为 JAX-RS web 服务开发 Spring Android 客户端

学习创建 **Spring Android** 客户端

Deepak Vohra, 顾问, Independent

发布日期: 2011 年 9 月 22 日
级别: 中级
原创语言: 英文
访问情况 : 945 次浏览
评论:

简介:

具象状态传输（Representational State Transfer，REST）软件体系结构基于资源表示的传输。RESTful web 服务是一种基于 REST 和 HTTP 协议的 web 服务，并被作为一个 URI 路径可用。此 web 服务由一些生成各种媒体类型（比如 XML、HTML、JSON 和文本）消息的方法组成。这些 web 服务方法响应 GET、PUT、POST 和 DELETE 等 HTTP 方法。RESTful web 服务的 Java API (JAX-RS) 被定义在 JSR 311 中，而 Jersey 是 JAX-RS 的一种参考实现。

Spring 是一种用于运行 Java 企业应用程序的平台，它提供几个优点，比如说提高了生产率和运行时性能。Spring Android 是 Spring 框架的一个扩展，它简化了 Android 应用程序的开发。Spring Android 的主要特性是一个针对 Android 的 REST 客户端和对访问安全 API 的 Auth 支持。

本文中，将学习利用 Spring Android REST 客户端访问 RESTful web 服务。

概述

本文包括以下小节:

- 设置环境
- 创建 JAX-RS web 服务资源
- 安装 Maven 插件
- 创建 Spring Android 客户端
- 配置 Maven 插件和依赖项
- 配置 Android Maven 目标
- 运行 Spring 客户端 Android 应用程序

回页首

设置环境

要设置环境，需完成以下任务。（有关链接，请参见 参考资料。）

- 安装 Eclipse IDE。
- 安装用于 Eclipse 的 Android Development Tools (ADT) 插件。用于 Eclipse 的 ADT 插件提供一组扩展来在 Eclipse 中开发 Android 应用程序。
- 安装 SDK Platform Android 2.2。Android SDK 为开发 Android 应用程序提供工具。
- 在 Eclipse 中创建 Android Virtual Device (AVD)，这是一个用于 Android 的仿真器。
- 还需要安装一个 web 服务器（比如 Tomcat）或者应用程序服务器（比如 WebSphere 或 WebLogic 服务器）。
- 下载包含 Jersey jars 和核心依赖项的 Jersey 归档文件 jersey-archive-1.4.zip。此外，下载 Jersey bundle JAR jersey-bundle-1.4.jar。由于 Jersey 是使用 JDK 6.0 构建的，所以您还需要安装 JDK 6.0。将 清单 1 中所示 JAR 文件添加到应用程序/web 服务器的运行时类路径。

常用缩写词

- API：应用程序编程接口
- HTML：超文本标记语言
- HTTP：超文本传输协议
- IDE：集成开发环境
- JSON：JavaScript 对象符号
- MIME：多用途 Internet 邮件扩展
- POJO：普通 Java 对象
- SDK：软件开发工具箱
- UI：用户界面
- URI：统一资源标识符
- URL：统一资源定位符
- XML：可扩展标记语言

清单 1. 将添加到服务器类路径的 JAR 文件

```
C:\Jersey\jersey-bundle-1.4.jar;C:\Jersey\jersey-archive-1.4\lib\asm-3.1.jar;
C:\Jersey\jersey-archive-1.4\lib\jsr311-api-1.1.1.jar
```

- 下载 Spring Android 项目 ZIP 文件，并解压到一个目录中。

回页首

创建 JAX-RS web 服务资源

本节中，您将创建一个针对 JAX-RS web 服务资源的 Spring 客户端。您的 JAX-RS web 服务将产生三种不同类型的消息，分别具有不同的 MIME 类型：text/plain、`text/xml` 和 `text/html`。

首先，创建一个 Eclipse 项目。

1. 创建一个 web 项目，并向它添加 JAX-RS facet。选择 File > New，并在 New 窗口中选择 Web > Dynamic Web Project。
2. 单击 Next。指定一个项目名称，并为 WebSphere、Tomcat 或 WebLogic 服务器配置一个新的目标运行时。
3. 选择默认的项目设置，并单击 Finish。

Eclipse 创建一个动态 web 项目并将它添加到 Project Explorer。

1. 在 Project Properties 窗口中，配置 JAX-RS (REST Web Services) 1.1 项目 facet。
2. 在 JAX-RS Capabilities 窗口中，指定 Servlet 名为 JAX-RS Servlet，配置一个 JAX-RS Implementation Library，并将 Jersey JARs 添加到该用户库。
3. 添加 Jersey JARs `jersey-bundle-1.4.jar`、`C:\Jersey\jersey-archive-1.4\lib\asm-3.1.jar` 和 `C:\Jersey\jersey-archive-1.4\lib\jsr311-api-1.1.1.jar`。
4. 在 JAX-RS Capabilities 窗口中，指定 JAX-RS servlet 类名为 `com.sun.jersey.spi.container.servlet.ServletContainer`。
   JAX-RS User 库被添加到项目，JAX-RS Servlet 和 Servlet 映射被配置在 `web.xml` 中。
5. 添加 `com.sun.jersey.config.property.resourceConfigClass` 和 `com.sun.jersey.config.property.packages` 初始参数的 init-param 元素。

清单 2 展示了此 `web.xml` 文件。

清单 2. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>EclipseJAX-RS</display-name>
  <servlet>
    <description>JAX-RS Tools Generated - Do not modify</description>
    <servlet-name>JAX-RS Servlet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
             <param-name>com.sun.jersey.config.property.resourceConfigClass</param-name>
             <param-value>com.sun.jersey.api.core.PackagesResourceConfig</param-value>
        </init-param>
        <init-param>
             <param-name>com.sun.jersey.config.property.packages</param-name>
             <param-value>jaxrs</param-value>
        </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JAX-RS Servlet</servlet-name>
    <url-pattern>/jaxrs/*</url-pattern>
  </servlet-mapping>
</web-app>
```

接下来，使用根资源类创建一个 RESTful web 服务资源。根资源类是一个用 @PATH 符号标注的 POJO，它至少由三个用 @GET 符号标注的方法组成，这个符号表示这些方法处理 HTTP GET 请求。将 Java 类要宿主在的 URI 路径指定为 `/helloworld`。参见 清单 3。

清单 3. 资源 URI 路径

```
@Path("/helloworld")
```

```
public class HelloWorldResource {...
}
```

添加资源方法用于生成三种不同的 MIME 类型。添加以下方法到资源类，并用 @GET 符号标注每个方法。

- getClichedMessage()。使用 MIME 类型 text/plain 输出一条 "Hello JAX-RS" 消息。
- getXMLMessage()。使用 MIME 类型 text/xml 输出一条 "Hello JAX-RS" 消息。
- getHTMLMessage()。使用 MIME 类型 text/html 输出一条 "Hello JAX-RS" 消息。

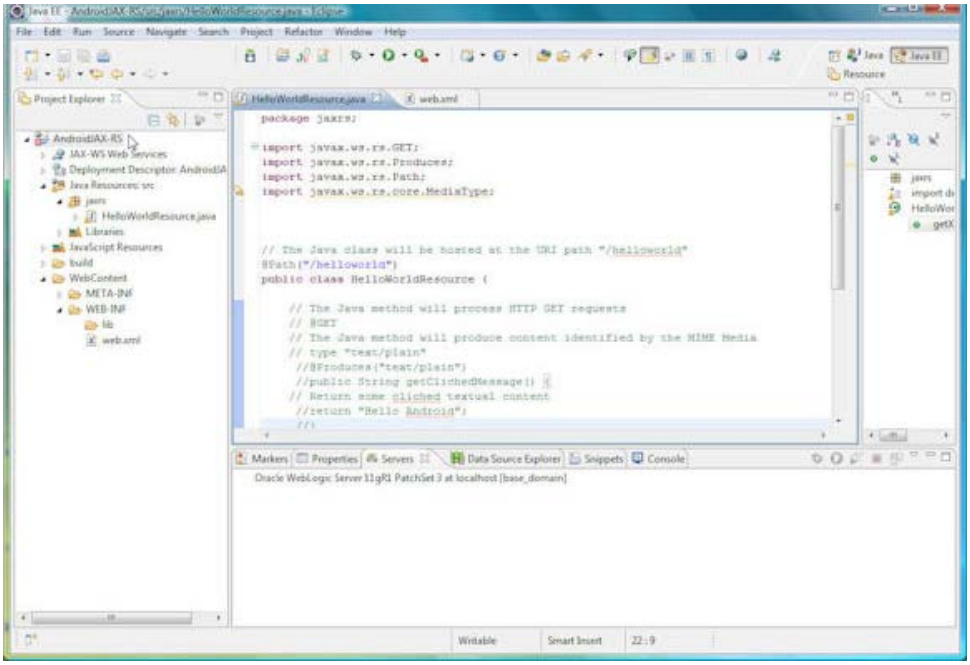
将每个方法的返回类型指定为 String，用 @PRODUCES 标注每个方法，并为它们指定不同的 MIME 类型。getXMLMessage 方法用 @Produces("text/xml") 符号标注，生成 XML 消息。对于每个部署，只取消注释其中一个用 @GET 符号标注的方法。清单 4 展示了此根资源类。

清单 4. JAX-RS web 服务资源类

