第3章 基本Servlet程序设计

Java EE建立在Java SE的基础上，为开发和部署企业应用程序提供API和服务。将Java SE与Java EE的服务以及API结合起来，将有助于开发独立于系统平台、基于Web的Java EE应用。Servlet属于Java EE应用中的Web组件，是JSP技术的基础，而且大型的Web应用开发需要Servlet与JSP相互配合才能完成。

本章将介绍Servlet的基本概念、Web服务器与Web容器的关系、基本Servlet结构，通过实例介绍在Oracle JDeveloper 10g环境下开发、部署和运行Servlet的基本方法和步骤。

3.1 Servlet的基本概念

Servlet是CGI程序设计的Java解决方案，是一种用于服务器端程序设计的Java API。Servlet自从1997年诞生以来，由于具有平台无关性、可扩展性以及能够提供比CGI脚本程序更加优越的性能，因此它的应用量快速的增长，其成为Java 2企业应用平台的一个关键组件。Servlet是一些Java类，用于动态地处理请求以及构造响应信息。它们动态地生成HTML Web页面作为对请求的响应，还可以向客户以其他格式发送数据。例如，串行化Java对象——Applet和Java应用程序，以及XML。这些Servlet在一个Servlet容器中运行，并且可以访问由该容器提供的服务。Servlet的客户可以是一个浏览器、Java应用程序，或者任何其他可以构造一个请求并从中接收响应的客户。当然，正常情况下这些请求是Servlet可以识别和响应的HTTP请求。

在用Servlet和Java EE技术进行Java企业服务器端编程时，Web服务器从客户机接收请求，并把该请求映射到适当的资源上。如果该请求是一个静态资源，则它会简单地返回该资源给相关的客户机。请求的对象也可以是一个Java EE组件(如Servlet)。在这种情形下，Java EE服务器会提供一个Web容器给Web服务器，Web服务器接着把这个对容器组件的

请求转发给指定的容器，随后由该容器把请求转发给相应的组件，再由该组件处理请求并且返回一条响应信息，其执行过程如图 3-1 所示。

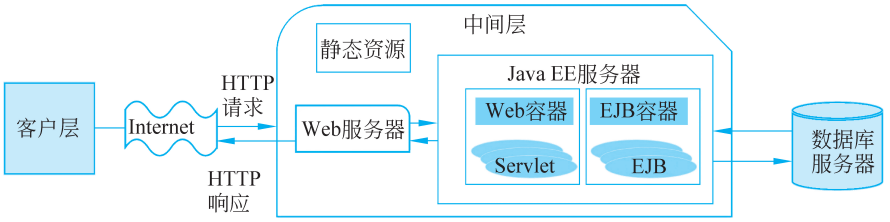

图 3-1 Servlet 的基本执行流程

中间层由 Web 服务器和 Java EE 服务器组成。一个 Java EE 服务器上有 2 个容器，即 Web 容器和 EJB 容器。Web 容器就是管理着一个 Web 应用的所有 Servlet 和 JSP 运行的 Java 环境，它是一个 Java EE 服务器的组成部分，它的请求来自 Web 服务器。它必须支持 HTTP，并且可以选择支持其他协议。它可以建立在一个 Java EE 服务器中，也可以作为一个组件插入 Web 服务器中。同时，它还负责管理 Servlet 和 JSP 实例的调用和生存周期。

EJB 容器是包含业务规则或逻辑的业务组件，主要提供对 JDBC 等 Java EE API 的访问。EJB 有两种类型：会话 Bean 是面向逻辑的，并处理客户机的请求，也进行数据逻辑的处理；实体 Bean 与数据本身紧密耦合，并处理数据访问和持久性。在客户机端，应用程序容器是由 Java EE 提供的在客户机上运行的 Java 应用程序，它通常使用 AWT、Swing API，以及 JavaFX 构造 GUI。Servlet 不是用户直接调用的程序，而是由实施该 Servle 的 Web 应用中的 Web 容器根据进入的 HTTP 请求调用的 Servlet。当一个 Servlet 被调用后，Web 容器把进入的请求信息转发到该 Servlet，这样 Servlet 就可以处理它并且生成动态响应信息。Web 容器通过接收 Servlet 的请求与 Web 服务器交互，并且把响应信息回送到 Web 服务器。

3.2 基本 Servlet 结构

Java Servlet API 2.3 为以 Servlet 技术为基础的基于 Java EE 和 Java SE 的 Web 应用开发提供了成熟的技术。Servlet 2.3 API 包括 javax.servlet 和 javax.servlet.http 两个包。第一个包包含所有的 Servlet 实现和扩展的通用接口和类；第二个包包含在 HTTP 实现特定的 Servlet 时所需要的扩充类。

