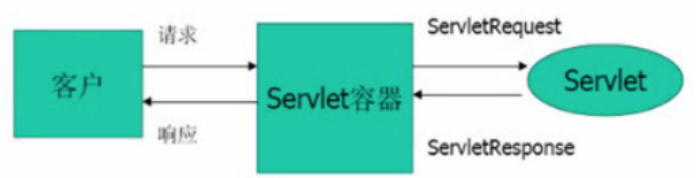
第2章 Servlet

2016年5月12日 15:51

第一部分

1. java Servlet是和平台无关的服务端组件，运行在Servlet容器中，负责Servlet和客户通讯以及调用Servlet的方法
   Servlet和客户端通讯采用“请求/响应”模式
2. Servlet完成的功能 ：
   a. 创建并返回基于客户请求的动态HTML页面
   b. 创建可嵌入到现有HTML的部分HTML片段
   c. 与其他服务器资源(数据库/基于java的应用程序…)进行通讯
3. Servlet容器相应客户请求过程 ：



4. 创建Servlet步骤
   a. 创建Servlet接口的实现类

```
public class TestServlet implements Servlet {}
```

   b. 在web.xml中配置和映射Servlet

```
<!-- 配置和映射Servlet -->
<servlet>
  <!-- servlet注册名称 -->
    <servlet-name>TestServlet</servlet-name>
    <!-- servlet的全类名 -->
    <servlet-class>org.yang.demo.TestServlet</servlet-class>
</servlet>
<servlet-mapping>
  <!-- 需要和servlet子节点servlet-name值保持相同 -->
    <servlet-name>TestServlet</servlet-name>
    <!-- 映射具体访问路径:/代表当前web应用的根目录 -->
    <url-pattern>/test</url-pattern>
</servlet-mapping>
```

5. Servlet容器
   a. 可以创建Servlet，并调用Servlet的相关生命周期方法
   b. JSP， Filter， Listener， Tag … 都运行在Servlet容器中
6. Servlet生命周期 (所有方法都由Servlet容器负责调用)
   a. 构造器 ： 只被调用一次，第一次请求Servlet时，创建Servlet的实例，调用构造器，Servlet是单例模式
   b. init ： 只被调用一次，在创建好实例后立即调用，用于初始化当前Servlet
   c. service ： 被多次调用，每次请求时都会调用service方法，实际用来相应请求的
   d. destory ： 只被调用一次，在当前Servlet所在的web应用被卸载时调用，用于释放所占用资源
7. 带注解的生命周期
   a. @PostConstruct
      使用@PostConstruct修饰无返回值且没有抛出异常声明的方法，在Servlet构造器执行之后init()方法运行之前执行，只执行一次
   b. @PreDestroy
      使用@PreDestroy修饰无返回值且没有抛出异常声明的方法，在Servlet中destroy()方法运行之后销毁之前执行，只执行一次
   c. 完整的生命周期
      服务器加载Servlet -> construct() -> @PostConstruct修饰的方法 -> init(ServletConfig conf) ->
      service(HttpServletResquest request, HttpServletResponse response) -> destory() -> @PreDestroy修饰的方法 ->
      服务器卸载Servlet
8. load-on-startup

```
<servlet>
  <!-- servlet注册名称 -->
    <servlet-name>TestServlet</servlet-name>
    <!-- servlet的全类名 -->
    <servlet-class>org.yang.demo.TestServlet</servlet-class>
    <!-- 指定Servlet被创建的时机 -->
    <load-on-startup>1</load-on-startup>
</servlet>
```

   负数 --->>> 在第一次请求时被创建
   0或正数 --->>> 当前web应用被Servlet容器加载时创建实例，数字越小创建越早
9. servlet-mapping
   a. 一个Servlet可以映射多个servlet-mapping
   b. 通配符的写法

```
<url-pattern>/*</url-pattern>
<url-pattern>*.html</url-pattern>
```

