第 1 章 Spring 基础

学习目的与要求

本章重点讲解 Spring 的基础知识。通过本章的学习，要求读者了解 Spring 的体系结构，理解 Spring IoC 与 AOP 的基本原理，了解 Spring Bean 的生命周期、实例化以及作用域。

本章主要内容

- Spring 开发环境的构建
- Spring IoC
- Spring AOP
- Spring Bean

Spring 是当前主流的 Java 开发框架，为企业级应用开发提供了丰富的功能，掌握 Spring 框架的使用已经是 Java 开发者必须具备的技能之一。本章将学习如何使用 IntelliJ IDEA 开发 Spring 程序，注意在此之前需要构建 Spring 的开发环境。

1.1 Spring 概述

1.1.1 Spring 的由来

Spring 是一个轻量级 Java 企业级应用程序开发框架，最早由 Rod Johnson 创建，目的是解决企业级应用开发的业务逻辑层和其他各层的耦合问题。它是一个分层的 Java SE/EE full-stack（一站式）轻量级开源框架，为开发 Java 应用程序提供全面的基础架构支持。Spring 负责基础架构，因此 Java 开发者可以专注于应用程序的开发。

Spring Framework 6.0 于 2022 年 11 月正式发布，这是 2023 年及以后新一代框架的开始，包含 OpenJDK 和 Java 生态系统中当前和即将到来的创新。Spring Framework 6.0 作为重大更新，要求使用 Java 17 或更高版本，并且已迁移到 Jakarta EE 9+（在 jakarta 命名空间中取代了以前基于 javax 的 API），以及对其他基础设施的修改。基于这些变化，Spring Framework 6.0 支持最新的 Web 容器，例如 Tomcat 10，以及最新的持久性框架 Hibernate ORM 6.1。这些特性仅可用于 Servlet API 和 JPA 的 jakarta 命名空间变体。

在基础架构方面，Spring Framework 6.0 引入了 Ahead-Of-Time（AOT）转换的基础以及对 Spring 应用程序上下文的 AOT 转换和相应的 AOT 处理支持的基础，能够事先将应用程序中或 JDK 中的字节码编译成机器码。在 Spring Framework 6.0 中还有许多新功能和改进可用，例如 HTTP 接口客户端、对 RFC 7807 问题详细信息的支持以及 HTTP 客户端的基于 Micrometer 的可观察性。

1.1.2 Spring 的体系结构

Spring 的功能模块被有组织地分散到约 20 个模块中，这些模块分布在核心容器（Core Container）层、数据访问/集成（Data Access/Integration）层、Web 层以及面向切面编程（Aspect-Oriented Programming，AOP）模块、植入（Instrumentation）模块、消息传输（Messaging）和测试（Test）模块中，如图 1.1 所示。

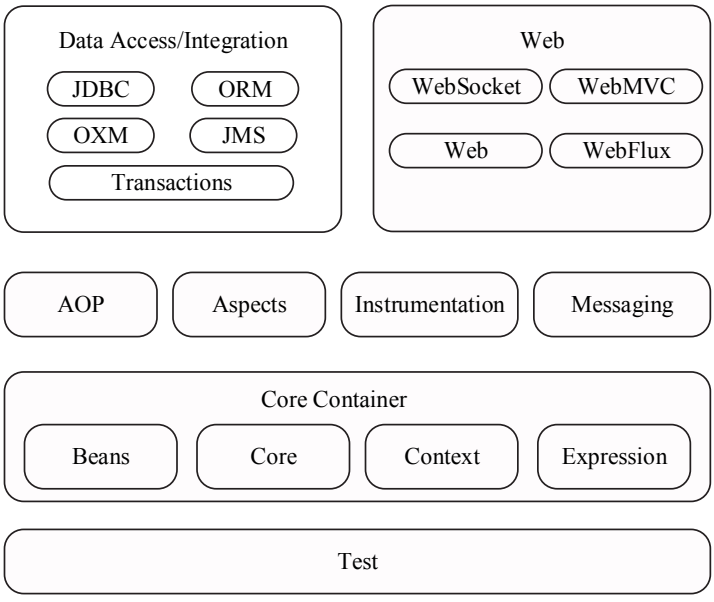

图 1.1 Spring 的体系结构

❶ Core Container

Spring 的 Core Container 是其他模块建立的基础，由 Beans（spring-beans）、Core（spring-core）、Context（spring-context）和 Expression（spring-expression，Spring 表达式语言）等模块组成。

spring-beans 模块：提供了 BeanFactory，是工厂模式的一个经典实现，Spring 将管理对象称为 Bean。

spring-core 模块：提供了框架的基本组成部分，包括控制反转（Inversion of Control，IoC）和依赖注入（Dependency Injection，DI）功能。

spring-context 模块：建立在 spring-beans 和 spring-core 模块的基础之上，提供一个框架式的对象访问方式，是访问定义和配置的任何对象媒介。

spring-expression 模块：提供了强大的表达式语言去支持运行时查询和操作对象图。这是对 JSP 2.1 规范中规定的统一表达式语言（Unified EL）的扩展。该语言支持设置和获取属性值、属性分配、方法调用、访问数组、集合和索引器的内容、逻辑和算术运算、变量命名以及从 Spring 的 IoC 容器中以名称检索对象。它还支持列表投影、选择以及常见的列表聚合。

