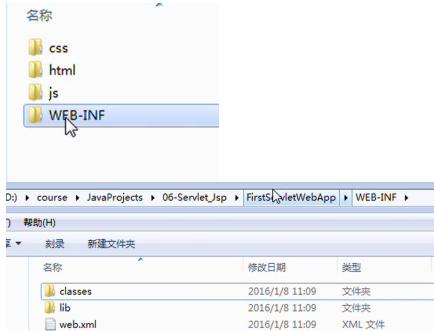


项目目录固定，因此可移植。



Servlet5个重要方法：

```
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;

public class HelloServlet implements Servlet
{
    >> public void init(ServletConfig config) throws ServletException{
    >>
    >>
    >> public void service(ServletRequest request,ServletResponse response){
    >>     >> throws IOException,ServletException{
    >>
    >> }
    >> public void destroy(){
    >>
    >> }
    >> public String getServletInfo(){
    >>     >> return null;
    >> }
    >> public ServletConfig getServletConfig(){
    >>     >> return null;
    >> }
```

第一个例子，一个资源可以编写多个url-pattern

```
</servlet>
<servlet-mapping>
    >> <servlet-name>thisIsServletName</servlet-name>
    >> <!--路径随意编写，但是必须以"/"开始-->
    >> <!--这个路径是一个虚拟的路径，只是代表一个资源的名称-->
    >> <url-pattern>/fda/sf/dsa/f/dsa/f/dsa/fd/sa/fd/sa/fds/a/fdsafds</url-pattern>
    >> <!--可以编写多个url-pattern，但是路径需要以"/"开始 -->
    >> <!--注意该路径中不需要添加项目的名称-->
    >> <url-pattern>/hello</url-pattern>

</servlet-mapping>
```

第二个例子，响应一个html网页到浏览器，文件名WelcomServlet.java

```
public void service(ServletRequest request,ServletResponse response) {
    >> throws ServletException,IOException{
        |
        //将信息输出到浏览器上
        //将HTML字符串输出到浏览器上，浏览器解释执行

        //获取输出流对象，流直接指向特定的浏览器客户端
        PrintWriter out = response.getWriter();

        //响应HTML代码到浏览器
        out.print("<html>");
        out.print("<head>");
        out.print("<title>welcome · servlet</title>");
        out.print("</head>");
        out.print("<body>");
        out.print("<h1 align=\\"center\\">welcome · study · servlet!</h1>");
        out.print("</body>");
        out.print("</html>");

    }

</servlet>
    >> <servlet-name>helloServlet</servlet-name>
    >> <servlet-class>WelcomServlet</servlet-class>
</servlet>
<servlet-mapping>
    >> <servlet-name>helloServlet</servlet-name>
    >> <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

//在获取响应流之前设置有效果

```
response.setContentType("text/html;charset=UTF-8");
```

第一执行，之后执行只执行service，请求访问一次执行一次service  
servlet是单例，多线程情况下执行