10. ServletConfig ： 封装了Servlet配置信息，并可以获取ServletContext对象
    a. 配置Servlet初始化参数

```
<!-- 配置和映射Servlet -->
<servlet>
  <!-- servlet注册名称 -->
    <servlet-name>TestServlet</servlet-name>
    <!-- servlet的全类名 -->
```

```
        <servlet-class>org.yang.demo.TestServlet</servlet-class>
        <!-- 配置Servlet初始化参数 -->
        <!-- init-param必须放在load-on-startup前面 -->
        <init-param>
            <!-- 参数名 -->
            <param-name>user</param-name>
            <!-- 参数值 -->
            <param-value>root</param-value>
        </init-param>
        <init-param>
            <param-name>password</param-name>
            <param-value>1234</param-value>
        </init-param>
              <!-- 指定Servlet被创建的时机 -->
        <load-on-startup>1</load-on-startup>
    </servlet>
```

b. 获取初始化参数
   - servletConfig.getInitParameter("key"); //通过键名获取对应键值
   - servletConfig.getInitParameterNames(); //返回键名组成的Enumeration

```
@Override
public void init(ServletConfig arg0) throws ServletException {
    // TODO Auto-generated method stub
    String user = arg0.getInitParameter("username");
    String pass = arg0.getInitParameter("password");
    System.out.println("1^ username is " + user);
    System.out.println("1^ password is " + pass);
    Enumeration<String> users = arg0.getInitParameterNames();
    while (users.hasMoreElements()) {
        String name = users.nextElement();
        System.out.println("2^ " + name + " is " + arg0.getInitParameter(name));
    }
}
```

c. 获取Servlet配置名称
   - servletConfig.getServletName(); //获取的值为<servlet-name>value</servlet-name>

11. ServletContext
    a. ServletContext对象有ServletConfig对象获得
       - ServletContext context = servletConfig.getServletContext();
    b. ServletContext是web应用的大管家，可以获得当前web应用的各个方面的信息
    c. 常用操作 ：
       i. 获取当前web应用的初始化参数
          1. 配置web应用初始化参数

```
<!-- 配置当前web应用初始化参数 -->
<context-param>
    <param-name>package</param-name>
    <param-value>org.yang.demo</param-value>
</context-param>
```

          2. 获取web应用参数
             - servletContext.getInitParameter("key"); //通过键名获取对应键值
             - servletContext.getInitParameterNames(); //返回键名组成的Enumeration
             （使用方式和servletConfig完全相同）
       ii. 获取当前web应用(webContent)的某一文件相对于服务器上的绝对路径
           - context.getRealPath(String path);
       iii. 获取当前web应用的名称
           - context.getContextPath();
       iv. 获取web应用某一文件的输入流
           - context.getResourceAsStream(String path); //path为相对于当前web应用的路径
       v. 和attribute相关的方法
12. 资源注射
    a. 在web.xml中配置资源

```
<env-enrty>
    <env-enrty-name>username</env-enrty-name>
    <env-enrty-value>yang</env-enrty-value>
</env-entry>
```

    b. 在Servlet中使用资源

```
@Resource(name="username");
private String user;
```

    c. 使用JNDI获取资源

```
Context context = new InitialContext();
String user = (String) context.lookup("username");
```

第二部分

12. GET&POST请求
    a. 使用GET方式传递参数
       i. 在浏览器输入url地址或单击超链接时，浏览器发出HTTP请求消息的请求方式为GET
       ii. 使用<form>指定method为"get"，则请求方式为GET
       iii. 使用GET请求方式传递参数的格式为：localhost:8989/test/index.jsp?search=123
       iv. 使用GET请求传送的数据量在1k以下
    b. 使用POST方式传递参数
       i. POST主要用于向服务器提交表单数据，<form>制定method为"post"

