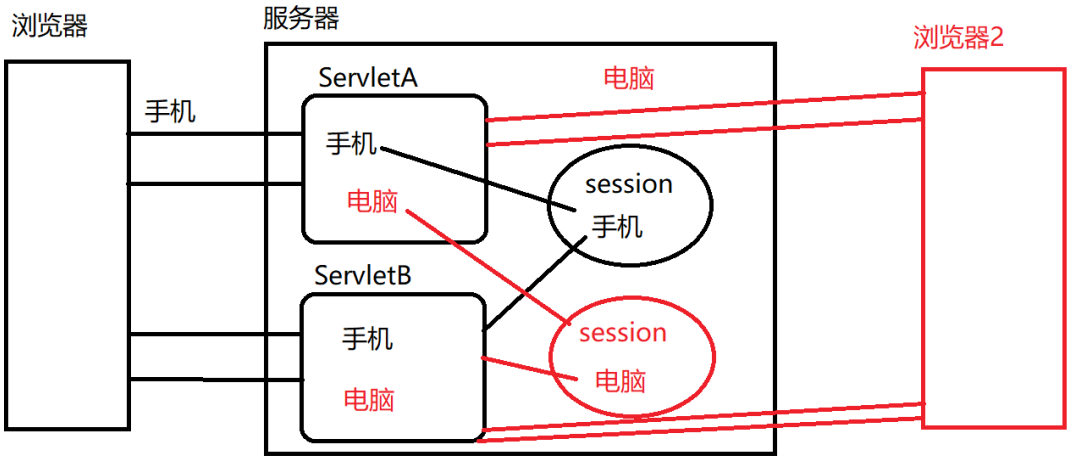


七 SESSION 及 Servlet 中的 HttpSession 接口

1 SESSION 原理

1.A 4.1.session 原理

将会话中产生的数据保存在服务器端



浏览器向服务器发送请求，服务器接收请求参数，然后在服务器中检查是否有为当前浏览器服务的 session，如果有则直接拿来使用，如果没有则创建一个 session，并将数据保存到 session 中

当浏览器再次访问服务器时，服务器可以找到为当前浏览器服务的 session，从中取出数据

通过这种方式也可以保存会话中产生的数据

每个 session 对应一个 id，此 id 通过 cookie（临时）保存在浏览器中，访问服务器时携带 cookie，服务器根据这个 id 判断是哪个 session

2 Session 域

2.A 生命周期

2.A.a 创建

第一次调用 request.getSession()方法时创建 session 对象

```
HttpSession session = request.getSession();
```

如果服务器中已经有 session，会直接拿来使用，如果没有才创建

2.A.b 销毁

(2.A.b.1) 超时死亡：

默认 30 分钟不使用会超时销毁，及时关闭浏览器未超时也不会销毁，Cookie 关闭浏览器时会销毁(如果没有设置超时时间)

可以通过 web.xml 配置超时时间，单位为分钟

```
<session-config>
    <session-timeout>1</session-timeout>
</session-config>
```

(2.A.b.2) 主动杀死：

可以调用 session.invalidate()方法，立刻杀死 session

```
session.invalidate();
```

(2.A.b.3) 意外身亡：

当服务器意外关闭，session 也会销毁

如果服务器正常关闭，session 会被钝化，当服务器启动时再活化

2.B Session 数据的钝化与活化：

由于 session 中保存大量访问网站相关的重要信息，因此过多的 session 数据就会服务器性能的下降，占用过多的内存。因此类似数据库对象的持久化，web 容器也会把不常使用的 session 数据持久化到本地文件或者数据中。这些都是有 web 容器自己完成，不需要用户设定。

不用的 session 数据序列化到本地文件中的过程，就是钝化；

当再次访问需要到该 session 的内容时，就会读取本地文件，再次放入内存中，这个过程就是活化。

2.C 作用范围

整个会话

2.D 作用

整个会话范围内共享数据

2.E 使用

request.getSession(true)：若存在会话则返回该会话，否则新建一个会话。

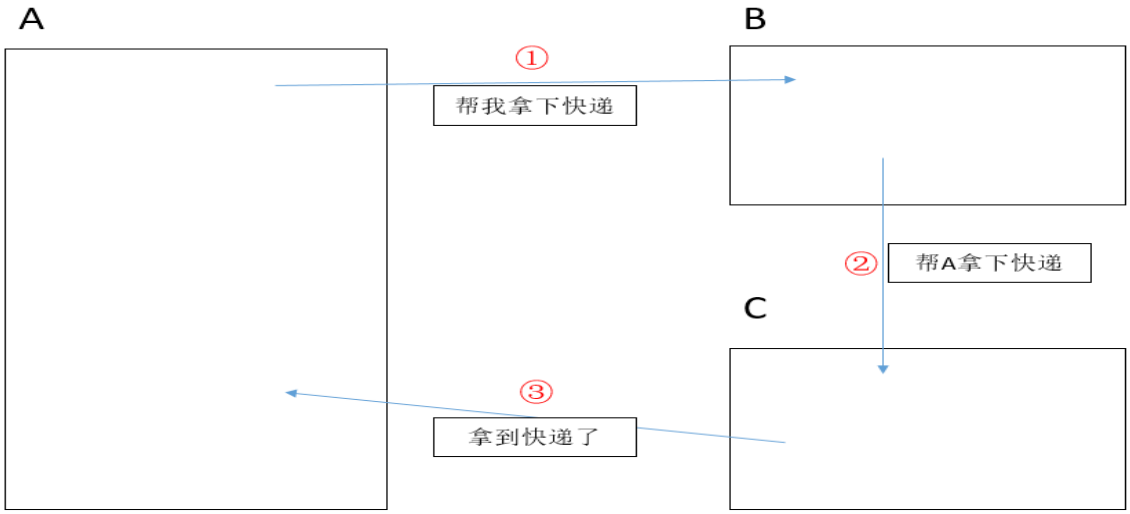
request.getSession(false)：若存在会话则返回该会话，否则返回 NULL