```
Servlet life cycle demo Application Configuration  
HelloServlet's Constructor execute!  
HelloServlet's init method execute!  
HelloServlet's service method execute!
```

#### 关于Servlet对象的生命周期

##### 1、什么是生命周期？

生命周期表示一个java对象从最初被创建到最终被销毁，经历的所有过程。

##### 2、Servlet对象的生命周期是谁来管理的？程序员可以干涉吗？

Servlet对象的生命周期，javaweb程序员是无权干涉的，包括该Servlet对象的相关方法的调用，javaweb程序员也是无权干涉的。

I Web Container管理Servlet对象的生命周期。

##### 3、“默认情况下”，Servlet对象在WEB服务器启动阶段不会被实例化。【若希望在web服务器启动阶段实例化Servlet对象，需要进行特殊的设置】

##### 4、描述Servlet对生命周期

- 1) 用户在浏览器地址栏上输入URL: <http://localhost:8080/prj-servlet-03/testLifeCycle>
- 2) web容器截取消费路径: /prj-servlet-03/testLifeCycle
- 3) web容器在容器上下文中找请求路径 /prj-servlet-03/testLifeCycle 对应的Servlet对象
- 4) 若没有找到对应的Servlet对象
  - 4.1) 通过 web.xml 文件中相关的配置信息，得到请求路径/testLifeCycle 对应的Servlet完整类名
  - 4.2) 通过反射机制，调用Servlet类的无参数构造方法完成Servlet对象的实例化
  - 4.3) web容器调用Servlet对象的init方法完成初始化操作
  - 4.4) web容器调用Servlet对象的service方法提供服务
- 5) 若找到对应的Servlet对象
  - 5.1) web容器直接调用Servlet对象的service方法提供服务
- 6) web容器关闭时或/webapp重新部署的时候，该Servlet对象长时间没有用户再次访问的时候，web容器会将该Servlet对象销毁，在销毁该对象之前，web容器会调用Servlet对象的destroy方法，完成销毁之前的准备。

##### 6、注意：

init方法执行的时候，Servlet对象已经被创建好了。  
destroy方法执行的时候，Servlet对象还没有被销毁，即将被销毁。

##### 7、Servlet对象是单例，但是不符合单例模式，只能称为伪单例。真单例的构造方法是私有化的，Tomcat服务器是支持多线程的。 所以Servlet对象在单例多线程的环境下运行的。那么Servlet对象若有实例变量，并且实例变量涉及到修改操作，那么这个Servlet对象一定会存在线程安全问题，不建议在Servlet对象中使用实例变量，尽量使用局部变量。|

#### 3、javaweb程序员在编程的时候，一直是面向ServletConfig接口去完成调用，不需要关心具体的实现类。

webapp放到Tomcat服务器中，ServletConfig的实现类是：org.apache.catalina.core.StandardWrapperFacade  
webapp放到JBoss服务器中，ServletConfig的实现类可能是另外一个类名了。

param其实就是初始化了ServletConfig参数

```
<servlet-name>a</servlet-name>  
<servlet-class>com.bjpowernode.javaweb.servlet.AServlet</servlet-class>  
<!-- 初始化参数：被封装到ServletConfig对象中 -->  
<init-param>  
<param-name>driver</param-name>  
<param-value>com.mysql.jdbc.Driver</param-value>  
</init-param>  
<init-param>  
<param-name>url</param-name>  
<param-value>jdbc:mysql://localhost:3366/bjpowernode</param-value>  
</init-param>  
<init-param>  
<param-name>user</param-name>  
<param-value>root</param-value>  
</init-param>  
<init-param>  
<param-name>password</param-name>  
<param-value>123</param-value>  
</init-param>
```

在service方法中获取初始化时的信息，通过设置一个实例对象实现

```
@Override  
public void service(ServletRequest request, ServletResponse response)  
    throws ServletException, IOException {  
    //获取ServletConfig  
    ServletConfig config = getServletConfig();  
  
    //通过初始化参数的name获取value  
    String driver = config.getInitParameter("driver");  
    String url = config.getInitParameter("url");  
    String user = config.getInitParameter("user");  
    String password = config.getInitParameter("password");  
  
    System.out.println(driver);  
    System.out.println(url);  
    System.out.println(user);  
    System.out.println(password);  
}  
  
@Override  
public void init(ServletConfig config) throws ServletException {  
    this.config = config;  
}  
  
@Override  
public void service(ServletRequest request, ServletResponse response)  
    throws ServletException, IOException {  
  
    //获取ServletConfig  
    ServletConfig config = getServletConfig();  
  
    //获取ServletContext  
    ServletContext application = config.getServletContext();  
}
```

1、javax.servlet.ServletContext接口, Servlet规范

2、Tomcat服务器对ServletContext接口的实现类的完整类名:org.apache.catalina.core.ApplicationContextFacade  
javaweb程序员还是只需要面向ServletContext接口调用方法即可, 不需要关心Tomcat具体的实现。

3、ServletContext到底是什么? 什么时候被创建? 什么时候被销毁? 创建几个?

- ServletContext被翻译为【Servlet上下文】[Context一般都翻译为上下文]
- 一个webapp只有一个web.xml文件, web.xml文件服务器启动阶段被解析
- 一个webapp只有一个ServletContext对象, ServletContext在服务器启动阶段被实例化
- ServletContext在服务器关闭的时候会被销毁
- ServletContext对应的的是web.xml文件, 是web.xml文件的代表
- ServletContext是所有Servlet对象四周环境的代表

4、ServletContext接口中有哪些常用的方法?

5、Servlet、ServletConfig、ServletContext之间的关系?

- 一个Servlet对应一个ServletConfig, 100个Servlet对应100个ServletConfig
- 所有的Servlet共享一个ServletContext对象。

- 所有的用户若想共享同一个数据, 可以将这个数据放到ServletContext对象中。
- 一般放到ServletContext对象中的数据是不建议涉及到修改操作的, 因为ServletContext是多线程共享的一个对象, 修改的时候会存在线程安全问题。

4、ServletContext接口中有哪些常用的方法?

- Object getAttribute(String name)
- String getInitParameter(String name)
- Enumeration getInitParameterNames()
- String getRealPath(String path)
- void removeAttribute(String name)
- void setAttribute(String name, Object object)