ii. POST方式将表单各字段元素作为HTTP消息的实体内容发送给WEB服务器，传送数据量比GET大得多

13. 在Servlet中获取请求参数
    a. Servlet中的service()方法用于应答请求，每次请求都会调用servlet方法

```
public void service(ServletRequest arg0, ServletResponse arg1) throws ServletException, IOException {
    //ServletRequest : 封装了请求信息，可以从中取到任何请求信息
    //ServletResponse : 封装了响应信息，如果想给用户相应，则使用该接口方法的实现
    //这两个接口的实现类都是由服务器得以实现的，并在服务器调用service方式时传入
}
```

    b. ServletRequest :
    - **String getParameter(String name); //根据请求参数名称，返回参数值**
    - **String[] getParameterValues(String name); //根据请求参数参数名称，返回对应字符串数组**
    - **Enumeration getParameterNames(); //返回参数名对应的Enumeration对象**

```
e.g.
Enumeration<String> names = arg0.getParameterNames();
while (names.hasMoreElements()) {
    String name = names.nextElement();
    String val = arg0.getParameter(name);
    System.out.println(name + " : " + val);
}
```

    - **Map getParameterMap(); //返回请求参数的键值对，key=>参数名，value=>参数值(String [])**

```
e.g.
String[] username = arg0.getParameterMap().get("username");
String[] password = arg0.getParameterMap().get("password");
System.out.println("username is " + username[0]);
System.out.println("password is " + password[0]);
```

    c. HttpServletRequest ： ServletRequest的子接口，针对HTTP请求所定义，里面包含的大量的HTTP相关方法

```
HttpServletRequest httpServletRequest = (HttpServletRequest) arg0;
```

    - String requestURI = httpServletRequest.getRequestURI(); //获取请求的URI
    - String requestMethod = httpServletRequest.getMethod(); //获取请求的方式
    - String queryStr = httpServletRequest.getQueryString(); //获取get请求对应字符串
    - String servletPath = httpServletRequest.getServletPath(); //获取请求servlet映射路径

    d. ServletResponse
    - PrintWriter printWriter = arg1.getWriter();
      printWriter.println("hello world");
      //返回PrintWriter对象，通过print方法将文本输出在浏览器上
    - arg1.setContentType("text/html"); //设置文档类型属性
    - void sendRedirect(String location); //请求重定向(此方法在HttpServletResponse中定义)
      （HttpServletResponse response = (HttpServletResponse) arg1;）

14. 自定义Servlet接口实现类 & GenericServlet
    a. 自定义Servlet接口实现类

```
public abstract class MyGenericServlet implements Servlet, ServletConfig{
    private ServletConfig servletConfig = null;
    public void destroy() {}
    public ServletConfig getServletConfig() {
        return this.servletConfig;
    }
    public String getServletInfo() {
        return null;
    }
    public void init(ServletConfig arg0) throws ServletException {
        this.servletConfig = arg0;
        init();
    }
    //避免原始init(ServletConfig arg0)被覆盖无法传入ServletConfig导致空指针异常
    public void init() {}
    public abstract void service(ServletRequest arg0, ServletResponse arg1) throws ServletException,
    IOException;
    //以下为ServletConfig接口实现
    public String getInitParameter(String arg0) {
        return servletConfig.getInitParameter(arg0);
    }
    public Enumeration<String> getInitParameterNames() {
        return servletConfig.getInitParameterNames();
    }
    public ServletContext getServletContext() {
        return servletConfig.getServletContext();
    }
    public String getServletName() {
        return servletConfig.getServletName();
    }
}
```

    b. GenericServlet
    i. GenericServlet是一个Servlet，是一个实现Servlet和ServletConfig接口的抽象类，其中service为抽象方法
    ii. 使用GenericServlet可以使Servlet开发更加简洁
    iii. 具体表现
        1. 在GenericServlet中生命ServletConfig类型的成员变量，并在init(ServletConfig arg0)中初始化
        2. 定义一个init()方法，在init(ServletConfig arg0)中进行调用，子类可以直接覆盖init()实现对Servlet的初始化

