本章将对现有框架进行优化，并提供一些扩展功能。通过本章的学习，您可以了解到：

- 如何优化 Action 参数；
- 如何实现文件上传功能；
- 如何与 Servlet API 完全解耦；
- 如何实现安全控制框架；

第 1 章

从一个简单的Web 应用开始

如果您已经掌握了 Java 的基础知识，理解了面向对象的基本概念，对 Java Web 开发有过一定的了解，比如会用 Servlet 与 JSP 开发一些简单的应用程序，那么本章的内容就是为这类您准备的。

现在我们打算使用业界最牛的 Java 开发工具 IntelliJ IDEA（简称 IDEA）开发一个简单的 Web 应用，该应用不包含任何业务功能，只是为了熟悉 Web 应用的开发过程与 IDEA 的使用方法。同时，我们也会使用业界最强大的项目构建工具 Maven 与代码版本控制工具 Git，利用这些工具让开发过程更加高效。

正所谓“工欲善其事，必先利其器”，在正式开始设计并开发我们的轻量级 Java Web 框架之前，有必要先掌握以下技能：

- 使用 IDEA 搭建并开发 Java 项目；
- 使用 Maven 自动化构建 Java 项目；
- 使用 Git 管理项目源代码。

假设您已经在 Windows 上安装了 IDEA、Maven、Git 这三个工具，现在要做的就是双击 IDEA 图标来启动它，如图 1-1 所示。

图 1-1　IDEA 启动界面

实际上 IDEA 有两个版本，一个是社区版（开源的个人版），另外一个是完全版（收费的企业版）。个人版提供的功能对于我们而言是基本够用的，可能对于 Web 开发来说有些不太方便，但足以满足我们基本的开发需求。不管使用 IDEA 的哪个版本，本书都是适用的。

下面我们就使用 IDEA 来创建一个基于 Maven 的 Java Web 项目。

如果此时不知道 Maven 是什么，或者听说过但不会使用它，那么强烈建议您通过“附录 A”来了解 Maven 是什么以及它的基本用法。

1.1　使用 IDEA 创建 Maven 项目

1.1.1　创建 IDEA 项目

我们无须单独下载 Maven，更不用将其集成到 IDEA 中，因为 IDEA 默认已经将其整合了。在 IDEA 中创建 Maven 项目非常简单，只需要按照以下步骤进行即可：

（1）单击“Create New Project”按钮，弹出 New Project 对话框。

（2）选择 Maven 选项，单击“Next”按钮。

（3）输入 GroupId（org.smart4j）、ArtifactId（chapter1）、Version（1.0.0），单击“Next”按钮。

（4）输入 Project name（chapter1）、Project location（C:\chapter1），单击“Finish”按钮。

需要说明的是，其中 GroupId 建议为网站域名的倒排方式，以便确保唯一性，类似于 Java 的包名。

说明：本书中出现的 GroupId（org.smart4j）所对应的域名 smart4j.org 是我购买的个人域名，您可使用自己喜欢的任意 GroupId，当然也可以与我保持一致。

按照以上操作，只需片刻之间，IDEA 就创建了一个基于 Maven 的目录结构，如图 1-2 所示。

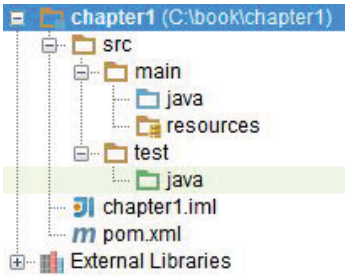

图 1-2　Maven 项目目录结构

1.1.2　调整 Maven 配置

当打开 Maven 项目配置文件（pom.xml）后，看到的应该是这样的配置（经笔者稍作整理）：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns=http://maven.apache.org/POM/4.0.0
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.smart4j</groupId>
    <artifactId>chapter1</artifactId>
    <version>1.0.0</version>

</project>
```

技巧：当调整 pom.xml 后，需单击右上角气泡中的 Import Changes，使 Maven 配置立即生效，此操作表示“手动生效”。也可以单击右上角气泡中的 Enable Auto-Import，一旦对 pom.xml 文件进行修改就会自动导入，此操作表示“自动生效”。不过建议还是使用前者，即“手动生效”方式，这样会更加可控一些，至少能够确定自己做过那些事情。

以上只是 pom.xml 的基本配置，下面需要为它添加一些常用配置。

首先，需要统一源代码的编码方式，否则使用 Maven 编译源代码的时候就会出现相关警告。一般情况下，我们都使用 UTF-8 进行编码，需要添加配置如下：