```
//获取所有上下文初始化参数的name
Enumeration<String> names = application.getInitParameterNames();
while(names.hasMoreElements()){
    String name = names.nextElement();
    //通过上下文初始化参数的name获取value
    String value = application.getInitParameter(name);
    System.out.println(name + " = " + value);
}
```

```
<error-page>
    <error-code>404</error-code>
    <location>/error/error.html</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/error/error.html</location>
</error-page>
```

1、前端的页面发送的请求方式应当和服务器端需要的请求方式一致

- 服务器需要前端发送POST请求, 那前端就应该发送POST请求, 若发送GET请求, 服务器应当提示错误信息。
- 服务器需要前端发送GET请求, 那前端就应该发送GET请求, 若发送POST请求, 服务器应当提示错误信息。

2、怎么完成以上的需求?

- 在javaweb程序中想办法获取该请求是什么类型的请求, POST? 还是GET?
- 当我们获取到请求方式之后, 在javaweb程序中可以使用java语言中if语句进行判断

```
if("POST".equals(method)){
    ...
} else if("GET".equals(method)){
    ...
}

public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException {

    //将ServletRequest,ServletResponse强制类型转换成带有Http的接口类型
    HttpServletRequest request = (HttpServletRequest)req;
    HttpServletResponse response = (HttpServletResponse)res;

    //获取浏览器发送的请求方式
    String method = request.getMethod();
    //LoginServlet是处理登录的, 要求前边必须发送POST请求
    if("GET".equals(method)){
        //后台报出错误
        //前台报出错误
    }

    //若程序能够执行到这里证明用户发送的请求是POST请求, 程序应当正常执行
}
```

以下是HttpServlet类实现, 以后处理Get和Post请求使用HttpServlet子类。

HttpServlet里面有默认doGet和doPost方法, 默认出错。

子类如果要处理get请求需要重写doGet方法, 如果需要处理doPost请求需要重写doPost方法。

当浏览器发送的请求方式和后台的处理方式不同的话, 会出现一个错误, 代码: 405

```

@Override
public void service(ServletRequest req, ServletResponse res)
    throws ServletException, IOException {
    HttpServletRequest request = (HttpServletRequest)req;
    HttpServletResponse response = (HttpServletResponse)res;
    service(request,response);
}

public void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String method = request.getMethod();
    if("GET".equals(method)){
        doGet(request,response);
    }else if("POST".equals(method)){
        doPost(request,response);
    }
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.print("405-您应该发送POST请求");
    throw new RuntimeException("405-您应该发送POST请求");
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.print("405-您应该发送GET请求");
    throw new RuntimeException("405-您应该发送GET请求");
}

* Student类不再关心算法，只关心算法的具体步骤的实现
* 模板方法设计模式，可以做到，在不改变算法的前提下，可以重新定义算法步骤的具体实现。
*/

```

#### 4、`HttpServletRequest`这个对象中封装了哪些信息呢？

- 封装了HTTP请求协议的全部内容：
- 请求方式
  - URI
  - 协议版本号
  - 表单提交的数据

#### 5、`HttpServletRequest`一般变量的名字叫做：`request`，表示请求，`HttpServletRequest`对象

代表一次请求，一次请求对应一个`request`对象，100个请求对应100个`request`对象，所以`request`对象的生命周期是短暂的，什么是一次请求？

到目前为止，我们可以这样理解一次请求：在网页上点击超链接，到最终网页停下来，这就是一次完整的请求。

##### `HttpServletRequest`接口中常用的方法：

- 表单提交的数据，会自动被封装到`request`对象中，`request`对象中有`Map`集合存储这些数据：`Map`集合的`key`是`name`，`value`是一个字符串类型的一维数组
  - `String getParameter(String name)` //通过`key`获取`value`这个一维数组中的首元素（通常情况下这个一维数组中只有一个元素，所以这个方法使用最多）
  - `Map getParameterMap()` //获取整个`Map`集合
  - `Enumeration getParameterNames()` //获取整个`Map`集合的所有`key`
  - `String[] getParameterValues(String name)` //通过`Map`集合`key`获取`value`

