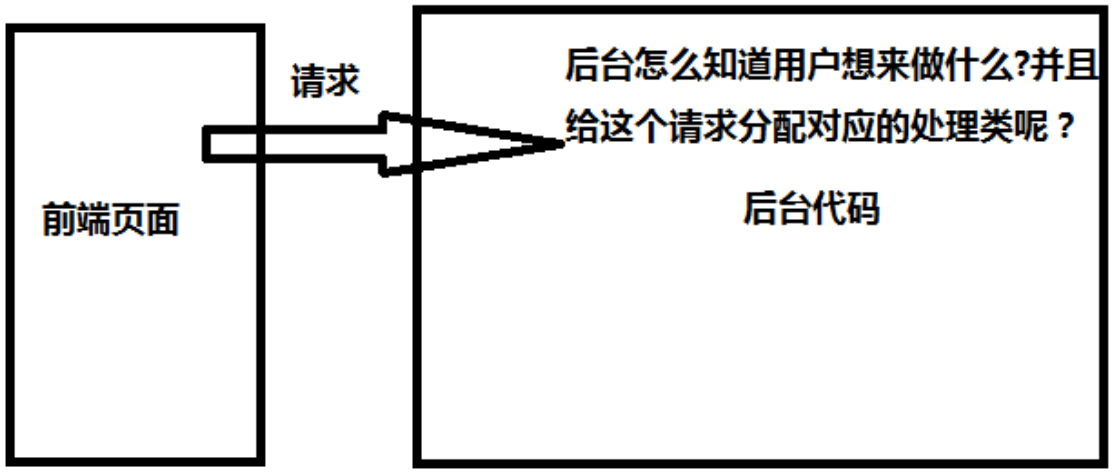
1. 什么是Servlet



(1) Servlet（Server Applet），全称Java Servlet。是用Java编写的服务器端程序，其主要功能在于交互式地浏览和修改数据，生成动态Web内容。狭义的Servlet是指Java语言实现的一个接口，广义的Servlet是指任何实现了这个Servlet接口的类，一般情况下，人们将Servlet理解为后者。

(2) Servlet运行于支持Java的应用服务器中。从实现上讲，Servlet可以响应任何类型的请求，但绝大多数情况下Servlet只用来扩展基于HTTP协议的Web服务器。

(3) Servlet工作模式：

① 客户端发送请求至服务器

② 服务器启动并调用Servlet，Servlet根据客户端请求生成响应内容并将其传给服务器

③ 服务器将响应返回客户端

2. Servlet API

3. 第一个Servlet

(1) 创建一个类实现Servlet接口,重写方法。或继承HttpServlet亦可

```
public class LoginServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        //初始化方法
    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        //处理get/post请求的方法
    }
```

```
    @Override
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {
        //销毁的方法
    }
}
```

(2) 在web.xml文档中配置映射关系

XML中配置好这个Servlet的映射关系：

```
<servlet>
    <servlet-name>自定义名称</servlet-name>
    <servlet-class>处理请求的类的完整路径</servlet-class>
 </servlet>
 <servlet-mapping><!-- mapping 表示映射 -->
   <servlet-name>自定义名称</servlet-name>
   <url-pattern>请求名</url-pattern>
 </servlet-mapping>
```

标签的执行顺序：

请求过来以后->web.xml->servlet-mapping标签中的url-pattern标签中的内容和请求名

进行匹配->匹配成功后找对应的servlet-mapping标签中的servlet-name->

去servlet标签中找和上一个servlet-name相同的name值->去找servlet标签中

的servlet-class中处理类的完整路径

(3) 启动tomcat，在浏览器输入http://localhost:8080/工程名/访问服务器的路径

4. Servlet工作原理

(1) Servlet接口定义了**Servlet与servlet容器**之间的契约。这个契约是：Servlet容器将Servlet类载入内存，并产生Servlet实例和调用它具体的方法。但是要注意的是，**在一个应用程序中，每种Servlet类型只能有一个实例。**

(2)用户请求致使Servlet容器调用Servlet的Service（）方法，**并传入一个ServletRequest对象和一个ServletResponse对象。ServletRequest对象和ServletResponse对象都是由Servlet容器（例如TomCat）封装好的，并不需要程序员去实现，程序员可以直接使用这两个对象。**

(3)ServletRequest中封装了当前的Http请求，因此，开发人员不必解析和操作原始的Http数据。ServletResponse表示当前用户的Http响应，程序员只需直接操作ServletResponse对象就能把响应轻松的发回给用户。