基本 Servlet 结构的核心部分是 javax.servlet.Servlet 接口，它提供了所有 Servlet 的框架结构。Servlet 接口提供了 5 种方法，其中 3 个最重要的方法是：init()方法对 Servlet 进行初始化；service()方法负责接收和响应客户请求；destroy()方法执行清除对象等收尾工作。所有的 Servlet 必须实现这个接口，实现的方式或者是直接地，或者是通过继承的方式。

3.2.1 GenericServlet 与 HttpServlet

在 Servlet API 2.3 中，两个主要的类是 GenericServlet 与 HttpServlet，而

HttpServlet 类是从 GenericServlet 类继承而来的。开发 Servlet 时，通常的做法是继承这两个类中的一个。Servlet 中没有 main()方法，这就是为什么所有的 Servlet 都必须实现 javax.servlet.Servlet 接口的原因。每当 Web 服务器收到一个指向某个 Servlet 的请求时，它总要调用 Servlet 的 service()方法。

当用户的 Servlet 继承 GenericServlet 类时，必须实现 service()方法。GenericServlet 类的 service()方法是一个抽象方法，其定义如下。

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

- ServletRequest——含有发送给 Servlet 的信息，用来保存客户机向服务器发出请求的各种属性，如 IP 地址。
- ServletResponse——保存返回给客户机的数据，如设置服务器如何对客户机进行响应。

与 GenericServlet 类不同，当用户的 Servlet 继承 HttpServlet 类时，不需要实现 service()方法，HttpServlet 类已经为用户实现了。HttpServlet 类的 service()方法的定义如下。

```
protected void service (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException;
```

当 HttpServlet.service()方法被调用时，它将读取请求中存储的方法类型，然后基于该值确定应该调用哪一个方法，这些方法是被强制执行的。如果方法的类型是 GET，service()方法将调用 doGet()方法；如果方法的类型是 POST，service()方法将调用 doPost()方法。

3.2.2　Servlet 的生命周期

Servlet 的生命周期如下。

(1) Servlet 由 Web 容器初始化，然后再处理请求。

(2) Servlet 组件从客户层接收请求。

(3) Servlet 处理相应的请求；

(4) 一旦处理完毕，就会向客户层返回一条响应信息。

(5) 最后，Web 容器负责销毁自己生成的任何 Servlet 实例。

上述执行过程中的第一步和第五步只执行一次，第二、第三和第四步将循环多次，以处理众多的请求。

javax.servlet.Servlet 接口说明了 Servlet 生命周期的框架结构。这个接口定义了 Servlet 生命周期的 3 个方法：init()、service()、destroy()。

1. init()方法

init()方法是 Servlet 生命周期的起点，一旦加载了某个 Servlet，服务器立即调用它的 init()方法。在 init()方法中，Servlet 将创建和初始化它在处理请求时要用到的资源，例如数据库连接。init()方法的定义如下。

```
public void init(ServletConfig config) throws ServletException;
```

init()方法使用 ServletConfig 对象作为参数。用户应该保存这个对象，以便在后续程序中引用。一般用如下的方式定义 init()方法。

```
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    ...
}
```

2. service()方法

service()方法处理客户发出的所有请求。在 init()方法执行之前，service()方法无法对客户的请求提供服务。因此，它不能直接实现 service()方法，除非继承了 GenericServlet 抽象类。

3. destroy()方法

destroy()方法标志着 Servlet 生命周期的结束。当服务需要关闭时，Web 容器调用 Servlet 的 destroy()方法。此时，在 init()方法中创建的任何资源都将被清除和释放。例如，如果有打开的数据库连接，就应当被关闭。destroy()方法的定义如下。

```
public void destroy();
```

3.3 基于 JDeveloper 开发 Servlet

本节通过一个实例介绍在 Oracle JDeveloper 10g 环境下怎样创建一个基本的 Servlet。下面对生成的 Servlet 源代码进行分析，这是主要着眼于这个 Servlet 的每个组成部分、Servlet 的实现方法以及 Servlet 使用的对象。

3.3.1 创建基本的 Servlet

(1) 启动 Oracle JDeveloper，创建一个新的工作区，如图 3-2 所示。

图 3-2 创建一个新的工作区

（2）单击“确定”按钮，将显示如图 3-3 所示的对话框，让用户创建一个工程文件。输入项目文件名为 BasicServlet，然后单击“确定”按钮，则可以完成工作区和工程文件的创建。

图 3-3　创建一个新的工程

（3）在创建的工程文件中增加一个 Servlet 对象。从 IDE 主菜单中选择 File→New 命令，将显示如图 3-4 所示的对话框。选择 Web Tier→Servlets→HTTP Servlet，单击“确定”按钮，将会显示 HTTP Servlet Wizard，如图 3-5 所示。