```

//获取整个参数Map集合
Map<String, String[]> parameterMap = request.getParameterMap();
Set<String> names = parameterMap.keySet();
for(String name : names){
    String[] values = parameterMap.get(name);
    for(String value : values){
        System.out.print(value + " ");
    }
    System.out.println(name + " = " + values);
}

```

```

//获取上下文路径【webapp】的根路径】
String contextPath = request.getContextPath();
out.print("context path = " + contextPath + "<br>");

//获取浏览器的请求方式
String method = request.getMethod();
out.print("method = " + method + "<br>"); I

```

```
//获取请求的URI  
String uri = request.getRequestURI();  
out.print("uri = " + uri + "<br>");  
  
//获取请求的URL  
String url = request.getRequestURL().toString();  
out.print("url = " + url + "<br>");  
  
//获取Servlet Path  
String servletPath = request.getServletPath();  
out.print("servlet path = " + servletPath + "<br>");
```

以上代码对应的输出：

```
context path = /prj-servlet-13  
method = GET  
uri = /prj-servlet-13/testRequest  
url = http://localhost:8080/prj-servlet-13/testRequest  
servlet path = /testRequest
```

```
- String getServletContext() //获取上下文路径, webapp的根路径  
- String getMethod() //获取浏览器的请求方式  
- String getRequestURI() //获取URI  
- StringBuffer getRequestURL() //获取URL  
- String getServletPath() //获取Servlet PATH url-pattern  
- String getRemoteAddr() //获取客户端IP地址
```

以下两图作为对比：

两组设置Attribute的方法区别：

---- ServletContext存在于整个应用存活期间。

---- ServletRequest只存在于一次请求，想要在俩次请求中共享数据，需要使用转发技术。

ServletContext是Servlet上下文对象，该接口中也有这样几个方法：  
- void setAttribute(String name, Object o) //向ServletContext范围中添加数据  
- Object getAttribute(String name) //从ServletContext范围中读取数据  
- void removeAttribute(String name) //移除ServletContext范围中的数据

ServletContext是一个怎样的范围？  
所有用户共享的一个范围对象，我们一般把ServletContext变量命名为：application  
可见这个对象代表一个应用。一个webapp只有一个这样的对象。范围极大！

HttpServletRequest接口中常用的方法：

```
- String getParameter(String name)  
- Map<String, Object> getParameterMap()  
- Enumeration<String> getParameterNames()  
- String[] getParameterValues(String name)  
  
- String getServletContext() //获取上下文路径, webapp的根路径  
- String getMethod() //获取浏览器的请求方式  
- String getRequestURI() //获取URI  
- StringBuffer getRequestURL() //获取URL  
- String getServletPath() //获取Servlet PATH url-pattern  
- String getRemoteAddr() //获取客户端IP地址  
  
- void setAttribute(String name, Object o) //向request范围内存储数据  
- Object getAttribute(String name) //从request范围内读取数据  
- void removeAttribute(String name) //移除request范围内的数据  
  
//转发  
request.getRequestDispatcher("/b").forward(request, response);
```

To Solve the Garbled Characters:

```
//第一种解决方案：万能解决方案，post和get都可以使用  
byte[] bytes = dname.getBytes("ISO-8859-1"); //解码  
dname = new String(bytes, "UTF-8"); //编码【这里的编码方式，需要和浏览器的编码方式一致】  
System.out.println(dname);
```

第二种解决方案现在get和post都支持了，也是万能解决方案。

```
//第二种解决方案：调用request的setCharacterEncoding方法，但是这种方式只适合POST请求  
request.setCharacterEncoding("UTF-8");  
String dname = request.getParameter("dname");  
System.out.println(dname);
```

第三种解决方案：专门解决GET请求的乱码问题，因为这种方式只对请求行编码

