Java EE 企业应用系统设计

ServletContext 和 Web 配置

王晓东

wxd2870@163.com

中国海洋大学

June 4, 2013

参考书目

1. 吕海东，张坤编著，Java EE 企业级应用开发实例教程，清华大学出版社，2010 年 8 月

大纲

接下来…

Web 应用环境对象

Web 应用环境对象

Java EE Web 的配置

Servlet 配置对象 ServletConfig

转发

概述

Java EE Web 应用需要部署在符合 Java EE 规范的 Web 容器中运行，如何取得 Web 应用本身的信息在 Web 应用编程中具有非常重要的意义。

❖ 核心内容

- Web 应用对象 ServletContext；
- Web 应用详细配置；
- MVC 模式 Web 开发中发挥核心作用的转发，区别**转发**与**重定向**。

接下来…

Web 应用环境对象

Web 应用环境对象

Java EE Web 的配置

Servlet 配置对象 ServletConfig

转发

Web 应用环境对象

将 Web 应用部署到服务器上，启动 Web 服务器后，Web 容器为每个 Web 应用创建一个表达 Web 应用环境的对象，即 ServletContext 对象，并将 Web 应用的基本信息存储在这个 ServletContext 对象中。

作用：

- 所有 Web 组件 JSP 和 Servlet 都可以访问此 ServletContext 对象，进而取得 Web 应用的基本信息。
- 此 ServletContext 还可以作为整个 Web 应用的共享容器对象，可以被所有会话请求共用，保存 Web 应用的共享信息。

Web 应用环境对象的类型和取得

JavaWeb 的环境对象是接口 javax.servlet.ServletContext 的对象。该接口的实现类由 Web 容器厂家负责实现，作为开发人员不需要考虑其实现类，只需要取得实现此接口的对象即可。
在 Servlet 方法内可以直接取得 ServletContext 接口对象：

```
1 ServletContext ctx = this.getServletContext();
```

然后可以使用 ServletContext 接口中提供的方法，取得 Web 应用的信息和数据，如容器版本、名称、端口和绝对路径。

服务器环境对象的生命周期

服务器环境对象的生命周期与 Web 应用相同，当 Web 应用启动后，它被 Web 容器创建，当 Web 应用停止时，它被 Web 容器销毁。

1. 创建：当 Web 容器启动后，自动创建 ServletContext 对象。
2. 销毁：当 Web 容器停止后，自动销毁 ServletContext 对象。

注意：如果在 ServletContext 对象中保存的对象信息需要长久保存，一般编写 ServletContext 对象的监听器类，在此对象销毁之前将其中保存的对象数据进行持久化处理，例如保存到数据库或者文件中。当 Web 服务重新启动后，将这些信息从数据库或文件中读入，并存入 ServletContext 对象中，得以在 Web 中继续使用。

服务器环境对象的功能和方法

主要功能：

- 取得 Web 容器的基本信息。
- 取得 Web 容器的环境信息。
- 保存/取得 Web 范围的数据。
- 编写 Web 容器的日志。

服务器环境对象的功能和方法

❖ Web 级数据共享容器

public void setAttribute(String name, Object object)

对象保存到 ServletContext，要求核对对象类型，支持自动装箱和拆箱。

```
1 ServletContext ctx = this.getServletContext();
2 ctx.setAttribute(userId",␣"Kevin");
3 ctx.setAttribute(age", 20); //自动完成 int 类型转换为Integer 对象类型
```

public Object getAttribute(String name)

读取保存在 ServletContext 对象中指定名称的属性对象，不存在则返回 null。

```
1 String useId = (String) ctx.getAttribute("userId");
2 int age = (Integer) ctx.getAttribute("age"); //自动拆箱，将 Integer 转为int
```

服务器环境对象的功能和方法

`public void removeAttribute(String name)`

将指定的属性从 ServletContext 对象中删除。

`Enumeration getAttributeNames()`

取得所有属性的名称列表，返回一个枚举器对象，可以用于遍历所有属性名称。

```
1 Enumeration nums = ctx.getAttributeName();
2 while (nums.hasMoreElements()) {
3   System.out.println(nums.nextElements());
4 }
```