request.getSession()：默认为 true

八 请求转发和请求重定向、定时刷新的区别

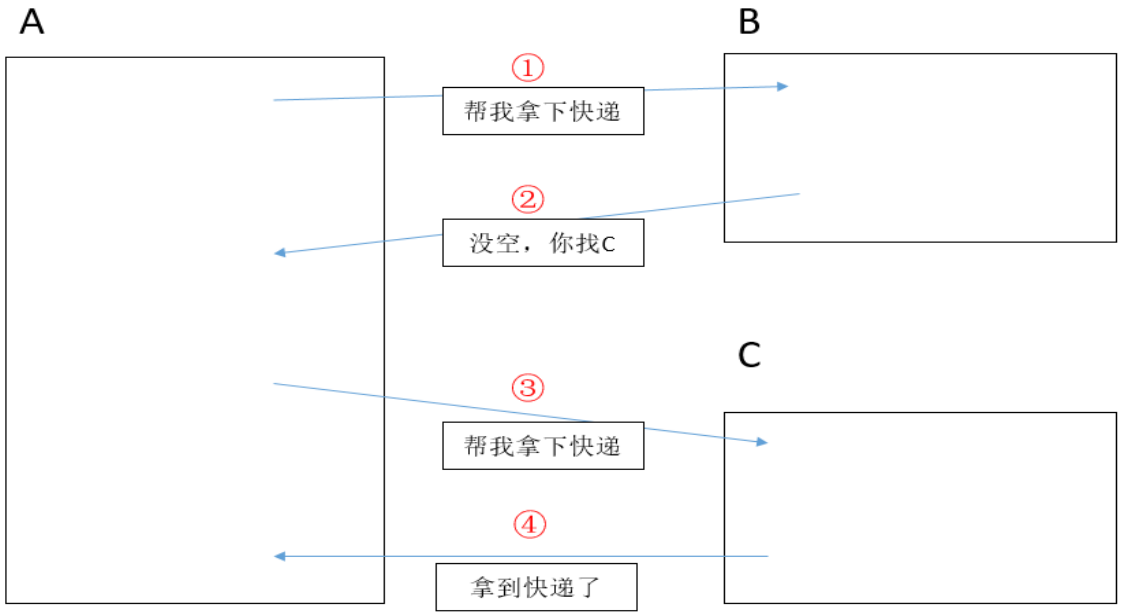
1 请求转发

在服务器端完成，所以在 `request` 对象中



2 请求重定向

在浏览器端完成，所以在 response 对象中(响应给浏览器)



3 定时刷新

只是延迟的重定向，因为时通过设置请求头的方式，没用违反一次请求一次响应原则，所以重定向之前可以输出内容

```
$(document).ready(function () {
    $("#valiimg").click(function () {
        $(this).attr("src","<%= request.getContextPath()%>/Captcha?x="+Math.random())
    })
})
```

在同一个 web 应用内部跳转，需要通过 request 对象携带数据到目的地 -- 请求转发

在同一个 web 应用内部跳转，地址栏不能发生变化 -- 请求转发
需要跳转到其他 web 应用 -- 重定向或者定时刷新，根据是否需要指定多长时间跳转决定
如果在同一个 web 应用跳转，没有其他需求，可以选用任意方式跳转

九 四大作用域总结

1 ServletContext

1.A 生命周期：

Web 应用启动时创建，web 应用销毁时销毁

1.B 作用范围：

整个 web 应用

1.C 功能：

整个 web 应用共享数据

2 Session

2.A 生命周期：

第一次调用 `request.getSession()`方法时创建
默认 30 分钟超时销毁、或者调用 invalidate()方法主动销毁、或服务器宕机意外身亡

2.B 作用范围：

整个会话

2.C 功能：

整个会话范围共享数据

3 Request

3.A 生命周期：

一次请求开始创建，请求结束销毁

3.B 作用范围：

整个请求链

3.C 功能：

整个请求链范围内共享数据

4 pageContext

4.A 生命周期：

访问 jsp 页面时创建，访问 jsp 页面结束时销毁

4.B 作用范围：

当前 jsp 页面

4.C 功能：

整个 jsp 页面共享数据