```
修改CATALINA_HOME/conf/server.xml文件
<Connector port="80"
    protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443"
    URIEncoding="UTF-8"/>
```

Servlet线程安全问题：

1、Servlet是单实例多线程环境下运行的。

2、什么时候程序存在线程安全问题？

- 多线程并发
- 有共享的数据
- 共享数据有修改操作

3、在JVM中，哪些数据可能会存在线程安全问题？

- 局部变量内存空间不共享，一个线程一个栈，局部变量在栈中存储，局部变量不会存在线程安全问题
- 常量不会被修改，所以常量不会存在线程安全问题
- 所有线程共享一个堆
  - \* 堆内存中new出来的对象在其中存储，对象内部有“实例变量”，所以“实例变量”的内存多线程是共享的，实例变量多线程共同访问，并且涉及到修改操作的时候就会存在线程安全问题。
- 所有线程共享一个方法区
  - \* 方法区中有静态变量，静态变量的内存也是共享的，若涉及到修改操作，静态变量也存在线程安全问题。

4、线程安全问题不止是体现在JVM中，还有可能发生在数据库中，例如：多个线程共享同一张表，并且同时去修改表中一些记录，那么这些记录就存在线程安全问题，怎么解决数据库中数据的线程安全问题呢？至少有两种方案：

- 第一种方案是：在Java程序中使用synchronized关键字，线程排队执行，自然不会在数据库中并发，解决线程安全问题。
- 第二种方案是：行锁【悲观锁】
- 第三种方案是：事务隔离级别，例如：串行化
- 第四种方案是：乐观锁.....

5、怎么解决线程安全问题？

5.1 不使用实例变量，尽量使用局部变量

5.2 若必须使用实例变量，那么我们可以考虑将该对象变成多例对象，一个线程一个java对象，实例变量的内存也不会共享。

5.3 若必须使用单例，那就只能使用synchronized线程同步机制，线程一旦排队执行，则吞吐量降低，降低用户体验。

6、Servlet怎么解决线程安全问题？

6.1 不使用实例变量，尽量使用局部变量

6.2 Servlet必须是单例的，所以剩下的方式只能考虑使用synchronized，线程同步机制。

\*\*\*\*\*

转发和重定向：

1、web系统中资源跳转，怎么做？

- 需要使用转发机制(forward)和重定向机制(redirect)完成资源跳转
- 注意：跳转的下一个资源不一定是一个Servlet，下一个资源可能是：JSP、Servlet、html....

2、转发

- 代码怎么写：

```
//转发forward:一次请求
request.getRequestDispatcher("/b").forward(request, response);
```

- 转发是一次请求
- 转发是request对象触发的
- 用户点击超链接：http://localhost/prj-servlet-16/a，浏览器地址栏上最终显示的地址也是：

http://localhost/prj-servlet-16/a

- 转发的资源路径中不需要添加contextPath
- 转发只能完成项目内部资源的跳转

3、重定向

- 代码怎么写：

```
//重定向redirect:两次请求
//执行到此处之后，将/prj-servlet-16/b路径响应给浏览器，浏览器又向服务器发送了一次全新的请求
response.sendRedirect("/prj-servlet-16/b");
```

- 重定向是两次请求
- 重定向是response对象触发的

- 用户点击超链接: `http://localhost/prj-servlet-16/a`,  
浏览器地址栏上最终显示的地址是: `http://localhost/prj-servlet-16/b`
- 重定向的资源路径需要添加contextPath
- 重定向可以完成跨项目资源跳转

#### 4、转发和重定向的区别

#### 5、什么时候使用转发?什么时候使用重定向?

- 若跨项目跳转只能用重定向
- 大部分情况下, 重定向使用较多
- 在上一个程序中向request范围内存储了数据, 希望能从下一个程序中将request范围中的数据取出, 必须使用转发
- 重定向可以解决页面刷新问题(F5)

#### 6、这个描述不正确:

在浏览器上用户点击超链接, 到最终网页停下来, 是一次请求。这个描述已经不正确了。

因为这个过程中可能发生重定向操作。

以下, 数据提交成功转到成功页面, 使用转发, 刷新一次页面会导致提交一次数据。

```
if( count == 1){  
    //保存成功, “跳转”到成功页面  
    //转发  
    request.getRequestDispatcher("/success.html").forward(request, response);
```

重定向只会刷新第二个页面, 不会导致数据重复提交。

浏览器刷新的是最后一次请求。