```
package jaxrs;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;


// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

     // The Java method will process HTTP GET requests
     // @GET
     // The Java method will produce content identified by the MIME Media
     // type "text/plain"
      //@Produces("text/plain")
      //public String getClichedMessage() {
     // Return some cliched textual content
      //return "Hello Android";
      //}

     // @GET
     // @Produces("text/xml")
     // public String getXMLMessage() {
     // return "<?xml version=\"1.0\"?>" + "<hello> //Hello Android" + "</hello>";
     // }

//      @GET
     //@Produces("text/html")
     //public String getHTMLMessage() {
          //return "<html> " + "<title>" + "Hello Android" + "</title>"
          //+ "<body><h1>" + "Hello Android" + "</body></h1>" + "</html> ";
//      }

}
```

图 1 展示了 AndroidJAX-RS 客户端的目录结构。

图 1. JAX-RS web 服务项目



接下来，运行资源类，生成不同类型的输出。

1. 对于每次测试运行，取消注释将被测试的方法。
2. 测试 `text/plain` MIME 类型作为输出。
3. 如果还未启动的话，就启动应用程序/web 服务器。
4. 右键单击资源类，并选择 `Run As > Run on Server`。

`AndroidJAX-RS` 应用程序被部署在服务器上。

回页首

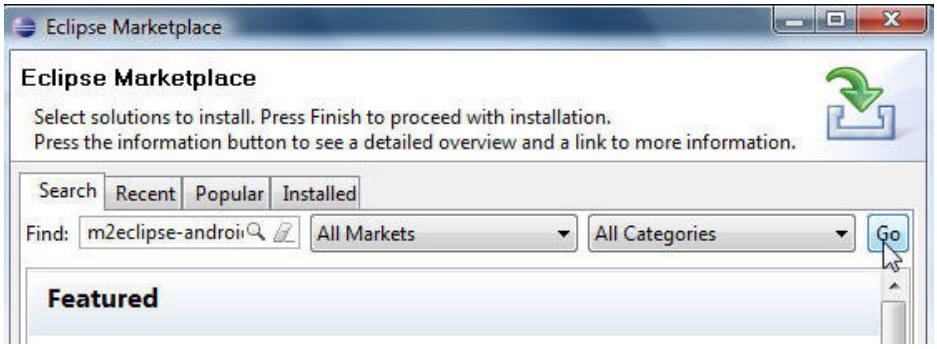
安装 Maven 插件

您将会使用 Apache Maven（一种软件管理工具）来为 Android JAX-RS web 服务的 Spring 客户端构建 Android 项目。使用 Maven Integration 项目向 Eclipse 添加 Maven 支持。对于利用 Maven 的 Android 应用程序开发，您需要用到 Maven Android 插件，这将在后面一节 配置 Maven 插件和依赖项 中安装。Maven Integration for Android Development Tools 是一个 Eclipse 插件，它向 Android Development Tools 和 Maven Android 插件添加对 Maven Integration 的支持。

您可以从 Eclipse Marketplace 向 Android Development Tools 安装 Maven Integration。

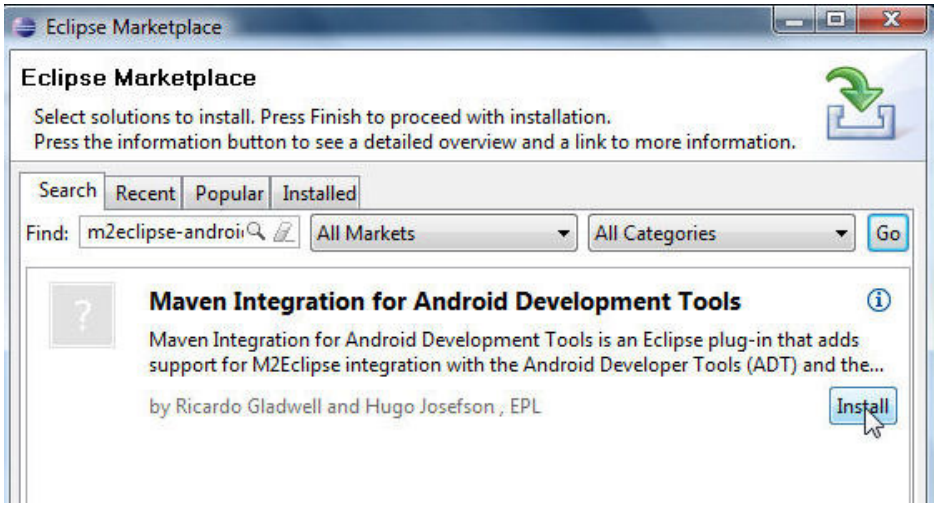
1. 打开 Eclipse IDE 并选择 `Help > Eclipse Marketplace`。
2. 在 Eclipse Marketplace 的 `Search` 选项卡，在 `Find` 字段中指定 `m2eclipse-android` 并单击 `Go`（参见 图 2）。

图 2. 选择 `m2eclipse-android` 插件



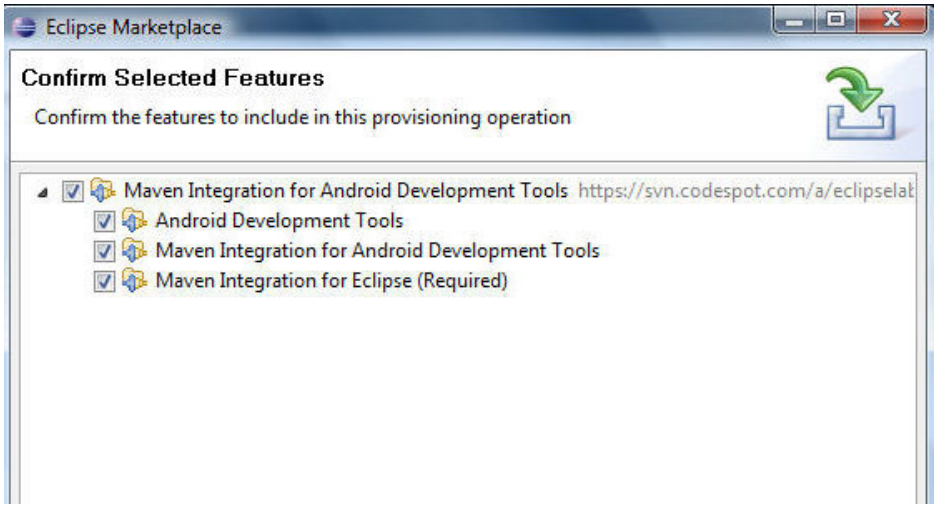
3. Search 选项卡现在列出了 Maven Integration for Android Development Tools。单击 `Install`（参见 图 3）。

图 3. 安装 Maven Integration for Android Development Tools



4. 在 Confirm Selected Features 窗口，选中 Android Development Tools、Maven Integration for Android Development Tools 和 Maven Integration for Eclipse 特性的复选框（参见 图 4）。单击 Next。

图 4. 选择要安装的插件



5. 接受许可协议条款，并单击 Finish，完成插件软件的安装。
   要检查已安装插件，选择 Help > About Eclipse 和 Installation Details in About Eclipse。

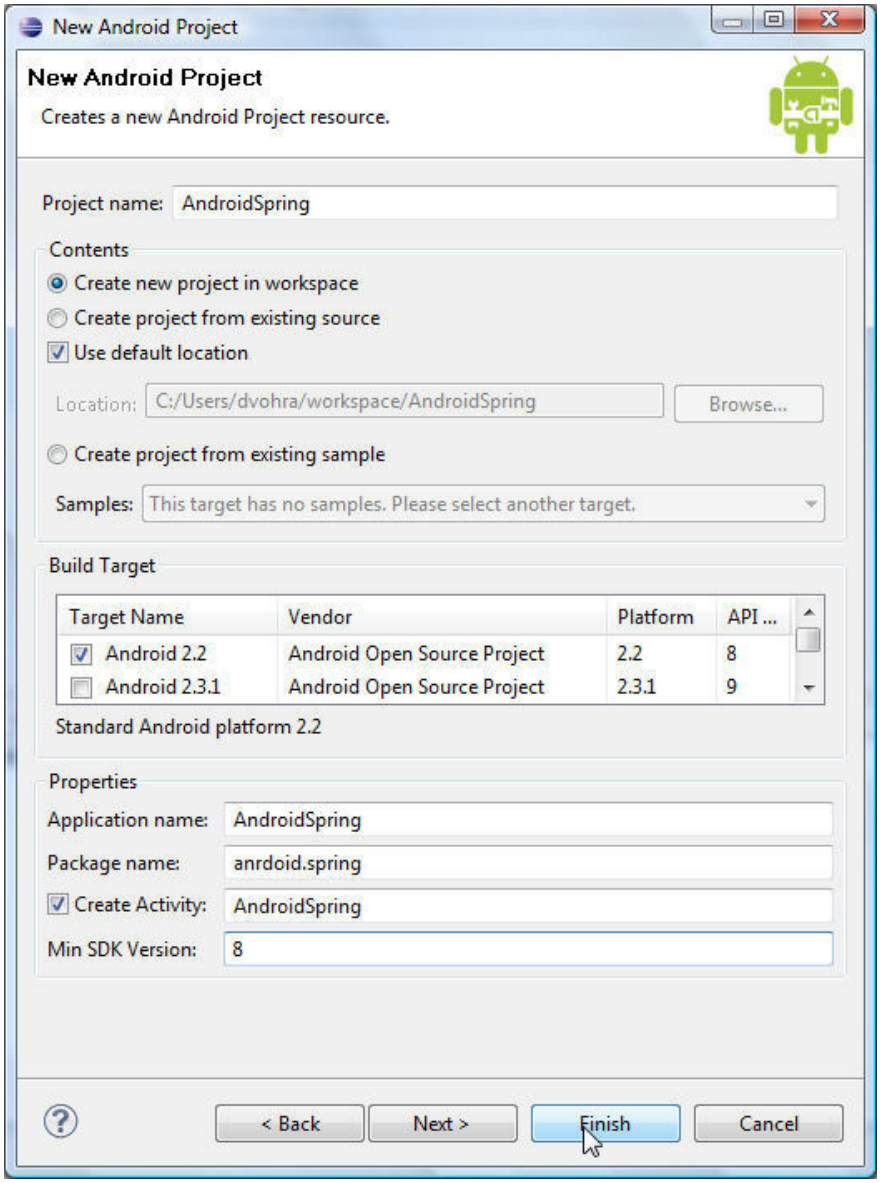
回页首

创建 Spring Android 客户端

本节将为 JAX-RS web 服务创建一个 Android Spring 客户端项目。您创建一个 Android 项目，然后将在该项目中为 Android 创建一个 Spring 客户端，用于访问 JAX-RS web 服务。

1. 在 Eclipse IDE 中，选择 File > New。
2. 在 New 窗口，选择 Android > Android Project。单击 Next。
3. 在 New Android Project 窗口，指定项目名称（AndroidSpring）。
4. 对于 Build Target，选择 Android Platform 2.2 API 8。
5. 对于 Properties，指定一个应用程序名称和一个包名称。
6. 选中 Create Activity 复选框，并指定 Activity 类（AndroidSpring），如 图 5 所示。一个活动代表一次用户交互，它扩展 Activity 类，为 UI 创建一个窗口。
7. 指定最小 SDK 版本为 8，并单击 Finish，如 图 5 所示。

图 5. 创建 Spring Android 客户端



Android 项目由以下文件组成:

- 一个活动类（`AndroidSpring`），它扩展 Activity 类。
- 一个 `res/layout/main.xml` 文件，它指定 Android 应用程序的布局。
- 一个 `AndroidManifest.xml` 文件，它包含应用程序配置，比如包名称、应用程序组件、进程、权限和 Android 系统的最小 API 级别。

在 `res/layout/main.xml` 文件中，在 `LinearLayout` 元素中指定 Android UI 组件的布局。将 `android:orientation` 属性的值指定为 `vertical`。创建一个 UI，来自 web 服务的响应将在此 UI 中显示为文本消息。

添加一个 id 为 "springmessage" 的 `TextView` 元素，以便显示对某个 `get` 方法的方法调用的 JAX-WS web 服务响应。方法调用得到一个 Hello 消息作为响应，形式为 XML、HTML 或文本。清单 5 展示了 `main.xml` 文件。

清单 5. `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView android:id="@+id/springmessage"
              android:layout_width="fill_parent"
android:layout_height="wrap_content"/>
</LinearLayout>
```