服务器环境对象的功能和方法

❖ 读取 Web 级初始化参数

一般不要在代码中放置各种外部资源的连接参数，如数据库驱动、连接 URL 等，否则会导致参数更改时需要重新编译 Java 代码。一般的做法是将这些参数放置在 Web 配置文件/WEB-INF/web.xml 中，提高了系统的可维护性。（后续讲解）

❖ 访问外部资源

ServletContext 对象提供了访问外部资源的方法：

- 取得 Web 文档的绝对路径
- 配合 I/O 流读写 Web 文档
- 取得转发对象
- 实现服务器端 Web 组件的转发

服务器环境对象的功能和方法

public String getRealPath(String path)

取得指定 Web 目录或文档的绝对目录地址，Path 要求以“/”开头，表示 Web 根目录。如，取得 Web 目录/upload 的绝对目录地址，当使用文件上传功能组件时此方法有用。

```
1 String realPath = ctx.getRealPath("/upload");
```

public InputStream getResourceAsStream(String path)

以二进制字节流的类型返回指定的 Web 资源，可以是 Web 应用中的任何文档，包括 JSP、图片、声音或视频文件，然后使用 Input 流读取此文件。

```
1 InputStream in = ctx.getResourceAsStream("/test.txt");
2
3 int read = in.read();
4 while (read != -1) {
5   System.out.println((char)read);
6   read = in.read();
7 }
```

服务器环境对象的功能和方法

public RequestDispatcher getRequestDispatcher(String path)

取得 Web 文档的转发对象，目的是实现到目标文档的服务器转发。

```
try {
  Thread.sleep(2000);
} catch (InterruptedException e) {
  e.printStackTrace();
}
RequestDispatcher rd = ctx.getRequestDispatcher("/test.html");
rd.forward(request, response);
```

注意：要求转发目标地址必须是以“/”开头，表示 Web 的根目录，否则抛出 java.lang.IllegalArgumentException。

服务器环境对象的功能和方法

public URL getResource(String path) throws MalformedURLException

返回指定 Web 资源的 URL 地址，例如取得 Web 页面/main.jsp 的 URL：

```
1 java.net.URL url = ctx.getResource("/test.html");
2 System.out.println(url.toString());
```

输出结果为：

```
jndi:/localhost/TestWebProj/test.html
```

public String getMimeType(String file)

```
1 String mime = ctx.getMimeType("/test.html");
2 System.out.println(mime);
```

输出结果为：

```
text/html
```

服务器环境对象的功能和方法

❖ 取得 Web 应用的基本信息

```
public int getMajorVersion()
```

取得 Servlet 容器的 API 主版本号。

```
public int getMinorVersion()
```

取得 Servlet 容器的 API 次版本号。上述两个方法用于测试代码的兼容性。

```
public String getServerInfo()
```

取得 Web 容器的名称和版本信息，即 Web 服务器的名称和版本。例如输出：Apache Tomcat/7.0.32

服务器环境对象的功能和方法

❖ Web 应用日志输出

项目开发人员为追踪代码的运行情况，尤其是出现异常时的错误信息，经常将此类信息写入日志文件，便于日后监控和维护。
尤其时已经发布运行的项目，不方便到用户现场服务器的控制台进行查看。
一般的做法是配置 Web 服务器的日志文件到可以远程访问的 FTP 服务器上，开发人员可以定期从 FTP 下载日志文件进行分析，找出系统的异常和错误时间和地点。

```
public void log(String msg)
```

将指定的消息文本写入到日志文件中。一般用于比较关键的事件，如用户登录应用系统和执行关键的操作像删除产品等。

服务器环境对象的功能和方法

```
1  String id = request.getParameter("userId");
2  String password = request.getParameter("password");
3
4  IUser user = BusinessFactory.createUser();
5
6  if (user.validate(id, password)) {
7    String ip = request.getRemoteAddr();
8    ServletContext ctx = this.getServletContext();
9    String msg = "User␣Login:␣" + id + new Date() + "Time,␣At" + ip;
10   ctx.log(msg);
11 }
```