```
//重定向  
response.sendRedirect(request.getContextPath() + "/success.html");
```

## Cookie

#### 1、Cookie是什么? Cookie作用? Cookie保存在哪里?

- Cookie可以保存会话状态, 但是这个会话状态是保留在客户端上。
- 只要Cookie清除, 或者Cookie失效, 这个会话状态就没有了。
- Cookie是保存在浏览器客户端上的
- Cookie可以保存在浏览器的缓存中, 浏览器关闭Cookie消失
- Cookie也可以保存在客户端的硬盘文件中, 浏览器关闭Cookie还在, 除非Cookie失效。

#### 2、Cookie只有在javaweb中有吗?

- Cookie不止是在javaweb中存在
- 只要是web开发, 只要是B/S架构的系统, 只要是基于HTTP协议, 就有Cookie的存在。
  - Cookie这种机制是HTTP协议规定的。

#### 3、Cookie实现的功能, 常见的有哪些?

- 保留购物车商品的状态在客户端上
- 十天内免登录
- .....

#### 4、在java中Cookie被当做类来处理, 使用new运算符可以创建Cookie对象, 而且Cookie由两部分组成, 分别是Cookie的name和value, name和value都是字符串类型String。

#### 5、在java程序中怎么创建Cookie?

```
Cookie cookie = new Cookie(String cookieName, String cookieValue);
```

#### 6、服务器可以一次向浏览器发送多个Cookie

response的方法

```
response.setContentType(type);
response.getWriter();
response.sendRedirect(location);
```

cookie由服务器创建，保存在浏览器客户端。

```
//将Cookie对象发送给浏览器客户端
response.addCookie(cookie1);
response.addCookie(cookie2);
```

7、默认情况下，服务器发送Cookie给浏览器之后，浏览器将Cookie保存在缓存当中，只要不关闭浏览器，Cookie永远存在，并且有效。

当浏览器关闭之后，缓存中的Cookie被清除。

8、在浏览器客户端无论是硬盘文件中还是缓存中保存的Cookie，什么时候会再次发送给服务器呢？

- 浏览器会不会提交发送这些Cookie给服务器，和请求路径有关系。
- 请求路径和Cookie是紧密关联的。
- 不同的请求路径会发送提交不同的Cookie

9、默认情况下Cookie会和哪些路径绑定在一起????

/prj-servlet-18/test/createAndSendCookieToBrowser 请求服务器，服务器生成Cookie，并将Cookie发送给浏览器客户端

这个浏览器中的Cookie会默认和“test/”这个路径绑定在一起。

也就是说，以后只要发送“test/”请求，Cookie一定会提交给服务器。

/prj-servlet-18/a 请求服务器，服务器生成Cookie，并将Cookie发送给浏览器客户端

这个浏览器中的Cookie会默认和“prj-servlet-18/”这个路径绑定在一起。

也就是说，以后只要发送“prj-servlet-18/”请求，Cookie一定会提交给服务器。

10、其实路径是可以指定的，可以通过java程序进行设置，保证Cookie和某个特定的路径绑定在一起。

假设，执行了这样的程序： cookie.setPath("/prj-servlet-18/king");

那么：Cookie将和"/prj-servlet-18/king"路径绑定在一起

只有发送“/prj-servlet-18/king”请求路径，浏览器才会提交Cookie给服务器。

```
//创建Cookie对象
Cookie cookie1 = new Cookie("username","zhangsan");
Cookie cookie2 = new Cookie("password","123");

//设置Cookie的关联路径
cookie1.setPath(request.getContextPath() + "/king");
cookie2.setPath(request.getContextPath() + "/king");

//将Cookie对象发送给浏览器客户端
response.addCookie(cookie1);
response.addCookie(cookie2);
```

11、默认情况下，没有设置Cookie的有效时长，该Cookie被默认保存在浏览器的缓存当中，只要浏览器不关闭Cookie存在，只要关闭浏览器Cookie消失，我们可以通过设置Cookie的有效时长，以保证Cookie保存在硬盘文件当中。但是这个有效时长必须是>0的。换句话说，只要设置Cookie的有效时长大于0，则该Cookie会被保存在客户端硬盘文件当中。有效时长过去之后，

则硬盘文件当中的Cookie失效。

cookie有效时长 = 0 直接被删除

cookie有效时长 < 0 不会被存储

cookie有效时长 > 0 存储在硬盘文件当中

cookie.setMaxAge(60 \* 60); // 1小时有效

12、浏览器提交Cookie给服务器，服务器怎么接收Cookie?