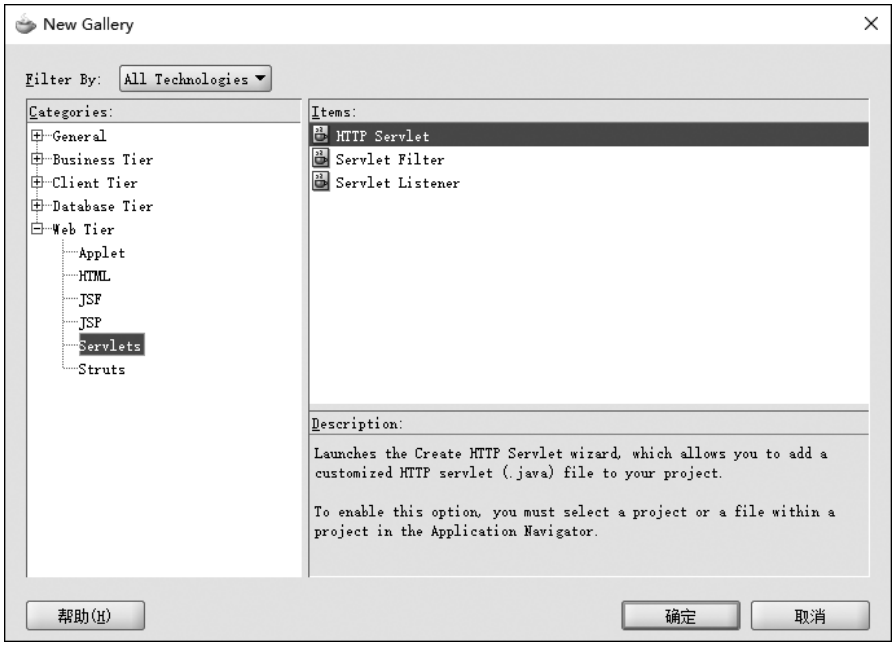

图 3-4　增加一个 Servlet 对象

（4）单击“下一步”按钮，将会显示 Web 应用版本对话框，如图 3-6 所示。单击“下一步”按钮，将会显示创建 Servlet 对话框，如图 3-7 所示。

（5）如图 3-7 所示，分别输入 Class 与 Package 域，勾选 doGet()与 doPost()复选框。之后单击“下一步”按钮，将会显示 Servlet 的 URL 映射名字对话框，如图 3-8 所示。

（6）图 3-8 用来指定创建的 Servlet 的 URL 映射名字和 URL 前缀。这样，在浏览器中运行该 Servlet 时，Web 容器 OC4J 就知道这个 Servlet 的具体存放位置。

图 3-5　HTTP Servlet Wizard

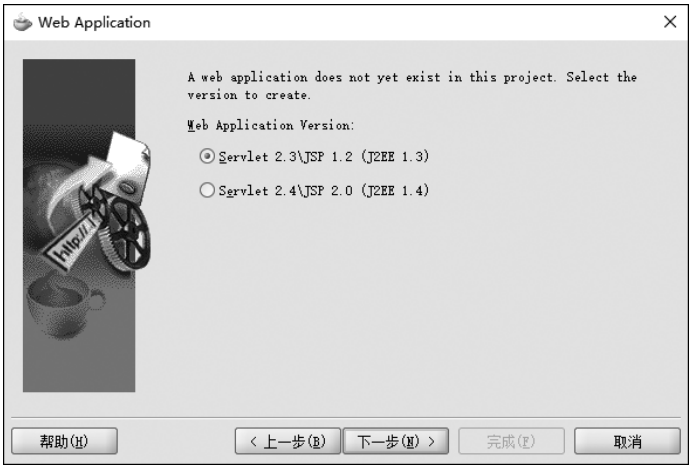

图 3-6　Web 应用版本对话框

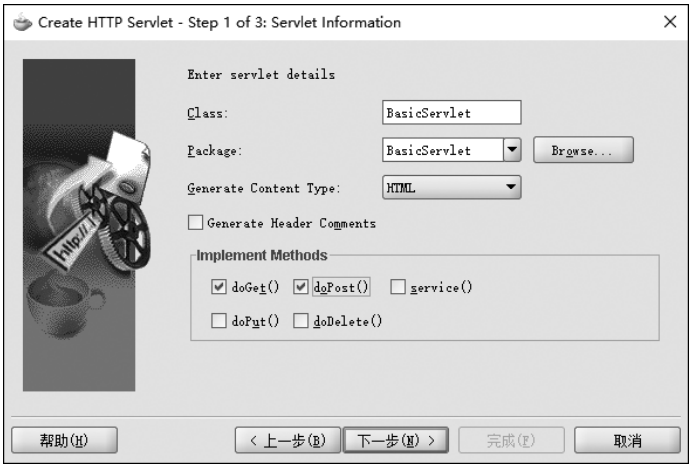

图 3-7　创建 Servlet 对话框

图 3-8　Servlet 的 URL 映射名字对话框

(7) 单击“完成”按钮，就可以在 Code Editor 窗口得到如下所示的 BasicServlet.java 的源代码。

```
package basicServlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
public class BasicServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html; charset=GBK";
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>BasicServlet</title></head>");
        out.println("<body>");
        out.println("<p>这个 servlet 接收一个 GET 请求.</p>");
        out.println("</body></html>");
        out.close();
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
```

```
            out.println("<html>");
            out.println("<head><title>BasicServlet</title></head>");
            out.println("<body>");
            out.println("<p>The servlet has received a POST. This is the reply.</p>");
            out.println("</body></html>");
            out.close();
        }
    }
```