要从 Android 设备访问 JAX-RS web 服务，需在 AndroidManifest.xml 中启用 android.permission.INTERNET 权限，这将允许应用程序打开网络套接字。在 清单 6 中添加 uses-permission 元素。

清单 6. 添加 Internet 权限

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

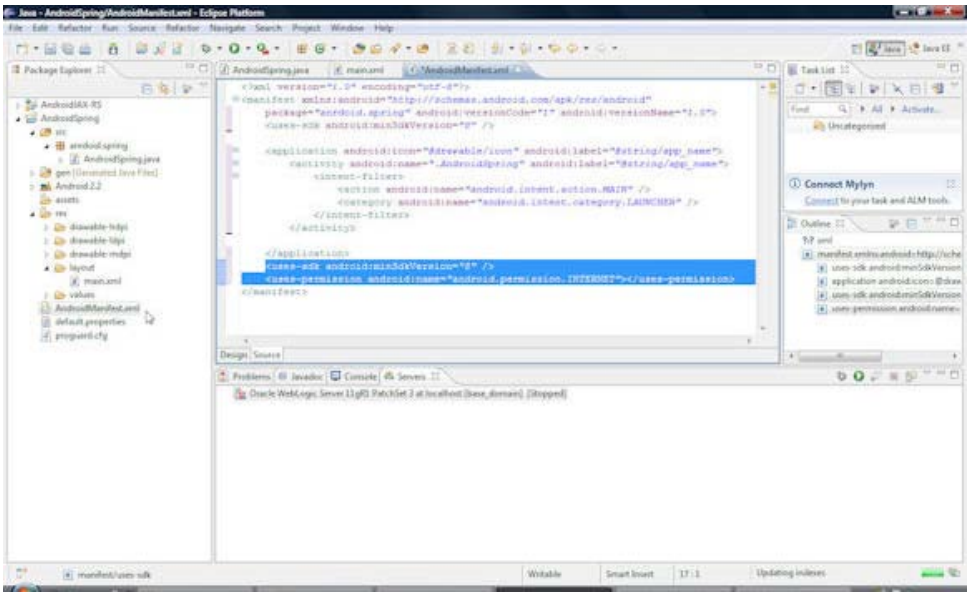
利用 uses-sdk 元素指定最小 Android 版本。AndroidSpring 活动、intent-filter 和 action 用以下元素指定。清单 7 展示了 AndroidManifest.xml 文件。

清单 7. AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="android.spring" android:versionCode="1" android:versionName="1.0">
     <uses-sdk android:minSdkVersion="8" />
     <application android:icon="@drawable/icon" android:label="@string/app_name">
          <activity android:name=".AndroidSpring" android:label="@string/app_name">
               <intent-filter>
                 <action android:name="android.intent.action.MAIN" />
                 <category android:name="android.intent.category.LAUNCHER" />
               </intent-filter>
          </activity>
     </application>
     <uses-sdk android:minSdkVersion="8" />
     <uses-permission
android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

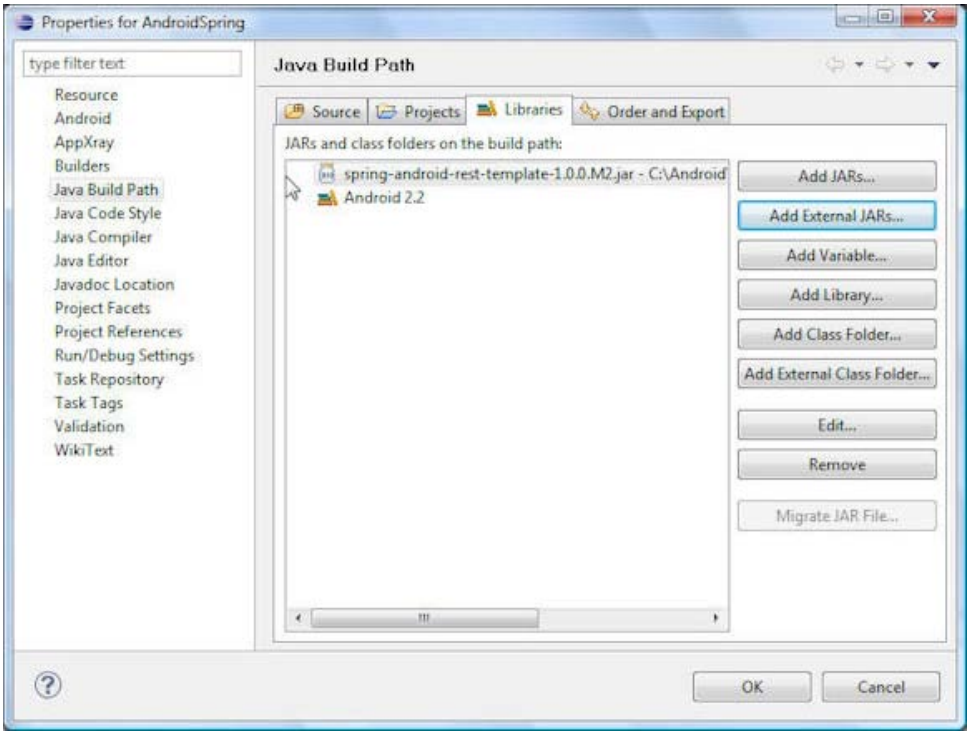
图 6 展示了在 Eclipse IDE 中查看的 AndroidManifest.xml 文件。