(4)对于每一个应用程序，**Servlet容器还会创建一个ServletContext对象。这个对象中封装了上下文（应用程序）的环境详情。每个应用程序只有一个ServletContext。每个Servlet对象也都有一个封装Servlet配置的ServletConfig对象。**

5. Servlet的生命周期

当客户端首次发送第一次请求后，由容器(web服务器(tomcat))去解析请求, 根据请求找到对应的servlet,判断该类的对象是否存在，不存在则创建servlet实例，调取init()方法 进行初始化操作,初始化完成后调取service()方法,由service()判断客户端的请求方式，如果是get，则执行doGet(),如果是post则执行doPost().处理方法完成后,作出相应结果给客户端.单次请求处理完毕。

当用户发送第二次以后的请求时,会判断对象是否存在,但是不再执行init()，而直接执行service方法,调取doGet()/doPost()方法。

当服务器关闭时调取destroy()方法进行销毁。

四个过程:

(1)实例化 --先创建servlet实例

(2)初始化 --init()

(3)处理请求 ---service()

(4)服务终止 --destory()

6. 请求

HttpServletRequest表示Http环境中的Servlet请求。它扩展于javax.servlet.ServletRequest接口)

常用方法:

1)String getParameter(String name) 根据表单组件名称获取提交数据，返回值是String

注：服务器在接收数据时使用字符串统一接收

2)String[ ] getParameterValues(String name) 获取表单组件对应多个值时的请求数据

3)void setCharacterEncoding(String charset) 指定每个请求的编码(针对post请求才起作用)

4)RequestDispatcher getRequestDispatcher(String path) --跳转页面

返回一个RequestDispatcher对象，该对象的forward( )方法用于转发请求

示例:

```
request.getRequestDispatcher("../success.jsp").forward(request,response);
```

5)存值 request.setAttribute("key",value);

6)取值 request.getAttribute("key");//取值后需要向下转型

示例: String a1=(String)request.getAttribute("uname");

补充1:客户端如何发送数据给服务器

方式1:通过表单 get/post提交

方式2:通过a标签发送数据（get提交）

```
<a href="请求名?key=value&key=value&key=value...">
示例:
<a href="/login?a=10&name=abc&pass=123">
```

这里的key值=表单元素的控件名，value值=表单中控件的value属性值

注:第一个参数使用?拼接,之后的参数使用&拼接，获取数据还是通过 String name=request.getParameter("name");

方式3:通过地址栏直接拼接-get请求

方式4:js提交数据-get请求

location.href="目标请求?key=value&key=value"

注:方式2/3都属于get提交方式,表单提交可以使用get、post提交方式

补充2：处理请求乱码的问题

方式1：setCharacterEncoding("UTF-8");//post提交时管用

方式2: String s=new String(变量名.getBytes("ISO-8859-1"),"UTF-8");//针对于get提交时中文乱码

示例: String s=new String(request.getParameter("key").getBytes("ISO-8859-1"),"GBK");

方式3：修改tomcat中配置文件://使用于get提交

在Tomcat目录结构\conf\server.xml中设置字符集

```
<Connector port="8080" protocol="HTTP/1.1"
            connectionTimeout="20000"
            redirectPort="8443" URIEncoding="UTF-8" />
```

注意:tomcat8.0以后不需要手动设置这个属性了

补充3:get和post的区别

1、GET请求，请求的数据会附加在URL之后，以?分割URL和传输数据，多个参数用&连接。URL的编码格式采用的是ASCII编码，而不是uniclde，即是说所有的非ASCII字符都要编码之后再传输。

POST请求：POST请求会把请求的数据放置在HTTP请求包的包体中。上面的item=bandsaw就是实际的传输数据。

因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。

2、传输数据的大小

在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。

对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST提交数据大小进行限制，Apache、IIS都有各自的配置。

3、安全性

POST的安全性比GET的高。这里的安全是指真正的安全，而不同于上面GET提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过GET请求，用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据还可能会造成Cross-site request frogery攻击

7. 响应

在Service 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。 在HttpServletResponse接口中定义了向客户端发送响应状态码，响应消息头，响应消息体的方法。