(8) 从 IDE 的主菜单中选择 Run→Run BasicServlet.jpr，运行该工程。JDeveloper 首先启动内嵌在 IDE 中的 OC4J，并常驻内存中，然后启动默认的 IE 浏览器，运行结果如图 3-9 所示。

图 3-9 BasicServlet.jpr 的运行结果

3.3.2 分析 BasicServlet 类

从上述开发过程可以看到，在 Oracle JDeveloper 环境下开发 Servlet 给开发人员带来了极大的便利。下面分析 BasicServlet 类的组成及使用方法。

1. BasicServlet 类的基本组成结构

首先，BasicServlet 类继承了 HttpServlet 类。HttpServlet 是一个抽象类，简化了 HTTPServlet 的编码工作。同时，HttpServlet 类继承了 GenericServlet 类，提供了处理 HTTP 特定请求的功能。

2. BasicServlet 类继承的方法

BasicServlet 类重写了它继承的 3 个方法，下面说明这 3 个方法的用法。

(1) init()方法

init()方法首先读取传递给它的 ServletConfig 对象，再把该对象传递给它的父 init()方法，否则将保存该对象以备以后使用。一个 Servlet 可以使用 ServletConfig 对象访问其配置数据。init()方法如果不能正常完成，将抛出一个 ServletException。

Servlet 技术规范保证了 init()方法只在 Servlet 的任何给定的实例上被调用一次，init()方法被允许在任何请求传递到该 Servlet 之前完成。

执行上述动作的代码如下。

```
super.init(config);
```

在 init()方法中可以实现下列一些典型任务。

- 从配置文件中读取配置数据。
- 使用 ServletConfig 对象读取初始化参数。
- 初始化诸如注册一个数据库驱动程序、连接日志记录服务等这样的一次性活动。

BasicServlet 在 init()方法中没有创建任何资源，所以也就没有定义 destroy()方法。

(2) doGet()和 doPost()方法

这两个方法的唯一区别是它们的服务的请求类型不同。doGet()方法处理 GET 请求，doPost()方法处理 POST 请求。它们均接收 HttpServletRequest 和 HttpServletResponse 对象，它们封装了 HTTP 请求/响应信息。HttpServletRequest 对象包含客户机发出的信息，而 HttpServletResponse 对象则包含回送给客户机的信息。在这两个方法中被执行的第一条语句如下。

```
response.setContentType(CONTENT_TYPE);
```

这个语句用于设置响应的内容类型和字符编码，这个响应属性只能设置一次，在开始向 Writer 或 OutputStream 输出流输出信息之前，必须先设置这个属性。在 BasicServlet 类中，正在使用的是 PrintWriter，所以把响应类型设置为 text/html。

下一步要做的就是创建 PrintWriter 的对象，以便通过输出流对象把 HTML 文本信息输出到客户机的浏览器上。可以通过调用 ServletResponse 的 getWriter()方法达到这一目的。执行这个动作的语句如下。

```
PrintWriter out=response.getWriter();
```

现在就有了一个输出流对象的引用，它允许输出 HTML 文本，并回送给 HttpServletResponse 对象所对应的客户机浏览器，下面的程序片段描述了这个过程。

```
out.println("<html>");
out.println("<head><title>BasicServlet</title></head>");
out.println("<body>");
out.println("<p>The servlet has received a GET. This is the reply.</p>");
out.println("</body></html>");
out.close();
```

上述程序片段是一个非常直观地把 HTML 文本回送给客户机浏览器的方法，需要做的仅是把在响应中包含的 HTML 文本传递给 PrintWriter 对象的 println()方法，然后再关闭输出流。

通过上面内容的学习，读者对 Servlet 的组成有了一定程度的了解，也知道了编制自己的 Servlet 的哪一部分需要符合 Servlet API 的框架结构要求，即类与类、类与接口之间的继承关系。因此，读者现在能够创建自己的基本 Servlet 了。