服务器环境对象的功能和方法

public void log(String message, Throwable throwable)

将异常类的跟踪堆栈并附加消息文本写入到 LOG 日志文件中，一般用于异常处理。

```
1  try {
2
3  } catch(Exception e) {
4    ctx.log("更新库存余额时错误异常", e);
5  }
```

接下来…

配置文件和位置

Web 的配置文件为/WEB-INF/web.xml，/WEB-INF 目录时被 Web 服务器保护的目录，客户端浏览器无法访问该目录下的任何文件。
其他框架，如 Struts、Spring 等都将配置文件保存在该目录下。
web.xml 配置的主要项目包括：

- Web 级初始参数（context-param）
- 过滤器（filter）
- 过滤器映射（filter-mapping）
- 监听器（listener）
- Servlet 声明（Servlet）
- Servlet 映射（Servlet-mapping）
- 异常跳转页面（error-page）
- MIME 类型映射（mime-mapping）
- 会话对象超时（session-config）
- 外部资源声明（resource-ref）
- 外部标记库描述符文件（taglib）

Web 初始参数配置

❖ Web 初始参数配置语法

```
<context-param>
  <description>数据库驱动</description>
  <param-name>driverName</param-name>
  <param-value>sun.jdbc.odbc.jdbcOdbcDriver</param-value>
</context-param>
```

Web 初始参数配置

❖ Web 组件取得 Web 初始参数

在 Servlet 中可以通过 ServletContext 对象取得 Web 初始参数，为此 ServletContext 提供如下方法：

`public String getInitParameter(String name)` 取得指定名称的 Web 初始参数。

```
1 ServletContext ctx = this.getServletContext();
2 String driverName = ctx.getInitParameter("driverName");
```

`public Enumeration getInitParameterNames()` 取得所有 Web 初始参数名称列表，以枚举器类型返回。

```
1 for (Enumeration ee = ctx.getInitParameterNames(); ee.hasMoreElements(); ) {
2   String paramName = (String) ee.nextElement();
3   System.out.println(paramName + "=" + ctx.getInitParameter(paramName));
4 }
```

Web 应用级异常处理配置

通过配置方式处理异常，当 Web 应用中 Web 组件 JSP 或 Servlet 抛出异常时，Web 容器自动在配置文件中查找对应的异常类型，根据配置自动跳转到异常处理页面。

Java EE 提供两种错误配置方法：

- 以错误状态码配置的处理方式。
- 以异常类型配置的处理方式。

Web 应用级异常处理配置

❖ 以错误状态码配置的处理方式

当 JSP 或 Servlet 的响应状态码与配置的状态码一致时，Web 容器自动跳转到配置的页面。

```
<error-page>
  <error-code>500</error-code>
  <location>/error/info500.jsp</location>
</error-page>
```

如在 Servlet 中编写如下代码：

```
out.print(10/0); // 会抛出零除异常
```

Servlet 响应为 500，表示内部错误，Web 容器将使用配置的/error/info500.jsp 页面代替默认的 500 错误页面。

Web 应用级异常处理配置

❖ 以异常类型配置的处理方式

```
1 <error-page>
2   <exception-type>java.lang.NullException</exception-type>
3   <location>/error/info500.jsp</location>
4 </error-page>
```

当 JSP 或 Servlet 内出现空指针异常时，Web 容器将检测到异常，自动转发到所配置的 location 元素指定的页面。

MIME 类型映射配置

有的文件没有出现在 MIME 中，需要开发人员手动进行文件和 MIME 类型的映射，当 Web 服务器取得此类文件的扩展名时，使用 getContextType 就可以取出对应的 MIME 类型。
映射语法如下：

```
<mime-mapping>
  <extension>jpg</extension>
  <mime-type>image/jpeg</mime-type>
</mime-mapping>
```