```
Cookie[] cookies = request.getCookies();
```

```

        if(cookies != null) {
            for(Cookie cookie : cookies) {
                String cookieName = cookie.getName();
                String cookieValue = cookie.getValue();
                System.out.println(cookieName + "=" + cookieValue);
            }
        }
    }

```

13、浏览器是可以禁用Cookie, 什么意思?

- 表示服务器发送过来的Cookie, 我浏览器不要, 不接收。
- 服务器还是会发送Cookie的, 只不过浏览器不再接收。

```

if(loginSuccess){
    //登录成功之后, 获取用户是否选择了十天内免登录
    String tenDayAutoLoginFlag = request.getParameter("tenDayAutoLoginFlag");
    if("ok".equals(tenDayAutoLoginFlag)){
        //创建Cookie对象
        Cookie cookie1 = new Cookie("username",username);
        Cookie cookie2 = new Cookie("password",password);
        //设置有效时间
        cookie1.setMaxAge(60 * 60 * 24 * 10);
        cookie2.setMaxAge(60 * 60 * 24 * 10);
        //设置关联路径
        cookie1.setPath(request.getContextPath());
        cookie2.setPath(request.getContextPath());
        //发送Cookie给浏览器
        response.addCookie(cookie1);
        response.addCookie(cookie2);
    }
}
*****
```

关于url-pattern的编写方式和路径的总结

1、路径的编写形式:

<a href="/项目名/资源路径"></a>  
<form action="/项目名/资源路径"></form>

- 重定向:

response.sendRedirect("/项目名/资源路径");

- 转发:

request.getRequestDispatcher("/资源路径").forward(request,response);

- 欢迎页面

<welcome-file-list>  
<welcome-file>资源路径</welcome-file>  
</welcome-file-list>  
- servlet路径  
<servlet>  
<servlet-name>hello</servlet-name>  
<servlet-class>com.bjpowernode.javaweb.servlet.HelloServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
<servlet-name>hello</servlet-name>  
<url-pattern>/资源路径</url-pattern>  
</servlet-mapping>

- Cookie设置path

cookie.setPath("/项目名/资源路径");

- ServletContext

ServletContext application = config.getServletContext();  
application.getRealPath("/WEB-INF/classes/db.properties");  
application.getRealPath("/资源路径");

## 2、url-pattern的编写方式

2.1 url-pattern可以编写多个

2.2 精确匹配

<url-pattern>/hello</url-pattern>

<url-pattern>/system/hello</url-pattern>

2.3 扩展匹配

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

2.4 后缀匹配

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

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

2.5 全部匹配

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

\*\*\*\*\*

关于web编程中的Session:

1、Session表示会话，不止是在javaweb中存在，只要是web开发，都有会话这种机制。

2、在java中会话对应的类型是： javax.servlet.http.HttpSession，简称session/会话

3、Cookie可以将会话状态保存在客户端， HttpSession可以将会话状态保存在服务器端。

4、HttpSession对象是一个会话级别的对象，一次会话对应一个HttpSession对象。

5、什么是一次会话？

“目前”可以这样理解：用户打开浏览器，在浏览器上发送多次请求，直到最终关闭浏览器，表示一次完整的会话。

6、在会话进行过程中， web服务器一直为当前这个用户维护着一个会话对象/HttpSession

7、在WEB容器中， WEB容器维护了大量的HttpSession对象，换句话说，在WEB容器中应该有一个“session列表”，

思考：为什么当前会话中的每一次请求可以获取到属于自己的会话对象？ session的实现原理？

- 打开浏览器，在浏览器上发送首次请求
- 服务器会创建一个HttpSession对象，该对象代表一次会话
- 同时生成HttpSession对象对应的Cookie对象，并且Cookie对象的name是JSESSIONID，Cookie的value是32位长度的字符串
- 服务器将Cookie的value和HttpSession对象绑定到session列表中
- 服务器将Cookie完整发送给浏览器客户端
- 浏览器客户端将Cookie保存到缓存中
- 只要浏览器不关闭，Cookie不会消失
- 当再次发送请求的时候，会自动提交缓存当中的Cookie
- 服务器接收到Cookie，验证该Cookie的name确实是： JSESSIONID，然后获取该Cookie的value
- 通过Cookie的value去session列表中检索对应的HttpSession对象。

8、和HttpSession对象关联的这个Cookie的name是比较特殊的，在java中就叫做： jsessionid

9、浏览器禁用Cookie会出现什么问题？怎么解决？

- 浏览器禁用Cookie，则浏览器缓存中不再保存Cookie

- 导致在同一个会话中，无法获取到对应的会话对象

- 禁用Cookie之后，每一次获取的会话对象都是新的

浏览器禁用Cookie之后，若还想拿到对应的Session对象，必须使用URL重写机制，怎么重写URL：

<http://localhost/prj-servlet-21/user/accessMySelfSession;jsessionid=D3E9985BC5FD4BD05018BF2966863E94>

重写URL会给编程带来难度/复杂度，所以一般的web站点是不建议禁用Cookie的。建议浏览器开启Cookie

10、浏览器关闭之后，服务器端对应的session对象会被销毁吗？为什么？

- 浏览器关闭之后，服务器不会销毁session对象

- 因为B/S架构的系统基于HTTP协议，而HTTP协议是一种无连接/无状态的协议
- 什么是无连接/无状态？
  - \* 请求的瞬间浏览器和服务器之间的通道是打开的，请求响应结束之后，通道关闭
  - \* 这样做的目的是降低服务器的压力。

11、session对象在什么时候被销毁？

- web系统中引入了session超时的概念。
- 当很长一段时间（这个时间可以配置）没有用户再访问该session对象，此时session对象超时，web服务器自动回收session对象。

– 可配置：web.xml文件中，默认是30分钟