3.3.3　部署 Web 应用

1. 创建与 OC4J 的连接

(1) 启动 OC4J，启动命令以及启动后的提示信息如图 3-10 所示。

(2) 连接 Web 容器 OC4J。在 Oracle JDeveloper IDE 的系统导航窗口展开

图 3-10 启动 OC4J

Connections 节点，选择 Application Server 对象，右击，在弹出的快捷菜单中选择 New Connection 命令，将显示如图 3-11 所示的连接 OC4J 向导窗口。

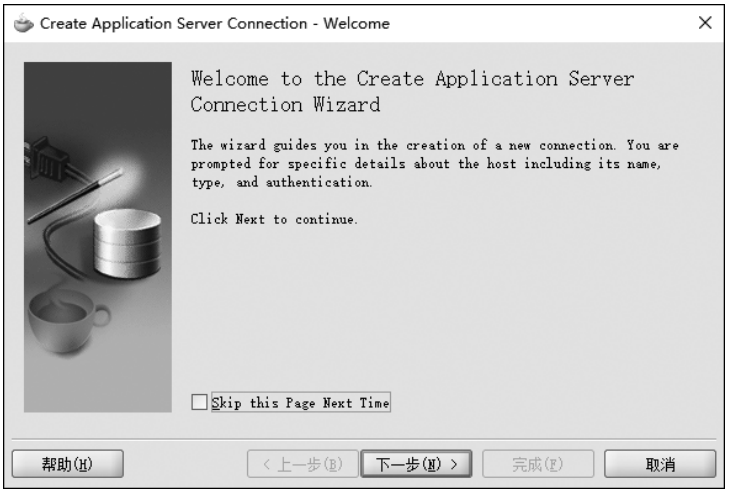

图 3-11 连接 OC4J 向导窗口

（3）单击“下一步”按钮，将会显示如图 3-12 所示的对话框。在 Connection Name 域输入 OracleAS10g，Connection Type 选择独立 OC4J 10g 10.1.3，如图 3-12 所示。

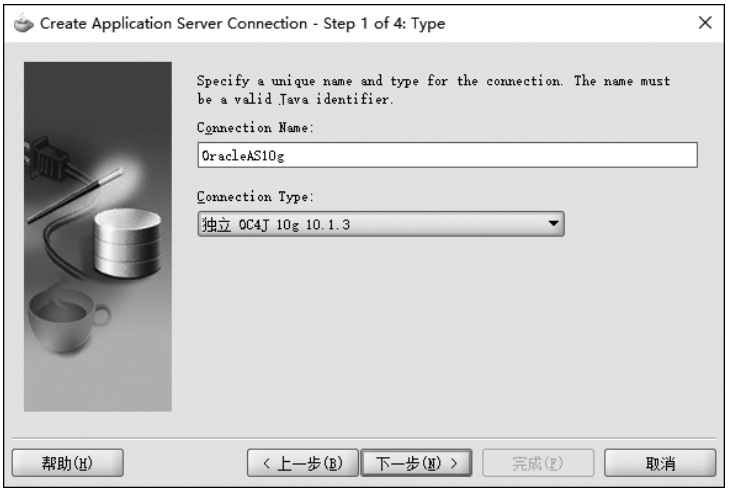

图 3-12 确定连接名称与连接类型

(4) 单击“下一步”按钮,将会显示如图 3-13 所示的对话框。Username 域使用默认值,Password 域输入连接 OC4J 的密码,并勾选 Deploy Password 复选框。

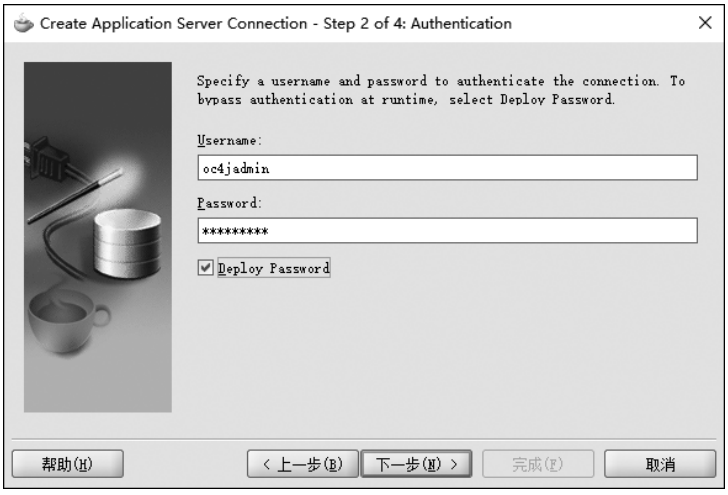

图 3-13　确定用户名与密码

(5) 单击“下一步”按钮,将会显示如图 3-14 所示的对话框。这里,输入 Host Name 的值为 Dell(作者所使用计算机的默认机器名),URL Path 可以省略。