取得文件 images/tu01.jpg 的 MIME 类型：

```
ServletContext ctx = this.getServletContext();
String mime = ctx.getMimeType("image/tu01.jpg");
```

如果输出 MIME 类型，将显示 image/jpeg。

会话超时配置

HttpSession 对象的超时时间可以通过如下代码实现：

```
1 HttpSession session = request.getSession();
2 session.setMaxInactiveInterval(15 * 60); // 设置会话超时为分钟15
```

Java EE 规范提供了在 Web 配置文件中进行会话超时配置的功能：

```
1 <session-config>
2   <session-timeout>900</session-timeout>
3 </session-config>
```

推荐在 web.xml 中配置会话超时。

外部资源的引用配置

Web 应用的 Web 组件经常需要各种外部资源和服务，如使用 Web 服务器配置的数据库连接池、JMS 消息服务等。
在 web.xml 中引入资源或服务的配置方法，以配置 JNDI 服务为例：

```
<resource-ref>
  <description>DB connection</description>
  <res-ref-name>java:comp/env/cityoa<res-ref-name>
  <ref-type>java.sql.DataSource</ref-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

取得此连接池的方法：

```
Context context = new InitialContext(); // 初始化JNDI 服务对象
DataSource ds = (DataSource) context.lookup("java:comp/env/cityoa"); // 使用引用名
Connection cn = ds.getConnection(); // 取得连接池中的一个数据库连接
```

接下来…

Servlet 配置对象 ServletConfig

Java EE 为取得 Servlet 的配置信息，提供一个 Servlet 配置对象 API。该对象在 Servlet 初始化阶段由 Web 容器实例化，并将当前 Servlet 的配置数据写入到此对象，供 Servlet 读取使用。

配置对象的类型和取得

配置对象类型：javax.servlet.ServletConfig，是一个接口，具体实现类由服务器厂家实现。

ServletConfig 对象在 Servlet 的 init 方法中取得，由 Web 容器以参数方式注入到 Servlet：

```
1  private ServletConfig config = null;
2
3  public void init(ServletConfig config) throws ServletException {
4    super.init(config);
5    this.config = config;
6  }
```

1. 要取得 ServletConfig 对象需要重写 init 方法，并传递 ServletConfig 参数。然后在 doGet 和 doPost 方法中即可以使用 config 对象。
2. 与 ServletContext 和 HttpSession 对象不同，Web 容器为每个 Servlet 实例创建一个 ServletConfig 对象，不同 Servlet 之间无法共享此对象。

ServletConfig 功能和方法

public String getInitParameter(String name)

取得指定的 Servlet 配置参数。与 Web 初始参数不同，Servlet 初始参数在 Servlet 声明中定义。

```
<servlet>
  <servlet-name>ServletConfigTest</servlet-name>
  <servlet-class>javaee.nseg.ServletConfigTest</servlet-class>
  <init-parm>
    <param-name>url</param-name>
    <param-value>jdbc:oracle:thin:@210.30.108.5:1521:oracle01</param-value>
  </init-parm>
  <init-param>
    ...
  </init-param>
</servlet>
```

注意：<init-param> 标签要放置在 <servlet-name> 和 <servlet-class> 后，否则编译错误。

取得配置的 Servlet 初始参数：

```
String url = config.getInitParameter("url");
```

ServletConfig 功能和方法

`public Enumeration getInitParameterNames()`

取得所有 Servlet 初始化参数。

`public String getServletName()`

取得 Servlet 配置的名称。

`public ServletContext getServletContext()`

ServletConfig 对象提供了取得 ServletContext 对象的方法，与在 Servlet 内使用 this.getServletContext() 一样，返回 ServletContext 实例的对象引用。