3. 不建议直接覆盖init(ServletConfig arg0)，如果忘记编写super(ServletConfig)并使用了ServletConfig相关的方法则会出现空指针异常
4. 新建的无参数init()方法并非Servlet生命周期方法，init(ServletConfig arg0)是Servlet生命周期方法

15. 自定义HttpServlet类

```
public class MyHttpServlet extends MyGenericServlet {
    @Override
    public void service(ServletRequest arg0, ServletResponse arg1) throws ServletException, IOException {
        if (arg0 instanceof HttpServletRequest) {
            HttpServletRequest request = (HttpServletRequest) arg0;
            if (arg1 instanceof HttpServletResponse) {
                HttpServletResponse response = (HttpServletResponse) arg1;
                service(request, response);
            }
        }
    }
    public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        //1.获取请求方式
        String method = request.getMethod();
        //2.根据请求方式再创建对应的处理方法
        if ("GET".equals(method)) {
            doGET(request, response);
        }
        if ("POST".equals(method)) {
            doPOST(request, response);
        }
    }
    public void doPOST(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {}
    public void doGET(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {}
}
```

16. HttpServlet
    a. 是一个Servlet，继承于GenericServlet，针对于HTTP协议所定制
    b. 在service()方法中直接把ServletRequest和ServletResponse转为HttpServletRequest和HttpServletResponse
    c. 调用重载的service方法，
    `public void service(HttpServletRequest request, HttpServletResponse response)`
    根据请求方式创建了doXXX()方法，xxx为具体响应方式(doGet/doPost…)
    d. 实际开发中，直接继承HttpServlet，并根据响应请求复写doXXX方法
    e. 优点 ： 直接使用HttpServletRequest和HttpServletResponse，不需要进行转换

	Servlet部分实验
2.1	**通过配置文件验证用户登陆信息**
要求	• 在web.xml中配置正确的用户登录信息(username/password) • 定义一个login.html，里边定义username和password请求字段，发送请求道loginServlet • 创建LoginServlet，在其中获取请求的username和password，并于web.xml中的正确信息进行比对 • 若用户账号密码匹配成功则在浏览器输出"Success : (username)"，失败则在浏览器输出"Error : (username)"
时间	2016/05/17

解答

login.html

```
<form action="loginServlet" method="post">
    username : <input type="text" name="username" />
    password : <input type="password" name="password" />
    <input type="submit" value="submit" />
</form>
```

---

web.xml

```
  <context-param>
      <param-name>username</param-name>
      <param-value>yang</param-value>
  </context-param>
  <context-param>
      <param-name>password</param-name>
      <param-value>root</param-value>
  </context-param>

  <servlet>
      <servlet-name>loginServlet</servlet-name>
      <servlet-class>org.yang.demo.LoginServlet</servlet-
class>
  </servlet>
  <servlet-mapping>
      <servlet-name>loginServlet</servlet-name>
      <url-pattern>/loginServlet</url-pattern>
  </servlet-mapping>
```

LoginServlet.java

```
public class LoginServlet implements Servlet {

    private ServletContext context = null;

    @Override
    public void destroy() {
    }

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

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

    @Override
    public void init(ServletConfig arg0) throws
    ServletException {
        this.context = arg0.getServletContext();
    }

    @Override
    public void service(ServletRequest arg0,
    ServletResponse arg1) throws ServletException,
    IOException {
        String succUsername =
        context.getInitParameter("username");
        String succPassword =
        context.getInitParameter("password");
        String getUsername =
        arg0.getParameter("username");
        String getPassword =
        arg0.getParameter("password");
        PrintWriter writer = arg1.getWriter();
    if (getUsername.equals(succUsername) &&
    getPassword.equals(succPassword)) {
        writer.println("Success : " + getUsername);
    } else {
        writer.println("Error : " + getUsername);
        }
    }

}
```