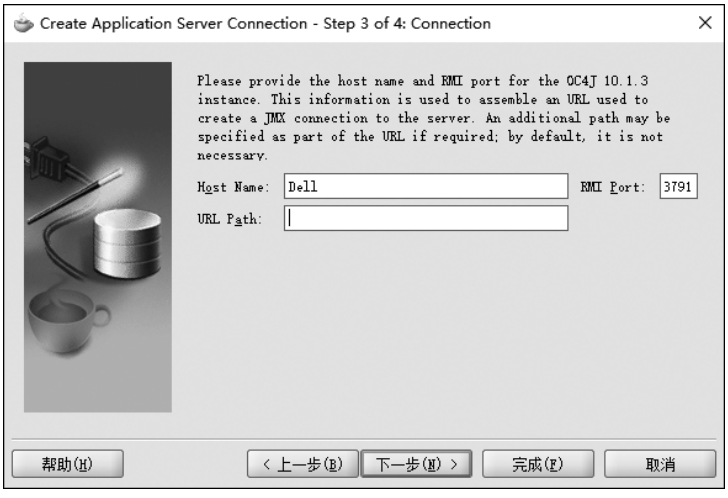

图 3-14　确定连接 URL 等值

(6) 单击“下一步”按钮,将会显示如图 3-15 所示的对话框。单击 Test Connection 按钮,如果显示 Success! 信息,则说明已经连接成功。单击“完成”按钮,完成与 OC4J 的连接。

2. 创建 Web 应用的部署描述文件

(1) 在工程文件 BasicServlet 中添加一个 Web 对象。从 IDEE 的主菜单中选择 File→New 命令,在所显示的如图 3-16 所示对话框的 Categories 区域选择 General→

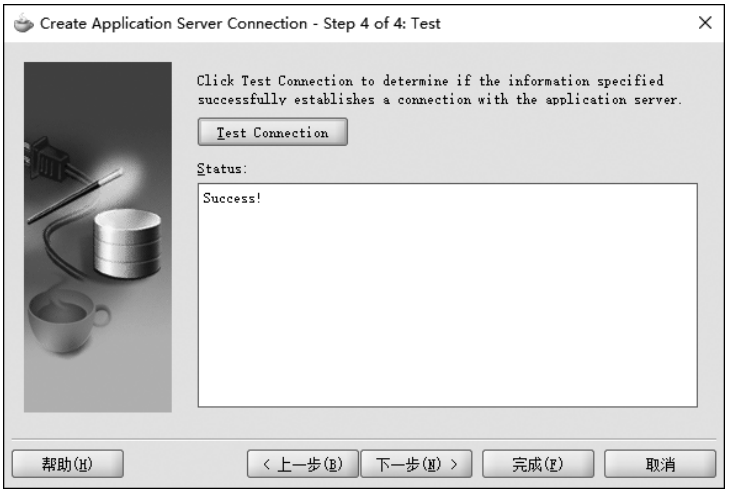

图 3-15　测试连接是否成功

Deployment Profiles，在 Items 区域选择 WAR File，单击“确定”按钮，将会显示如图 3-17 所示的对话框。将 File Name 域的文件名改为 BasicServlet.deploy。