图 6. 在 Eclipse IDE 中查看的 AndroidManifest.xml 文件



选择 Java Build Path。在 Libraries 选项卡，将 spring-android-rest-template JAR 文件添加到 Java 构建路径，如 图 7 所示。

图 7. Java 构建路径中的 Spring Android REST 模板 JAR



`org.springframework.web.client.RestTemplate` 执行 RESTful 原则，是客户端 HTTP 访问的中心类。`org.springframework.http` 包包含客户端/服务器端 HTTP 传输的基本抽象。

1. 在 `AndroidSpring` 类中，导入 `RestTemplate` 类和 `org.springframework.http` 包。`AndroidSpring` 类扩展 `Activity` 类。onCreate(Bundle savedInstanceState) 方法在活动首次调用时被调用。
2. 使用 `setContentView` 方法和布局资源定义用户界面。
   ```
   setContentView(R.layout.main);
   ```
3. 在 `main.xml` 中定义的 id 为 "springmessage" 的 `TextView` 元素上，使用 `findViewById` 方法创建一个 Android 小部件 `TextView` 对象。
   ```
   TextView springmessage = (TextView) findViewById(R.id.springmessage);
   ```
4. 创建一个 `HttpHeaders` 对象，它表示 HTTP 请求和响应头。
   ```
   HttpHeaders requestHeaders = new HttpHeaders();
   ```
5. 将主体的媒体类型设置为跟 `Content-Type` 头指定的一样。媒体类型应该匹配 JAX-RS web 服务生成的媒体类型。
   ```
   requestHeaders.setContentType(new MediaType("text","plain"));
   ```
6. 创建一个包含请求头的 HTTP 请求实体。
   ```
   HttpEntity<String> requestEntity = new HttpEntity<String>(requestHeaders);
   ```
7. 使用构造函数，利用默认设置，创建 `RestTemplate` 的一个新实例。
   ```
   RestTemplate restTemplate = new RestTemplate();
   ```
8. 指定到宿主在 URI 路径 `/helloworld` 上的资源的 URL。
   ```
   String url = "http://192.168.1.68:7001/AndroidJAX-RS/jaxrs/helloworld";
   ```
9. 通过使用 `exchange` 方法将请求实体发送到请求，调用到指定 URI 模板的 HTTP 方法。`exchange` 方法返回响应为 `ResponseEntity`。
   ```
   ResponseEntity<String> responseEntity =
   restTemplate.exchange(url, HttpMethod.GET, requestEntity, String.class);
   ```
10. 使用 `getBody` 从 `ResponseEntity` 检索响应字符串。
    ```
    ResponseEntity<String> String result = responseEntity.getBody();
    ```

11. 设置 TextView UI 组件上的字符串消息。

```
springmessage.setText(result);
```

清单 8 展示了 AndroidSpring 类。

清单 8. AndroidSpring 类

```
package anrdoid.spring;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import org.springframework.web.client.RestTemplate;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;


public class AndroidSpring extends Activity {
    /** Called when the activity is first created. */
    @Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView
springmessage = (TextView) findViewById(R.id.springmessage);
      //  RestTemplate restTemplate = new RestTemplate();
      //  String url =

"http://192.168.1.68:7001/AndroidJAX-RS/jaxrs/helloworld";
      //  String result = restTemplate.getForObject(url, String.class);

        HttpHeaders
requestHeaders = new HttpHeaders();
       requestHeaders.setContentType(new MediaType("text","xml"));
       HttpEntity<String> requestEntity = new HttpEntity<String>(requestHeaders);
        String url = "http://192.168.1.68:7001/AndroidJAX-RS/jaxrs/helloworld";
        RestTemplate restTemplate = new RestTemplate();
ResponseEntity<String> responseEntity =
restTemplate.exchange(url,  HttpMethod.GET, requestEntity, String.class);
       String result =
responseEntity.getBody();
        springmessage.setText(result);

    }
}
```

回页首

配置 Maven 插件和依赖项

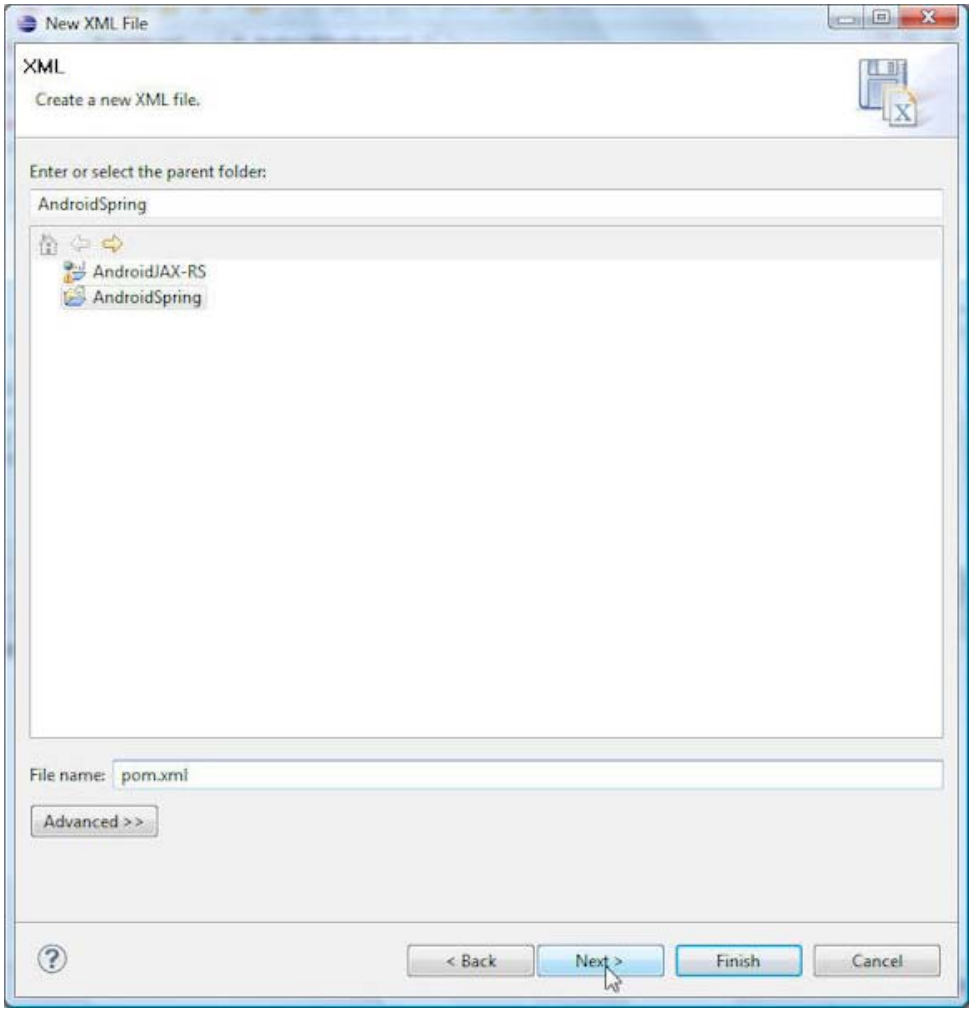
由 Maven 用来构建项目的配置详细信息在 pom.xml 中指定，这个文件中定义了 Project Object Model for Maven。项目依赖项、知识库和插件是 pom.xml 文件中指定的一些配置详细信息。您将在 pom.xml 中配置以下知识库、依赖项和插件。

- Spring Maven 知识库 - 利用 Maven 获得 Spring 3 工件
- Spring Maven Milestone 知识库 - 支持最新 Spring 里程碑的开发
- Maven Android 插件 - 一个用于 Android 的 Maven 插件
- Maven compiler 插件 - 编译项目的源代码
- Google Android 依赖项 - 指定 Google Android 平台上的依赖项
- Spring Android REST Template Module 依赖项 - 指定 spring-android-rest-template 上的依赖项