```
1 ServletContext ctx = config.getServletConfig();
```

ServletConfig 应用案例

注意课下练习。

接下来…

Web 跳转方式

Web 应用在运行中需要不断的在各个页面之间进行跳转和传递数据。

- 重定向
- 转发

Web 跳转方式

❖ 重定向（redirect）

典型的重定向跳转方式如下：

- 地址栏手工输入新的 URL 地址；
- 单击超链接；
- 提交 FORM 表单；
- 使用响应对象 response 的 sendRedirect() 方法。

重定向跳转方法都是由客户端浏览器来执行的。由此可见重定向增加了网络的访问流量。

Web 跳转方式

❖ 转发（forward）

- 转发是在服务器端进行页面直接跳转的方法。
- 转发是指 Web 组件在服务器端直接请求到另外 Web 组件的方式。
- 转发在 Web 容器内部完成，不需要通过客户端浏览器，因此客户端浏览器的地址还停留在初次请求的地址上。

Web 开发中应该尽量使用转发实现 Web 组件之间的导航。

转发的实现

转发对象 API： javax.servlet.RequestDispatcher

❖ 取得转发对象

通过请求对象 HttpServletRequest 取得

```
1 RequestDispatcher rd = request.getRequestDispatcher("main.jsp");
```

使用 ServletContext 对象的方法取得

```
1 RequestDispatcher rd = this.getServletContext().getRequestDispatcher("/main.jsp");
```

实现转发

取得转发对象后，调用转发对象的方法 forward 完成转发。

```
1 RequestDispatcher rd = request.getRequestDispatcher("main.jsp");
2 rd.forward(request, response);
```

关于目标页面 main.jsp 的目录说明：

- 如果上述代码所在的 Servlet 被映射到/employee/main.action（一个虚拟请求地址），则转发的目标页面 main.jsp 也需要放到/employee 目录下；
- 如果 main.jsp 和 Servlet 映射地址不在同一目录，就需要使用相对路径定位，如把 main.jsp 放在/department/main.jsp，则取得转发对象需要按照如下示例代码进行修改：

```
1 RequestDispatcher rd = request.getRequestDispatcher("../department/main.jsp");
```

实现转发的两种方法的区别

- 从请求对象得到的转发对象要求使用相对路径

```
1 RequestDispatcher rd = request.getRequestDispatcher("../department/main.jsp");
2 rd.forward(request, response);
```

- 从 ServletContext 对象取得的转发对象要求使用绝对路径，即以“/”开头，否则会抛出 java.lang.IllegalArgumentException 异常。

```
1 RequestDispatcher rd = request.getRequestDispatcher("/department/main.jsp");
2 rd.forward(request, response);
```

转发之间传递数据

- 与重定向不同，转发是在一次请求过程中完成的，在此过程中，转发目标可以与原始请求对象共用请求对象。
- 在进行转发之前，将传递数据存入请求对象属性，然后进行转发；转发目标可以在请求对象中取得存入的数据，从而完成数据的传递。

转发之间传递数据

1. 原始 Servlet 保存数据到请求对象

   Servlet：EmployeeMainAction；URL 地址：/employee/main.action；

   doGet 方法：