```
...
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
...
```

除了需要统一源代码编码方式以外，还需要统一源代码与编译输出的 JDK 版本。由于本书中使用 JDK 1.6 进行的开发，因此需要添加如下配置：

```
<build>
    <plugins>
        <!-- Compile -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
```

```
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.3</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

所有的 plugin 都需要添加到 build/plugins 标签下，Maven 插件或下文中所出现的 Maven 依赖都来自于 Maven 中央仓库，可以通过以下链接访问：

```
http://search.maven.org/
```

如果说上面添加的两个配置是必需的，那么下面这个配置应该就是可选的。如果在使用 Maven 打包时想跳过单元测试，那么可以添加如下插件：

```
<!-- Test -->
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-surefire-plugin</artifactId>
   <version>2.18.1</version>
   <configuration>
      <skipTests>true</skipTests>
   </configuration>
</plugin>
```

至此，一个 Maven 项目就搭建完毕了，下面我们就要在这个项目中添加有关 Java Web 的代码。

1.2　搭建 Web 项目框架

1.2.1　转为 Java Web 项目

目前，在 pom.xml 中还没有任何的 Maven 依赖（dependency），随后会添加一些 Java Web 所需的依赖，只不过在添加这些依赖之前，有必要先将这个 Maven 项目调整为 Web 项目结构。

提示：可以简单地将 Maven 依赖理解为 jar 包，只不过 Maven 依赖具备传递性，只需配置某个依赖，就能自动获取该依赖的相关依赖。

只需按照以下三步即可实现：

（1）在 main 目录下，添加 webapp 目录。

（2）在 webapp 目录下，添加 WEB-INF 目录。

（3）在 WEB-INF 目录下，添加 web.xml 文件。

此时，IDEA 给出一个提示，如图 1-3 所示。

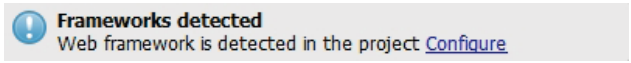

图 1-3 IDEA 框架检测提示

表示 IDEA 已经识别出目前我们使用了 Web 框架（即 Servlet 框架），需要进行一些配置才能使用。单击“Configure”按钮，会看到一个确认框，单击“OK”按钮即可将当前项目变为 Web 项目。

这里打算使用 Servlet 3.0 框架，所以在 web.xml 中添加如下代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        version="3.0">
</web-app>
```

实际上，使用 Servlet 3.0 框架是可以省略 web.xml 文件的，因为 Servlet 无须在 web.xml 里配置，只需通过 Java 注解的方式来配置即可，下面会描述具体的用法。

1.2.2 添加 Java Web 的 Maven 依赖

由于 Web 项目是需要打 war 包的，因此必须在 pom.xml 文件里设置 packaging 为 war（默认为 jar），配置如下：

```
<packaging>war</packaging>
```

接下来就需要添加 Java Web 所需的 Servlet、JSP、JSTL 等依赖了，添加如下配置：

```
<dependencies>
    <!-- Servlet -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
    <!-- JSP -->
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.2</version>
        <scope>provided</scope>
    </dependency>
    <!-- JSTL -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
        <scope>runtime</scope>
    </dependency>
<dependencies>
```

需要说明的是：

（1）Maven 依赖的“三坐标”（groupId、artifactId、version）必须提供。

（2）如果某些依赖只需参与编译，而无须参与打包（例如，Tomcat 自带了 Servlet 与 JSP 所对应的 jar 包），可将其 scope 设置为 provided。

（3）如果某些依赖只是运行时需要，但无须参与编译（例如，JSTL 的 jar 包），可将其 scope 设置为 runtime。

为了便于阅读，以下是 pom.xml 的完整代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.smart4j</groupId>
    <artifactId>chapter1</artifactId>
    <version>1.0.0</version>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <!-- Servlet -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
        <!-- JSP -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.2</version>
            <scope>provided</scope>
        </dependency>
        <!-- JSTL -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

    <build>
```