常用方法:

void addCookie(Cookie var1);//给这个响应添加一个cookie

void sendRedirect(String var1) ;//发送一条响应码，将浏览器跳转到指定的位置

PrintWriter getWriter() 获得字符流，通过字符流的write(String s)方法可以将字符串设置到response 缓冲区中，随后Tomcat会将response缓冲区中的内容组装成Http响应返回给浏览器端。

setContentType() 设置响应内容的类型

重定向和转发的对比

重定向:response.sendRedirect()

转发:request.getRequestDispatcher("../success.jsp").forward(request,response);

相同点:都用来跳转页面

不同点:

a.重定向时地址栏会改变,request中存储的数据会丢失.转发时地址栏显示的是请求页面的地址,request数据可以保存。

b.转发属于一次请求一次响应,重定向属于两次请求(地址栏修改了两次)两次响应。

补充:使用out对象往页面中输出js或html,css

```
out.print("<script type='text/javascript'>alert('登录失
败');location='../login.jsp'</script>");
```

注:使用js跳转页面，也会丢失request中的数据

8. 会话

request存的值只能在单次请求中保存，保存的数据不能跨页面,当重定向时,request存的值会丢失

session的数据可以在多个页面中共享,即使重定向页面,数据不会丢失

session中可以包含n个request。

会话的概念:从打开浏览器到关闭浏览器,期间访问服务器就称为一次会话

常用方法:

void setAttribute(String key,Object value) 以key/value的形式保存对象值,将数据存储在服务器端

Object getAttribute(String key) 通过key获取对象值

void invalidate() 设置session对象失效

String getId() 获取sessionid,当第一次登录成功后，session会产生一个唯一的id，浏览器之后访问时如果发现id值还是之前id，那么说明 当前访问的属于同一个会话

void setMaxInactiveInterval(int interval) 设定session的非活动时间

示例:

方式1: session.setMaxInactiveInterval(10*60);//设置有效时间为10分钟

```
    方式2:修改web.xml
```

```
<session-config>
    <session-timeout>10</session-timeout>//单位:分钟
</session-config>
```

int getMaxInactiveInterval() 获取session的有效非活动时间(以秒为单位)，默认的有效时间:30分钟

void removeAttribute(String key)

从session中删除指定名称(key)所对应的对象

小结 :让session失效的方式

（1）invalidate() （2）removeAttribute("key") （3）直接关闭浏览器。

示例:使用session验证用户是否登录

补充:

自动刷新到某页面:

注:在head标签中添加该标签，单位:秒

9.获得初始化参数

request.setCharacterEncoding("utf-8");代码的耦合度太高，不便于后期维护修改。可以通过初始化参数实现

实现方式:

（1）web.xml中先定义初始化参数

```
 <servlet>
    <servlet-name></servlet-name>
    <servlet-class></servlet-class>
    <init-param>
       <param-name>encoding</param-name>
       <param-value>utf-8</param-value>
    </init-param>
</servlet>
```

(2)servlet中获得初始化参数，重写init()方法

```
public void init(ServletConfig config) throws ServletException {
          encoding= config.getInitParameter("encoding");
}
```

注意:这种方式的初始化参数仅限于当前servlet中使用。

2.全局初始化参数

(1)定义，context-param是和servlet标签同级别

```
<context-param>
   <param-name>bianma</param-name>
   <param-value>utf-8</param-value>
  </context-param>
```

(2)获得数据

```
@Override  // 请求->init()->service()->doget/dopost->destory();

public void init(ServletConfig config) throws ServletException {
    bianhao=config.getServletContext().getInitParameter("bianma");
}
```

10.servlet3.0

注解(提供给程序读取的信息) -- 注释(提供给程序员看的信息)

注解的格式：@开头的 如:@Override

@WebServlet注解配置Servlet

从Servlet3.0开始，配置Servlet支持注解方式，但还是保留了配置web.xml方式，所有使用Servlet有两种方式：

（1）Servlet类上使用@WebServlet注解进行配置

（2）web.xml文件中配置

@WebServlet常用属性

属性	类型	是否必须	说明
asyncSupported	boolean	否	指定Servlet是否支持异步操作模式
displayName	String	否	指定Servlet显示名称
initParams	webInitParam[]	否	配置初始化参数
loadOnStartup	int	否	标记容器是否在应用启动时就加载这个Servlet，等价于配置文件中的标签
name	String	否	指定Servlet名称
urlPatterns/value	String[]	否	这两个属性作用相同，指定Servlet处理的url

```
<servlet>
    <servlet-name>a</servlet-name>
    <servlet-class>
       <!-- 处理类的完整路径 -->
       com.yhp.web.SelectServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

测试代码如下：

1.UserServlet

通过注解方式配置，web.xml中不需要配置该Servlet