```
<session-config>
<session-timeout>120</session-timeout>
</session-config>
```

12、什么是一次会话session呢？

- 一般多数情况下，是这样描述的：用户打开浏览器，在浏览器上进行一些操作，然后将浏览器关闭，表示一次会话结束。
- 本质上的描述：从session对象的创建，到最终session对象超时之后销毁，这个才是真正意义的一次完整会话。

13、关于javax.servlet.http.HttpSession接口中常用方法：

```
void setAttribute(String name, Object value)
Object getAttribute(String name)
void removeAttribute(String name)
void invalidate() 销毁session
```

14、ServletContext、HttpSession、HttpServletRequest接口的对比：

14.1 以上都是范围对象：

```
ServletContext application; 是应用范围
HttpSession session; 是会话范围
HttpServletRequest request; 是请求范围
```

14.2 三个范围的排序：

```
application > session > request
```

14.3

application完成跨会话共享数据、  
 session完成跨请求共享数据，但是这些请求必须在同一个会话当中、  
 request完成跨Servlet共享数据，但是这些Servlet必须在同一个请求当中【转发】

14.4 使用原则：有小到大尝试，优先使用小范围。

例如：登录成功之后，已经登录的状态需要保存起来，可以将登录成功的这个状态保存到session对象中。

登录成功状态不能保存到request范围中，因为一次请求对应一个新的request对象。

登录成功状态也不能保存到application范围中，因为登录成功状态是属于会话级别的，不能所有用户共享。

15、补充HttpServletRequest中的方法：

```
HttpSession session = request.getSession(); 获取当前的session，获取不到，则新建session
HttpSession session = request.getSession(true); 获取当前的session，获取不到，则新建session
HttpSession session = request.getSession(false); 获取当前的session，获取不到，则返回null
```

**登录成功保存会话**

```
if(rs.next()){
    //登录成功，将用户信息包装到实体对象中
    user = new User();
    user.setId(rs.getInt("id"));
    user.setUsername(rs.getString("username"));
    user.setPassword(rs.getString("password"));
}
if(user != null){
    //登录成功，将登录成功的状态保存，跳转到index.html页面
    //获取session对象
    HttpSession session = request.getSession();
    //将登录成功的用户存储到会话范围
    session.setAttribute("user", user);
    //重定向
    response.sendRedirect(request.getContextPath() + "/index.html");
} else{
    //登录失败跳转到失败页面
    response.sendRedirect(request.getContextPath() + "/login-error.html");
}
```