2.2	自定义Servlet接口实现类用于简化开发Servlet
要求	• 新建文件MyGenericServlet.java，实现Servlet与ServletConfig接口

- 使用编写好的自定义Servlet简化开发习题2.1中的LoginServlet.java
- 在LoginServlet.java中测试初始化方法，保证程序运行正常没有空指针异常的发生

时间 2016/5/17

解答

MyGenericServlet.java

```
public abstract class MyGenericServlet implements Servlet,
ServletConfig {

    private ServletConfig config;

    public void destroy() {}
    public ServletConfig getServletConfig() {
        return this.config;
    }
    public String getServletInfo() {
        return null;
    }
    public void init(ServletConfig arg0) throws
    ServletException {
        this.config = arg0;
        init();
    }
    public void init() {}
    public abstract void service(ServletRequest arg0,
    ServletResponse arg1) throws ServletException,
    IOException;

    public String getInitParameter(String arg0) {
        return config.getInitParameter(arg0);
    }
    public Enumeration<String> getInitParameterNames() {
        return config.getInitParameterNames();
    }
    public ServletContext getServletContext() {
        return config.getServletContext();
    }
    public String getServletName() {
        return config.getServletName();
    }
}
-------------------------------------------------------------
---
```

LoginServlet.java

```
public class LoginServlet extends MyGenericServlet {
    @Override
    public void init() {
        System.out.println("this is init test");
    }
    @Override
    public void service(ServletRequest arg0,
    ServletResponse arg1) throws ServletException,
    IOException {
        String succUsername =
        getServletContext().getInitParameter("username");
        String succPassword =
        getServletContext().getInitParameter("password");
        String getUsername =
        arg0.getParameter("username");
        String getPassword =
        arg0.getParameter("password");
        PrintWriter writer = arg1.getWriter();
        if (getUsername.equals(succUsername) &&
        getPassword.equals(succPassword)) {
            writer.println("Success : " + getUsername);
        } else {
            writer.println("Error : " + getUsername);
        }
    }

}
```

2.3 自定义HttpServlet类用于简化开发Servlet

要求

- 创建MyHttpServlet.java继承于习题2.2中的MyGenericServlet.java，编写HttpServlet
- 使用编写好的MyHttpServlet方法简化习题2.1中的LoginServlet.java
- 在浏览器上输出用户的请求方式

时间 2016/5/18

解答

MyHttpServlet.java

```
public class MyHttpServlet extends MyGenericServlet {
    @Override
```

```
    public void service(ServletRequest arg0,
    ServletResponse arg1) throws ServletException,
    IOException {
        // TODO Auto-generated method stub
        if (arg0 instanceof ServletRequest) {
            HttpServletRequest request =
            (HttpServletRequest) arg0;
            if (arg1 instanceof ServletResponse) {
                HttpServletResponse response =
                (HttpServletResponse) arg1;
                service(request, response);
            }
        }
    }
public void service(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
    String method = request.getMethod();
    if ("GET".equals(method)) {
        doGet(request, response);
    }
    if ("POST".equals(method)) {
        doPost(request, response);
        }
    }
    public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {}
    public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {}
}
------------------------------------------------------------
---
LoginServlet.java
public class LoginServlet extends MyHttpServlet {
    @Override
    public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
        String succUsername =
        getServletContext().getInitParameter("username");
        String succPassword =
        getServletContext().getInitParameter("password");
        //获取请求方式
        String method = request.getMethod();
        //获取请求值
        String getUsername =
        request.getParameter("username");
        String getPassword =
        request.getParameter("password");
        PrintWriter printWriter = response.getWriter();
        if (getUsername.equals(succUsername) &&
        getPassword.equals(succPassword)) {
            printWriter.println("Success : " +
            getUsername);
        } else {
            printWriter.println("Error : " +
            getUsername);
        }
        printWriter.println("method is : " + method);
    }
}
```