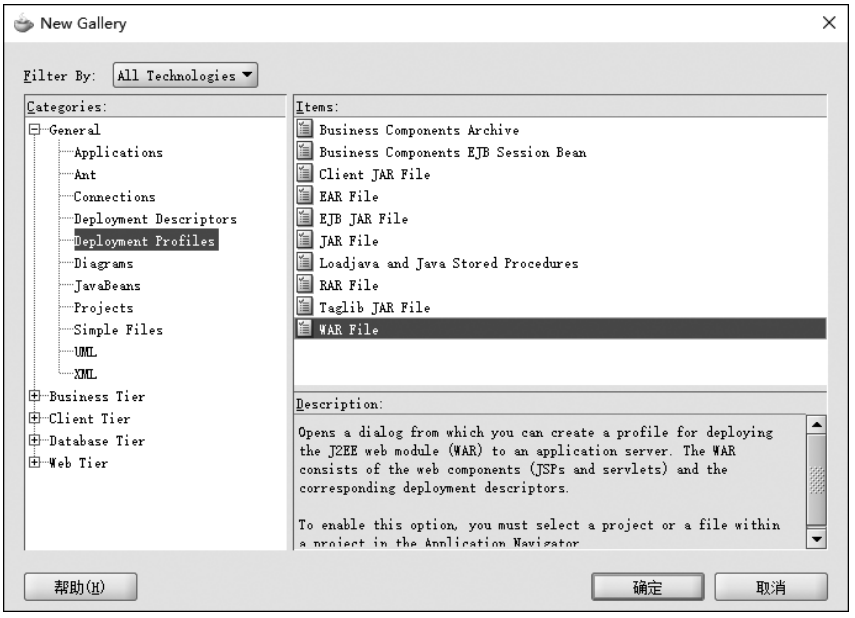

图 3-16　创建部署描述文件

(2) 单击“确定”按钮，将会显示如图 3-18 所示的 Web 应用的部署信息。单击“确定”按钮，将完成 Web 应用的部署描述文件的创建工作。

3. 部署与运行 Web 应用

在系统导航窗口选择 BasicServlet.deploy，右击，从弹出的快捷菜单中选择 Deploy to OracleAS 10g，如图 3-19 所示。

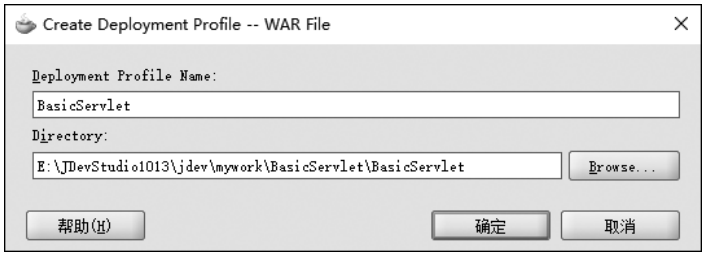

图 3-17　确定部署描述文件的名称和路径

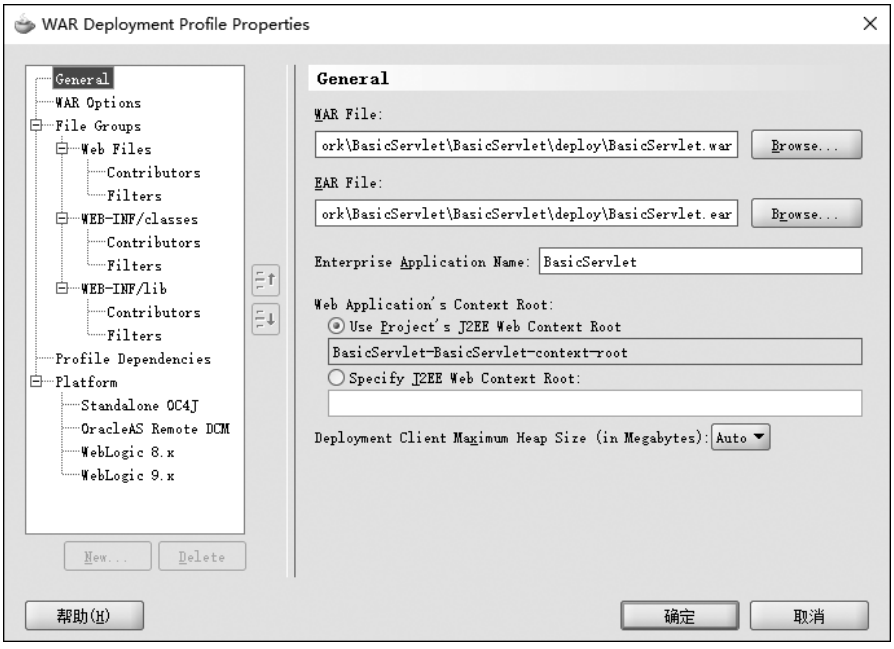

图 3-18　Web 应用的部署信息

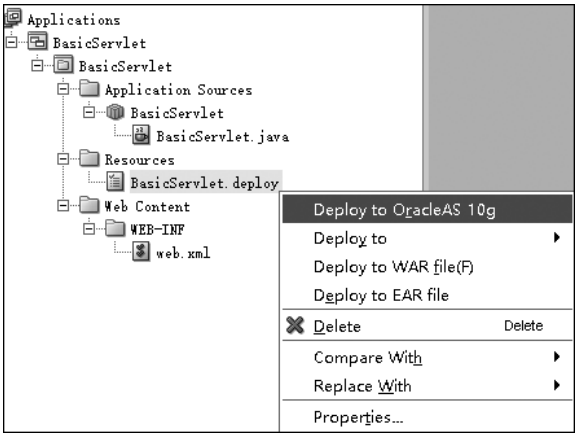

图 3-19　部署 Web 应用到 Web 容器

此时，IDE 将根据用户的配置进行 BasicServlet.jpr 工程的部署工作。在部署过程中，信息浏览工作区将不断显示 Web 应用的部署信息。如果用户的配置有错误，也会将错误信息显示出来。如果最后显示如下信息，则说明 Web 应用的部署配置成功完成。

```
----  Deployment started.  ----2021-7-20 11:56:12
Target platform is 独立 OC4J 10g 10.1.3 (OracleAS10g).
Wrote WAR file to E:\JDevStudio1013\jdev\mywork\BasicServlet\BasicServlet\
deploy\BasicServlet.war
Wrote EAR file to E:\JDevStudio1013\jdev\mywork\BasicServlet\BasicServlet\
deploy\BasicServlet.ear
BasicServlet 的 Application UnDeployer 开始。
......
初始化 BasicServlet 开始…
初始化 BasicServlet 结束…
已启动的应用程序: BasicServlet
将 Web 应用程序绑定到站点 default-web-site 开始…
将应用程序 BasicServlet 的 BasicServlet Web 模块绑定到上下文根 BasicServlet-
BasicServlet-context-root 下的站点 default-web-site
将 Web 应用程序绑定到站点 default-web-site 结束…
BasicServlet 的 Application Deployer 完成。操作时间: 107 msecs
Elapsed time for deployment: 15 seconds
----  Deployment finished.  ----2021-7-20 11:56:27
```