首先在 AndroidSpring web 项目中创建一个 pom.xml。

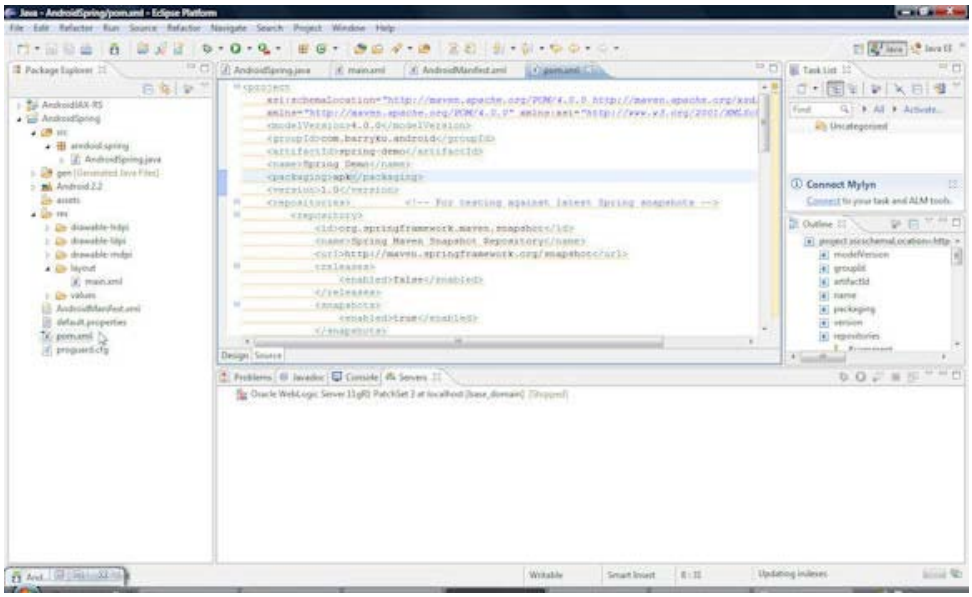
1. 选择 File > New。
2. 在 New 窗口，选择 XML > XML File，并单击 Next。
3. 在 New XML File 向导中，选择 AndroidSpring 文件夹。
4. 将 File Name 指定为 pom.xml，如 图 8 所示。单击 Next。

图 8. 创建 pom.xml



5. 选择 Create XML File from an XML template，并单击 Next。
6. 选择 xml 声明模板，并单击 Finish。
   SpringAndroid 项目现在显示 pom.xml 配置文件，如 图 9 所示。

图 9. pom.xml



配置 前面列出的插件、知识库和依赖项。要指定 Spring Maven Snapshot Repository，需设置以下值（参见 清单 9）：

- 在 <id> 元素中，指定 org.springframework.maven.snapshot
- 在 <url> 元素中，指定 http://maven.springframework.org/snapshot
- 在版本的 enabled 元素中，将值设置为 false
- 在快照的 enabled 元素中，将值设置为 true

清单 9. Spring Maven Snapshot Repository

```
<repository>
     <id>org.springframework.maven.snapshot</id>
     <name>Spring Maven Snapshot Repository</name>
     <url>http://maven.springframework.org/snapshot</url>
          <releases>
               <enabled>false</enabled>
          </releases>
          <snapshots>
               <enabled>true</enabled>
          </snapshots>
</repository>
```

类似地，用以下值配置 Spring Maven Milestone Repository：

- 在 id 元素中，指定 org.springframework.maven.milestone
- 在 releases/enabled 元素中，指定 true
- 在快照的 enabled 元素中，将值设置为 false

利用 清单 10 中的值配置 Maven Android 插件：

- 在 groupId 元素中，指定 com.jayway.maven.plugins.android.generation
- 在 artifactId 元素中，指定 maven-android-plugin
- 在 Maven Android 插件的 <configuration> 元素中，指定 SDK 平台为 8，到 SDK 的路径为 C:/Android/android-sdk
- 在 Maven Android 插件的 <emulator> 元素中，指定将用到的 <avd>。

清单 10. Spring Maven Snapshot Repository

```
<plugin>
     <groupId>com.jayway.maven.plugins.android.generation2</groupId>
          <artifactId>maven-android-plugin</artifactId>
                    <version>2.8.3</version>
                    <configuration>
                         <sdk>
                    <platform>8</platform>
                         <path>C:/Android/android-sdk</path>
                         </sdk>
                         <emulator>
                              <avd>rhoAndroid30</avd>
                         </emulator>
                <deleteConflictingFiles>true</deleteConflictingFiles>
                <undeployBeforeDeploy>true</undeployBeforeDeploy>
                     </configuration>
                     <extensions>true</extensions>
</plugin>
```

在 <dependencies> 元素中，将带有 <artifactId> 的 Google Android 依赖项配置为 android。在带有 <artifactId> 的 Spring Android REST Template Module 上，将 <dependency> 元素配置为 spring-android-rest-template。清单 11 列出了 pom.xml 配置文件。

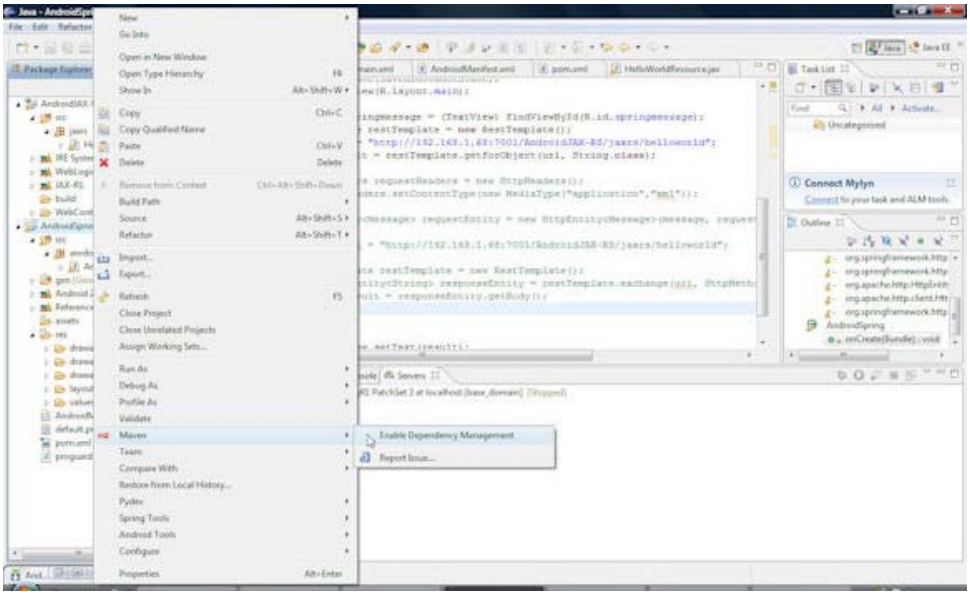
清单 11. pom.xml

```
<project
     xsi:schemaLocation=
     "http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
     xmlns=
     "http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <modelVersion>4.0.0</modelVersion>
     <groupId>anrdoid.spring</groupId>
     <artifactId>spring-demo</artifactId>
     <name>Spring Demo</name>
     <packaging>apk</packaging>
     <version>1.0</version>

     <repositories>
```

```
          <repository>
               <id>org.springframework.maven.snapshot</id>
               <name>Spring Maven Snapshot Repository</name>
               <url>http://maven.springframework.org/snapshot</url>
               <releases>
                    <enabled>false</enabled>
               </releases>
               <snapshots>
                    <enabled>true</enabled>
               </snapshots>
          </repository><!-- For developing against latest Spring milestones -->
          <repository>
               <id>org.springframework.maven.milestone</id>
               <name>Spring Maven Milestone Repository</name>
               <url>http://maven.springframework.org/milestone</url>
               <snapshots>
                    <enabled>false</enabled>
               </snapshots>
          </repository>
     </repositories>
     <build>
          <sourceDirectory>src</sourceDirectory>
          <finalName>${project.artifactId}</finalName>
          <plugins>
               <plugin>
<groupId>com.jayway.maven.plugins.android.generation2</groupId>
               <artifactId>maven-android-plugin</artifactId>
               <version>2.8.3</version>
               <configuration>
                    <sdk>
                         <platform>8</platform>
                         <path>C:/Android/android-sdk</path>
                         </sdk>
                         <emulator>
                              <avd>rhoAndroid30</avd>
                         </emulator>
               <deleteConflictingFiles>true</deleteConflictingFiles>
                         <undeployBeforeDeploy>true</undeployBeforeDeploy>
                    </configuration>
                    <extensions>true</extensions>
               </plugin>
               <plugin>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>2.3.2</version>
               </plugin>
          </plugins>
     </build>
     <dependencies>
          <dependency>
               <groupId>com.google.android</groupId>
               <artifactId>android</artifactId>
               <version>2.2.1</version>
               <scope>provided</scope>
          </dependency>
          <dependency>
               <groupId>org.springframework.android</groupId>
               <artifactId>spring-android-rest-template</artifactId>
               <version>1.0.0.BUILD-SNAPSHOT</version>
          </dependency>
     </dependencies>
</project>
```