```
@WebServlet(name = "myUserServlet",
    urlPatterns = "/user/test",   //斜杠必须
    loadOnStartup = 1,
    initParams = {
            @WebInitParam(name="name", value="小明"),
            @WebInitParam(name="pwd", value="123456")
    }
)
public class UserServlet extends HttpServlet {
    private static final long serialVersionUID = 7109220574468622594L;

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);

        System.out.println("servlet初始化...");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        PrintWriter pw = response.getWriter();
```

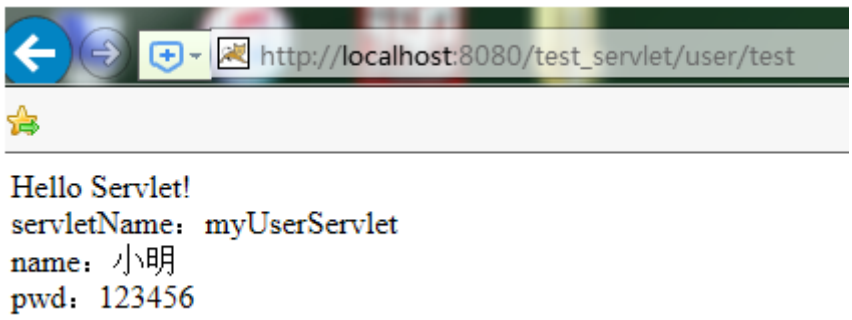
```
        pw.append("Hello Servlet!<br>" );

        //servletName
        pw.append("servletName: " + getServletName() + "<br>");

        //initParam
        ServletConfig servletConfig = this.getServletConfig();
        Enumeration<String> paramNames = servletConfig.getInitParameterNames();
        while (paramNames.hasMoreElements()) {
            String paramName = paramNames.nextElement();
            pw.append(paramName + ": " +
servletConfig.getInitParameter(paramName) + "<br>");
        }

        pw.close();

    }


    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

2.测试结果

2.1应用启动时，servlet就会初始化，因为配置了loadOnStartup=1

```
四月 17, 2017 5:02:01 下午 org.apache.catalina.startup.TaglibUriRule body
信息: TLD skipped. URI: http://java.sun.com/jsp/jstl/xml is already defined
servlet初始化...
四月 17, 2017 5:02:01 下午 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deploying web application directory G:\Java\apache-tomcat-8.0.0-RC1\webapps\docs
四月 17, 2017 5:02:01 下午 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deploying web application directory G:\Java\apache-tomcat-8.0.0-RC1\webapps\examples
四月 17, 2017 5:02:02 下午 org.apache.catalina.core.ApplicationContext log
信息: ContextListener: contextInitialized()
```

2.2访问/user/test，页面上显示结果；表名参数正确初始化以及自定义的servlet名称



3.注

(1).loadOnStartup属性：标记容器是否在启动应用时就加载Servlet，默认不配置或数值为负数时表示客户端第一次请求Servlet时再加载；0或正数表示启动应用就加载，正数情况下，数值越小，加载该Servlet的优先级越高；

实例：//斜杠必须

```
@WebServlet(value="/test1",loadOnStartup=1)
```

(2).name属性：可以指定也可以不指定，通过getServletName()可以获取到，若不指定，则为Servlet的完整类名，如：cn.edu.njit.servlet.UserServlet

(3).urlPatterns/value属性： String[]类型，可以配置多个映射，如：urlPatterns={"/user/test", "/user/example"}

实例：

//斜杠必须

```
@WebServlet(loadOnStartup=1,urlPatterns= {"/test1","/test2"})
```

(4).在使用注解方式时，需要注意：

根元素中不能配置属性metadata-complete="true"，否则无法加载Servlet。metadata-complete属性表示通知Web容器是否寻找注解，默认不写或者设置false，容器会扫描注解，为Web应用程序构建有效的元数据；metadata-complete="true"，会在启动时不扫描注解（annotation）。如果不扫描注解的话，用注解进行的配置就无法生效，例如：@WebServlet

(5).urlPatterns的常用规则：

/*或者/：拦截所有

*.do：拦截指定后缀

/user/test：拦截路径

/user/*.do*、/.do、test*.do都是非法的，启动时候会报错