```
request.setAttribute("userId", "kevin");
RequestDispatcher rd = request.getRequestDispatcher("view.action");
rd.forward(request, response);
```

2. 转发目标 Servlet 取得保存数据

   Servlet：EmployeeViewAction；URL 地址：/employee/view.action；

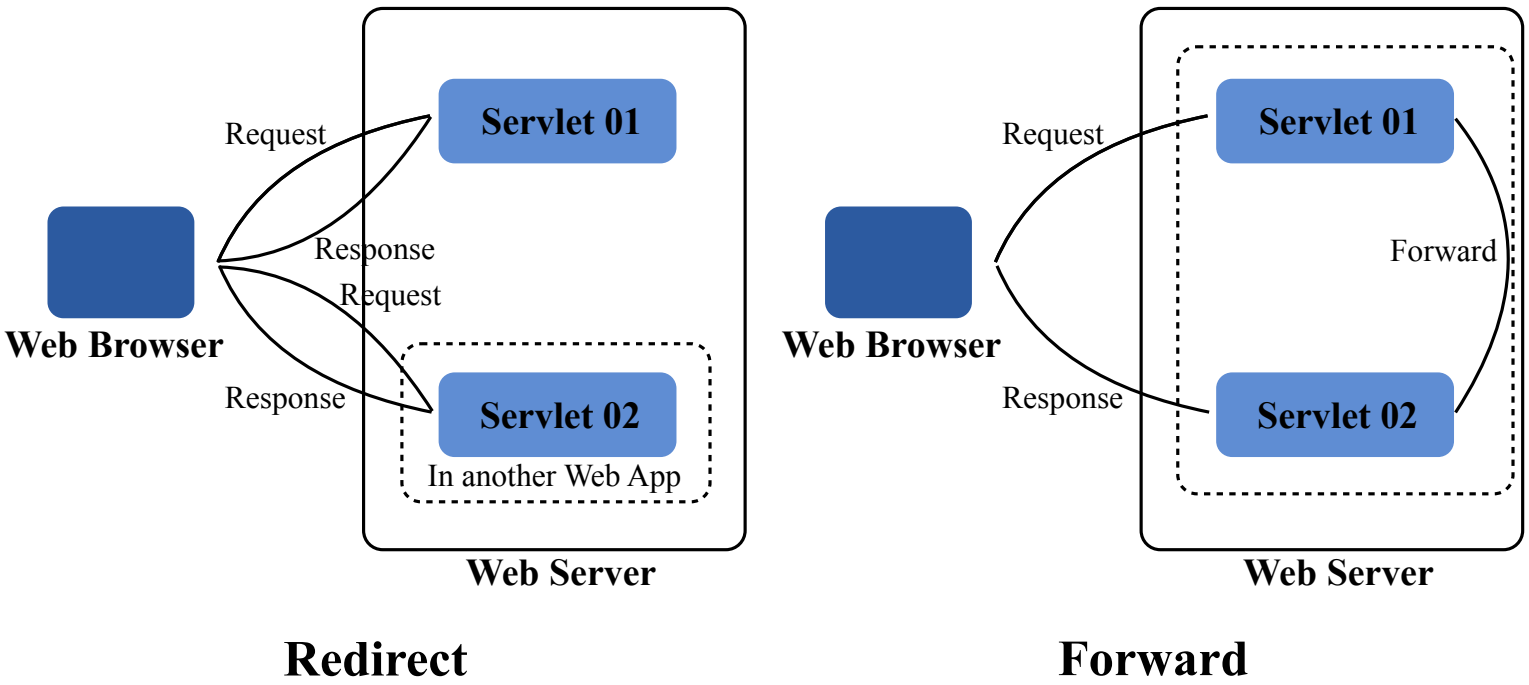
   doGet 方法：

```
String userId = (String) request.getAttribute("userId");
if (userId != null) {
  out.println("账号信息：" + userId);
}
```

总结：Servlet 之间共享数据的方法

1. **使用 ServletContext 对象：**对象生命周期长，会长时间占用内存。
2. **使用会话对象：**对象生命周期较长，会长时间占用内存。
3. **使用请求对象，基于转发传递数据：**对象生命周期短，内存会及时释放。

转发与重定向的区别


Request
Servlet 01
Response
Request
Web Browser
Response
Servlet 02
In another Web App
Web Server
Redirect
Request
Servlet 01
Forward
Web Browser
Response
Servlet 02
Web Server
Forward

转发与重定向的区别

1. 发生的地点不同
   重定向由客户端完成，而转发由服务器完成。
2. 请求/响应的次数不同
   重定向两次请求，创建两个请求对象和响应对象，而转发是一次请求，只创建一个请求对象和响应对象。重定向无法共享请求/响应对象，而转发可以。
3. 目标位置不同
   重定向可以跳转到 Web 应用以外的文档，而转发只能在一个 Web 内部文件中间进行。

转发编程的注意事项

1. 转发目录与源目录要在同一个目录。
2. 转发之前不应有响应发送，否则导致异常 javax.servlet.IllegalStateException 抛出。
3. 更改请求目录最好在重定向中。

THE END

wxd2870@163.com