需要时，从 `pom.xml` 的 XML 模式（http://maven.apache.org/xsd/maven-4.0.0.xsd）指定额外的依赖项和其他元素。既然已经配置了 Maven Android 插件、Android 依赖项、Spring Android REST Template 依赖项、Maven Integration for Eclipse 插件和 Maven Integration for Android Development Tools，您就可以使用 Maven 在 Eclipse 中利用 Spring 客户端开发 Android 应用程序了。但是，Maven 与 Eclipse 的集成还没有完成。您需要启用依赖项管理，这是由 Maven Integration for Eclipse 插件提供的。右键单击 AndroidSpring 项目，并选择 Maven > Enable Dependency Management。参见 图 10。

图 10. pom.xml



所需的来自 Maven 知识库的 Maven 依赖项和源代码被下载和更新，项目被构建。`target` 文件夹被添加到 `SpringAndroid` 目录。

回页首

配置 Android Maven 目标

Maven 2.0 构建生命周期由不同的构建阶段组成。表 1 列出并描述了默认的构建周期阶段。

表 1. 默认构建周期阶段

阶段	说明
validate	验证项目
compile	编译项目源代码
test	利用单元测试框架测试已编译的源代码
package	打包已编译的代码
integration-test	运行集成测试
verify	验证打包的有效性
install	将打包内容安装到本地知识库中
deploy	在集成和发布环境中，将包复制到远程知识库中

您调用一个构建阶段时，也就调用了所有前面的构建阶段。一个构建阶段包含多个目标，每个目标代表更小的特定任务。一个构建阶段可能与零个或多个目标相关联。如果一个构建阶段没有任何与之绑定的目标，那么此构建阶段就不会运行。目标被利用包和插件分配给构建阶段。在 `pom.xml` 中将包设置为 `apk`：

```
<packaging>apk</packaging>
```

根据指定的包类型，特定的目标被绑定到不同的构建阶段。一些包类型对 pom.xml 中配置的插件可用。apk 包类型对 Maven Android 插件可用。您在 `pom.xml` 中配置了 Maven Android 插件。要使用与 Maven Android 插件关联的包类型，需将 `extensions` 元素设置为 `true`，如 清单 12 所示。

清单 12. `pom.xml`

```
 <plugin>
<groupId>com.jayway.maven.plugins.android.generation2</groupId>
<artifactId>maven-android-plugin</artifactId>
......
<extensions>true</extensions>
</plugin>
```

您也可以通过在 `pom.xml` 中配置插件来添加目标。每个插件都提供一些目标，它们的配置（比如到特定构建阶段的绑定）可以在 `pom.xml` 中配置。利用包类型 `apk`，Maven Android 插件定制默认的 Maven 生命周期，并运行一些额外的任务。 表 2 列出并描述

了这些对默认 Maven 生命周期的定制。

表 2. 对默认 Maven 阶段的定制

Maven 阶段	说明
generate-sources	使用 Android Asset Packaging Tool (AAPT) 打包特定于 Android 的资源，比如 AndroidManifest.xml
process-classes	使用 dx 工具将所有类（库、资源和项目代码）都转换成 davlik 可执行格式
package	为仿真器或设备上的安装，使用 Android 包工具 (apk) 创建 Android 包文件 (Apk)
pre-integration-test	将 Android 包文件 (apk)，包括依赖项在内，都部署到仿真器或设备
integration-test	针对已部署的应用程序，运行插桩（instrumentation）测试类

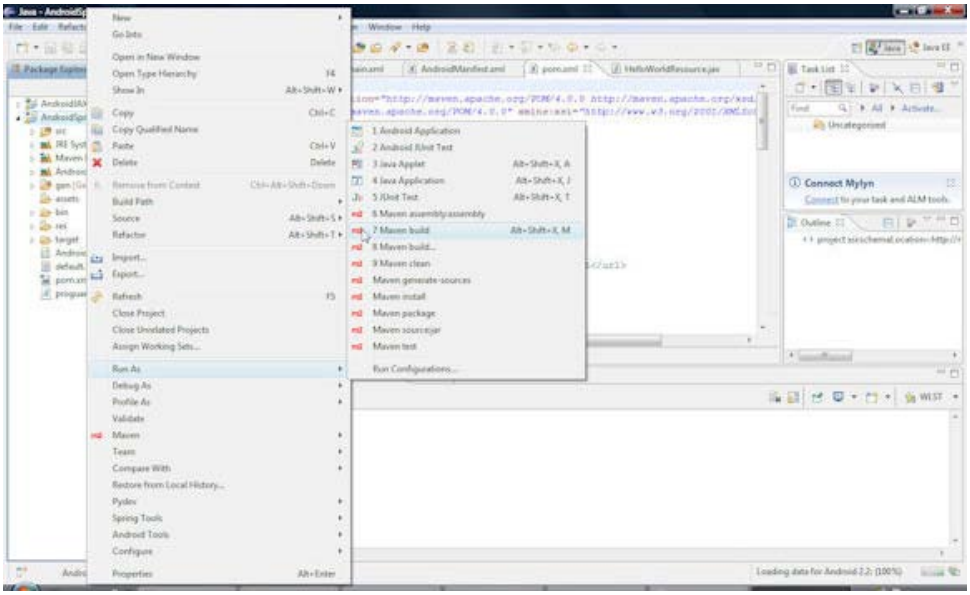
表 3 列出并描述了 Maven Android 插件提供的目标。

表 3. Maven Android 插件目标

目标	说明
android:apk	创建 Android 包文件 (apk)
android:deploy	将构建（或其他）apk 部署到仿真器或设备
android:deploy-dependencies	部署类型 apk 的所有依赖项
android:dex	将 Java 类转换成 Android Dalvik Executable (dex) 格式
android:emulator-start	启动 Android 仿真器。您已经在 `pom.xml` 中为 Maven Android 插件配置了一个仿真器：<emulator><avd>rhoAndroid30</avd></emulator> 。您也可以在 `emulator` 元素中配置启动参数和选项
android:generate-sources	停止 Android 仿真器
install	生成 `R.java` 文件并删除源目录中的任何 R.java。根据 .aidl 文件生成 Java 文件，并删除任何与 `.aidl` 文件同名的 .java 文件
android:instrument	在仿真器/设备上运行插桩 Android 包
android:internal-integration-test	是一个与集成测试阶段关联的内部目标
android:internal-pre-integration-test	是一个与集成测试之前阶段关联的内部目标
android:pull	从仿真器或设备复制文件和目录
android:push	将文件和目录复制到仿真器或设备
android:undeploy	从仿真器或设备解除部署与当前构建项目关联的 apk 或者另一个指定的 apk

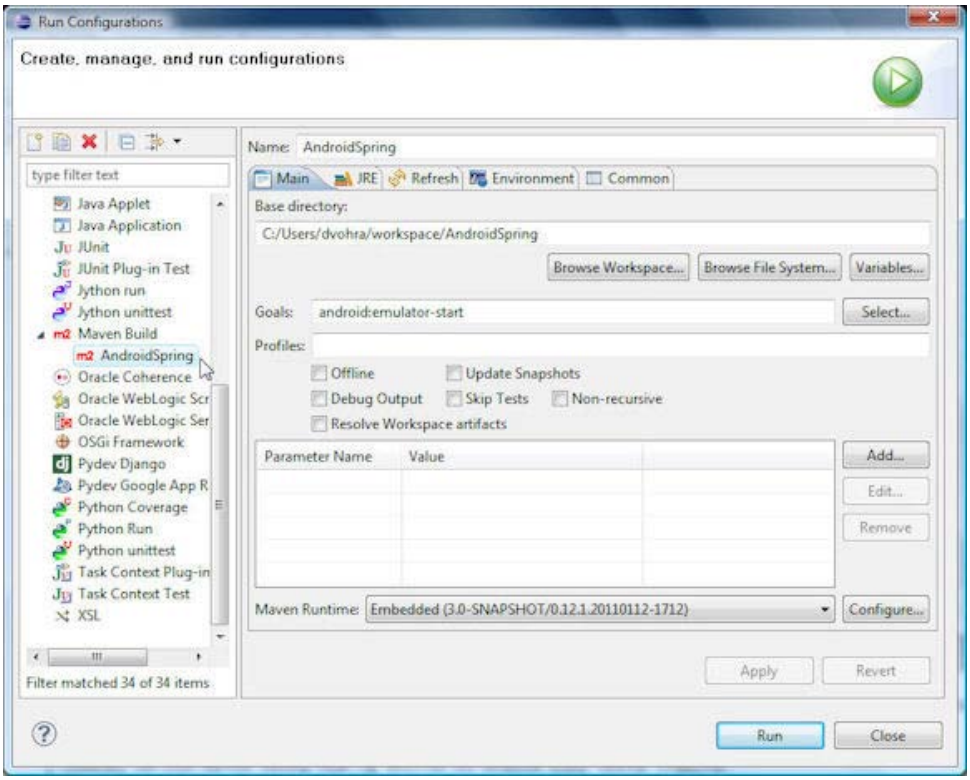
接下来，您从 Maven Android 插件配置一些目标到 Maven 生命周期。右键单击 `AndroidSpring`，并选择 `Run As > Maven build`，如 图 11 所示。