在 MS-DOS 窗口，改变当前目录为 E:\jdevstudio1013\J2EE\HOME\config，用 IDE 或其他的文本编辑器打开 default-web-site.xml 文件。其文件内容如下。

```
1.  <?xml version="1.0"?>
2.  <web-sitexmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:
    noNamespaceSchemaLocation="http://xmlns.oracle.com/oracleas/schema/web-
    site-10_0.xsd"port="8888" display-name="OC4J 10g (10.1.3) Default Web
    Site" schema-major-version="10" schema-minor-version="0" >
3.  <default-web-app application="default" name="defaultWebApp" />
4.  <web-app application="system" name="dms0" root="/dmsoc4j" access-log=
    "false" />
5.  <web-app application="system" name="dms0" root="/dms0" access-log=
    "false" />
6.  <web-app application="system" name="JMXSoapAdapter-web" root=
    "/JMXSoapAdapter" />
7.  <web-app application="default" name="jmsrouter_web" load-on-startup=
    "true" root="/jmsrouter" />
8.  <web-app application="ascontrol" name="ascontrol" load-on-startup=
    "true" root="/em" ohs-routing="false" />
9.  <web-app application="bc4j" name="webapp" load-on-startup="true" root=
    "/webapp" />
10. <web-app application="BasicServlet" name="BasicServlet" load-on-
```

```
    startup="true"root="/ServletWS-BasicServlet -context-root" />
11.  <access-log path="../log/default-web-access.log" split="day" />
12.  </web-site>
```

在一个 Web 容器中，每个 Web 应用与一个上下文环境(Context)相关联，并且一个 Web 应用中包含的所有资源都相对于其上下文环境而存在。一个 Servlet 上下文环境存在于 Web 容器的一个已知路径中，这个路径提供了用户访问 Web 容器中存放的 HTML 文件和 Servlet/JSP 等 Web 资源的 URL 值的前缀。

在 Web 容器中，default-web-site.xml 文件提供了部署 Web 应用的上下文环境。

- 从第 3 行开始，针对每一个已经部署的 Web 应用提供服务，包括每个 Web 应用的名称和上下文环境(root)，而这个 root 就是 URL 所指定的 Servlet 的映射地址的前缀。即客户机浏览器访问 URL 的前缀。
- 第 10 行处画了波浪线，就是部署的 BasicServlet 这个 Web 应用的上下文环境。因此，可以确定 URL 值为：
  - 对于 HTML 文档——http://dell:8888/ServletWS-BasicServlet-context-root/HTML 文档名
  - 对于 Servlet——http://dell:8888/ServletWS-BasicServlet-context-root/servlet/BasicServlet

图 3-20 所示就是 BasicServlet 的运行结果。

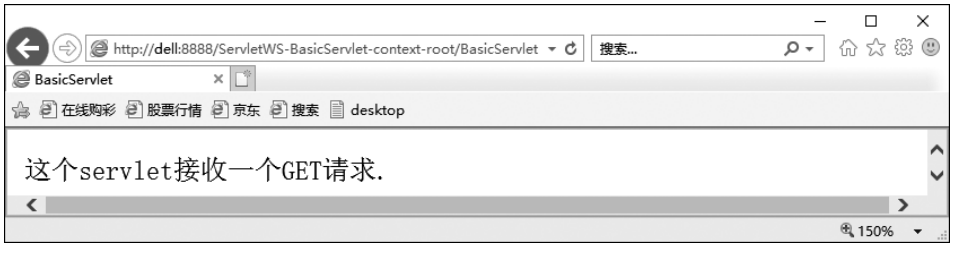

图 3-20　BasicServlet 的运行结果

3.4　本章小结

本章首先介绍了 Java Servlet 的基础知识，并通过实例介绍了在 Oracle JDeveloper 10g 环境下开发、部署和运行 Servlet 的基本方法和步骤，然后分析了一个基本 Servlet 的组成，使读者对一个基本 Servlet 的组成有了一定的认识，也了解了开发 Servlet 的哪一部分需要符合 Servlet 的框架结构要求；最后介绍了怎样确定一个 Servlet 的 URL 的值，以及用户访问这个 Servlet 的 URL 值的方法。