图 11. 配置 Maven 运行配置



在 Maven Build 节点中，为 android:emulator-start 目标添加一个 Run Configuration。指定一个 Run Configuration 名称（AndroidSpring），并在 Goals 中指定 android:emulator-start，如 图 12 所示。Maven Runtime 是 Embedded 3.0-Snapshot。单击 Apply。

图 12. 配置 android:emualtor-start 目标

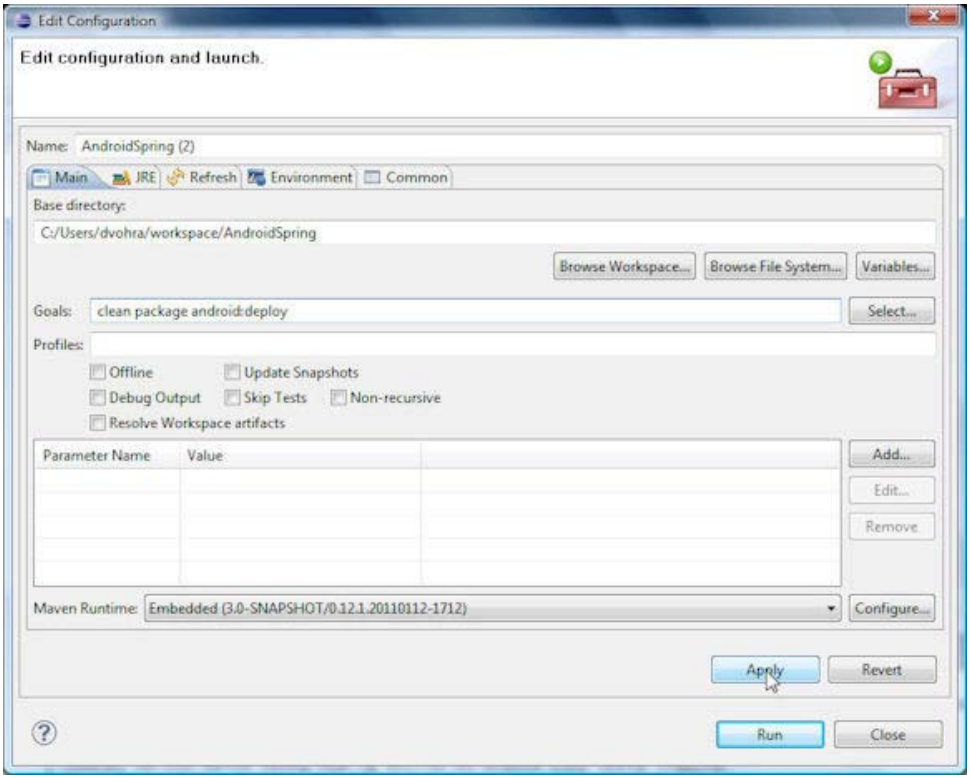


类似地，配置另一个 Run Configuration（AndroidSpring(2)）。在 Goals 中，指定以下 Maven 构建阶段和 Maven Android 插件目标。

```
clean package android:deploy
```

构建阶段（以及每个构建阶段生命周期的任何构建阶段之前阶段）和目标都按指定的顺序被调用。图 13 展示了 Run Configuration AndroidSpring (2)。

图 13. 用于打包和部署 Spring 客户端的 Run configuration

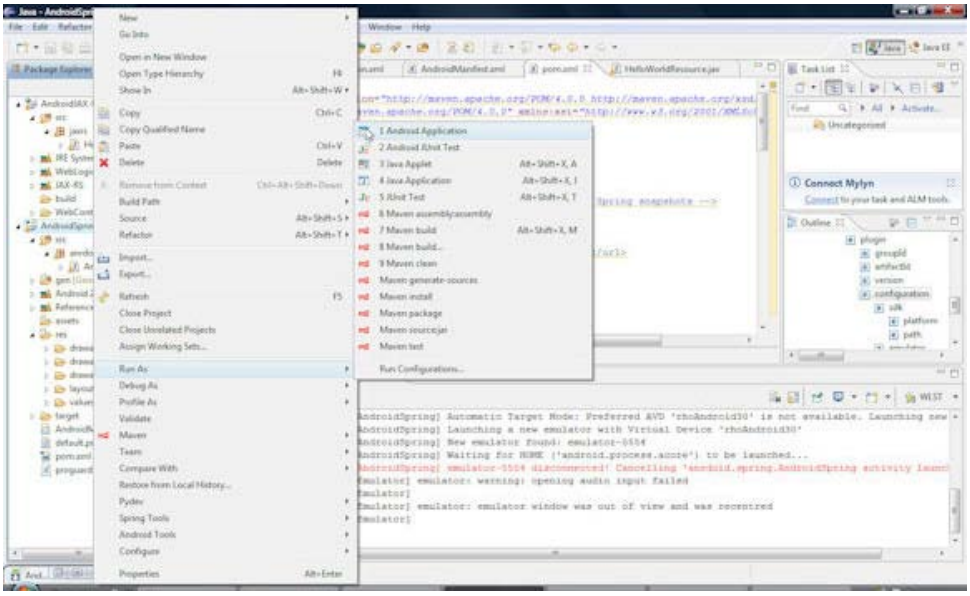


回页首

运行 Spring 客户端 Android 应用程序

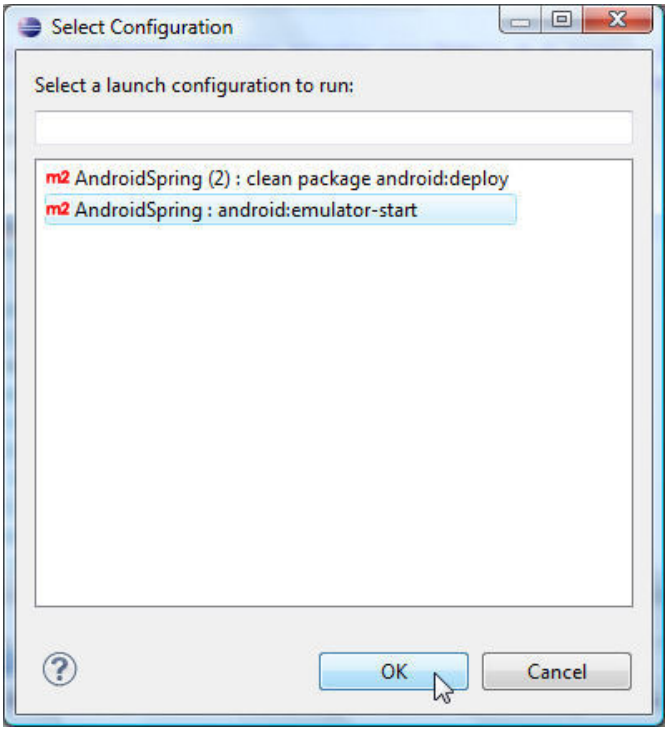
接下来，运行 Android Spring 应用程序。右键单击 AndroidSpring，并选择 Run As > Android Application，如 图 14 所示。

图 14. 运行 Spring Android 应用程序



您配置的 Maven 配置被列出来了。首先，选择此配置以启动 Android 仿真器，如 图 15 所示。单击 OK。

图 15. 启动 Android 仿真器

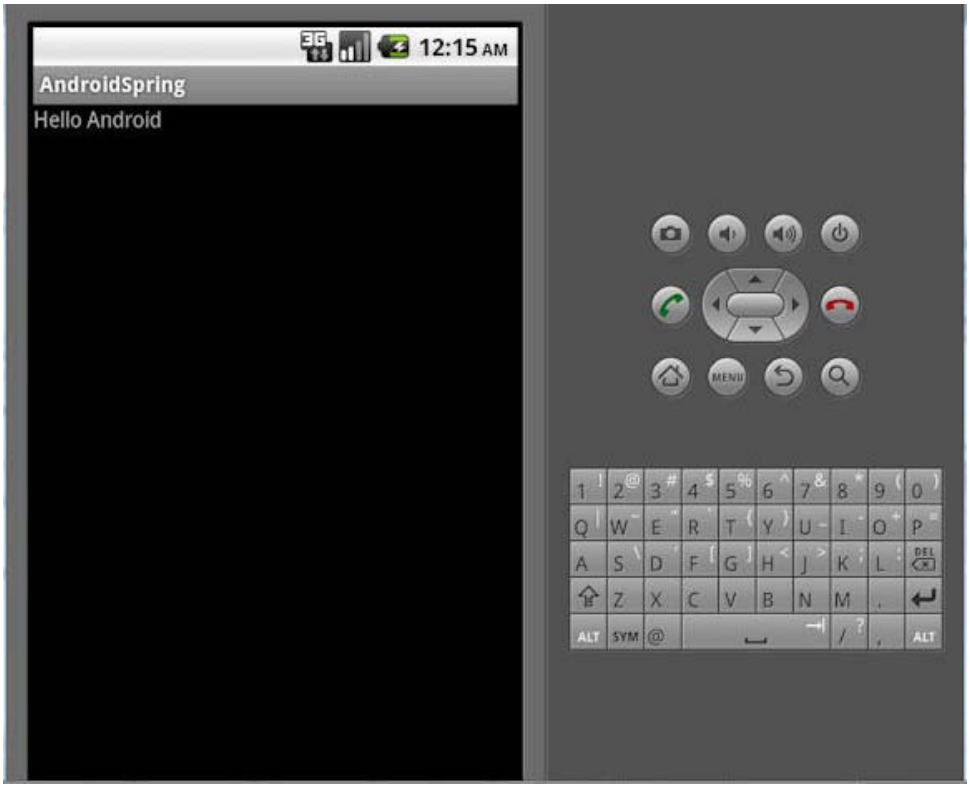


然后，选择此配置以部署 Android apk 文件。`AndroidSpring` 应用程序被打包为一个 apk，并被部署到 Android 仿真器。图 16 展示了 Android 仿真器中的 `AndroidSpring` 应用程序。

图 16. Spring Android 客户端应用程序

单击运行 AndroidSpring 应用程序。JAX-RS web 服务的 Spring 客户端调用此 web 服务，web 服务返回的消息显示在 Android 仿真器中，如 图 17 所示。

图 17. 来自运行 Spring Android 客户端的文本响应



修改 JAX-RS web 服务资源类，以生成一个 `text/xml` 消息，而不是 `text/plain` 消息。参见 清单 13。

清单 13. 生成 `text/xml` 消息

```
@GET
@Produces("text/xml")
      public String getXMLMessage() {
      return "<?xml version=\"1.0\"?>" + "<hello> Hello Android" + "</hello>";
      }
```

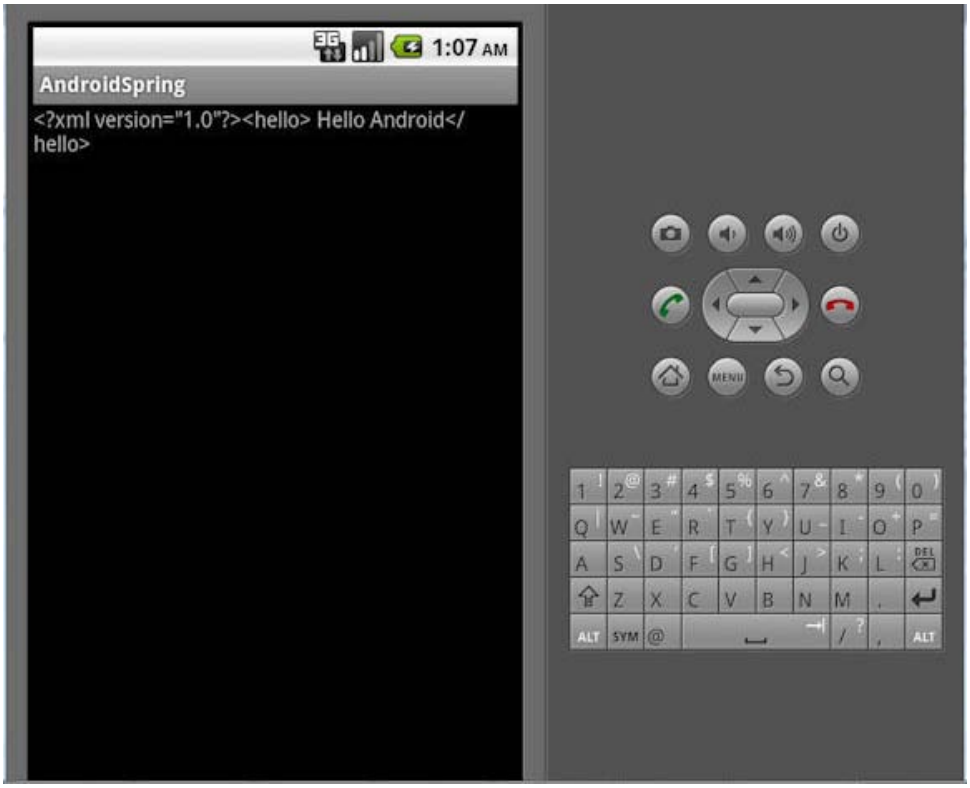
修改 `SpringAndroid` 客户端应用程序，以将请求头内容类型设置为 `text/xml`。参见 清单 14。

清单 14. 将请求头设置为 `text/xml`

```
requestHeaders.setContentType(new MediaType("text","xml"));
```

重新部署 `AndroidJAX-RS` web 服务资源类，并重新部署 `SpringAndroid` 客户端应用程序。在仿真器上运行 SpringAndroid 应用程序，以输出从 JAX-RS web 服务收到的 XML 消息，如 图 18 所示。

图 18. Android 中对 Spring Android 客户端的 `text/xml` 响应



在本文中，您使用 Spring Android 插件为一个 JAX-RS web 服务创建了 Spring 客户端。

回页首

下载

描述	名字	大小	下载方法
Spring Android Client Eclipse Project	android-spring.zip	627KB	HTTP

关于下载方法的信息

参考资料

学习

- Android：了解 Android 工具，阅读这篇关于如何创建 Android 应用程序的文章。
- developerWorks 中国网站 SOA and Web services 技术专区：获得提升您的 web 服务技能所需的项目资源。
- IBM JAX-RS: Developer's Guide：获得关于 IBM JAX-RS 和 WebSphere Application Server 的指南。
- JAX-RS Selection Algorithm：更多地了解用 JAX-RS 符号标注（以表明是一种 web 资源）的 Java 类。
- 本作者的更多文章（Deepak Vohra，developerWorks，2005 年 4 月至今）：阅读关于 Android、Ajax、PHP、XML、web 服务、Ruby on Rails、EJB 和其他技术的文章。
- XML 新手入门 获得学习 XML 所需的资源。
- developerWorks 中国网站 XML 技术专区：在 XML 专区获得提高您的专业技能所需的资源。参见 XML 技术文档库，获得大量技术文章和技巧、教程、标准以及 IBM 红皮书。
- IBM XML 认证：了解如何才能成为一名 IBM 认证的 XML 和相关技术的开发人员。
- developerWorks 技术活动 和 网络广播：随时关注技术的最新进展。
- developerWorks 播客：收听针对软件开发人员的有趣访谈和讨论。

- developerWorks 演示中心：观看演示，包括面向初学者的产品安装和设置演示，以及为经验丰富的开发人员提供的高级功能。

获得产品和技术

- Android SDK：下载这个 SDK，获得工具、样例代码和文档。
- Spring Android：下载并开始在 Android 环境中使用 Spring Framework。
- Eclipse for Java EE：下载用于创建 Java EE 和 web 应用程序的最新版工具，包括一个 Java IDE、一些针对 Java EE、JPA、JSF、Mylyn 等的工具。
- Jersey：下载 Jersey 包和 Jersey 归档文件。
- JDK 6.0：下载 JDK 6.0 二进制文件。
- 试用版: IBM WebSphere Application Server：下载一个 Java EE 5 认证的、基于 EJB 3.0 支持的技术的应用程序平台，它在业内最广范围的平台上，利用您的 SOA 环境的一个创新的、基于性能的基础，驱动业务敏捷性。
- IBM 产品评估试用版软件：下载或 IBM SOA 人员沙箱，并尝试使用来自 DB2®、Lotus®、Rational®、Tivoli® 和 WebSphere® 的应用程序开发工具和中间件产品。

讨论

- XML 专区讨论论坛：参与任何一个 XML 相关讨论。
- developerWorks 中文社区：查看开发人员推动的博客、论坛、组和 wikis，并与其他 developerWorks 用户交流。

关于作者

IBM developerWorks Contributing Author

Deepak Vohra 是一名 Web 开发人员，同时也是一名独立顾问，一名经过 Sun 认证的 Java 程序员和 Web 组件开发人员。Deepak 曾在 ONJava.com、java.net、XML Journal 和 Oracle Magazine 上发表过文章。

为本文评分

评论

回页首

打印此页面　分享此页面　关注 developerWorks

技术主题
AIX and UNIX
IBM i
Information Management
Lotus
Rational
WebSphere
Cloud computing
Java technology
Linux
Open source
SOA and web services
Web development
XML
更多...

查找软件
IBM 产品
评估方式（下载，在线试用，Beta 版，云）
行业
技术讲座

社区
群组
博客
Wiki
文件
使用条款
报告滥用
更多...

关于 developerWorks
反馈意见
在线投稿
投稿指南
网站导航
请求转载内容
相关资源
ISV 资源 (英语)
IBM 教育学院教育培养计划

IBM
解决方案
软件
支持门户
产品文档
红皮书 (英语)
隐私条约
浏览辅助