❷ AOP 和 Instrumentation

Spring 框架中与 AOP 和 Instrumentation 相关的模块有 AOP（spring-aop）模块、Aspects（spring-aspects）模块以及 Instrumentation（spring-instrument）模块。

spring-aop 模块：提供了一个符合 AOP 要求的面向切面的编程实现，允许定义方法拦截器和切入点，将代码按照功能进行分离，以便干净地解耦。

spring-aspects 模块：提供了与 AspectJ 的集成功能，AspectJ 是一个功能强大且成熟的 AOP 框架。

spring-instrument 模块：提供了类植入（Instrumentation）支持和类加载器的实现，可以在特定的应用服务器中使用。Instrumentation 提供了一种虚拟机级别支持的 AOP 实现方式，使得开发者无须对 JDK 做任何升级和改动就可以实现某些 AOP 的功能。

❸ Messaging

Spring 4.0 以后新增了 Messaging（spring-messaging）模块，该模块提供了对消息传递体系结构和协议的支持。

❹ Data Access/Integration

数据访问/集成层由 JDBC（spring-jdbc）、ORM（spring-orm）、OXM（spring-oxm）、JMS（spring-jms）和 Transactions（spring-tx）模块组成。

spring-jdbc 模块：提供了一个 JDBC 的抽象层，消除了烦琐的 JDBC 编码和数据库厂商特有的错误代码解析。

spring-orm 模块：为流行的对象-关系映射（Object-Relational Mapping）API 提供集成层，包括 JPA 和 Hibernate。使用 spring-orm 模块可以将这些 O/R 映射框架与 Spring 提供的所有其他功能结合使用，例如声明式事务管理功能。

spring-oxm 模块：提供了一个支持对象/XML 映射的抽象层实现，例如 JAXB、Castor、JiBX 和 XStream。

spring-jms 模块（Java Messaging Service）：指 Java 消息传递服务，包含用于生产和使用消息的功能。自 Spring 4.1 以后，提供了与 spring-messaging 模块的集成。

spring-tx 模块（事务模块）：支持用于实现特殊接口和所有 POJO（普通 Java 对象）类的编程和声明式事务管理。

❺ Web

Web 层由 Web（spring-web）、WebMVC（spring-webmvc）、WebSocket（spring-websocket）和 WebFlux（spring-webflux）模块组成。

spring-web 模块：提供了基本的 Web 开发集成功能。例如多文件上传功能、使用 Servlet 监听器初始化一个 IoC 容器以及 Web 应用上下文。

spring-webmvc 模块：也称为 Web-Servlet 模块，包含用于 Web 应用程序的 Spring MVC 和 REST Web Services 实现。Spring MVC 框架提供了领域模型代码和 Web 表单之间的清晰分离，并与 Spring Framework 的所有其他功能集成。本书后续章节将会详细讲解 Spring MVC 框架。

spring-websocket 模块：Spring 4.0 以后新增的模块，它提供了 WebSocket 和 SockJS 的实现，主要是与 Web 前端的全双工通信的协议。

spring-webflux 模块：一个新的非堵塞函数式 Reactive Web 框架，可以用来建立异步的、非阻塞、事件驱动的服务，并且扩展性非常好（该模块是 Spring 5 新增的模块）。

❻ Test

Test（spring-test）模块支持使用 JUnit 或 TestNG 对 Spring 组件进行单元测试和集成测试。

扫一扫

视频讲解

1.2 Spring 开发环境的构建

在使用 Spring 框架开发 Spring 应用之前，应该先搭建其开发环境。

1.2.1 配置 IntelliJ IDEA 的 Web 服务器

如果用户的计算机上已经安装了 IntelliJ IDEA（本书使用的是 ideaIU-2022.2.1），那么可以使用 IntelliJ IDEA 便捷地构建 Java Web 应用。虽然 IntelliJ IDEA 自带了 OpenJDK，但是建议在使用之前先安装并配置 JDK 和 Web 服务器。其具体步骤如下：

❶ 安装 JDK

安装并配置 JDK（本书采用的 JDK 是 jdk-18_windows-x64_bin.exe），在按照提示安装完成 JDK 以后，需要配置“环境变量”的“系统变量”Java_Home 和 Path。在 Windows 10 系统下，系统变量示例如图 1.2 和图 1.3 所示。

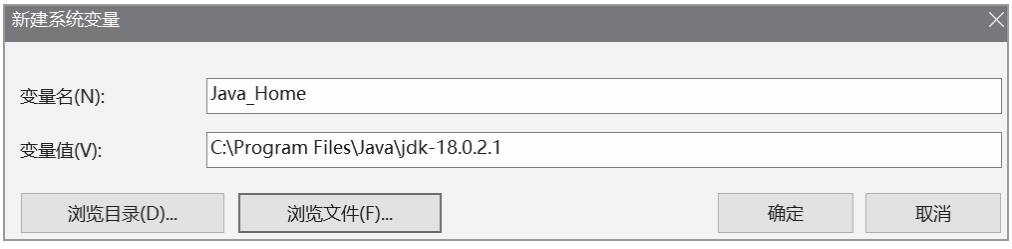

图 1.2 新建系统变量 Java_Home

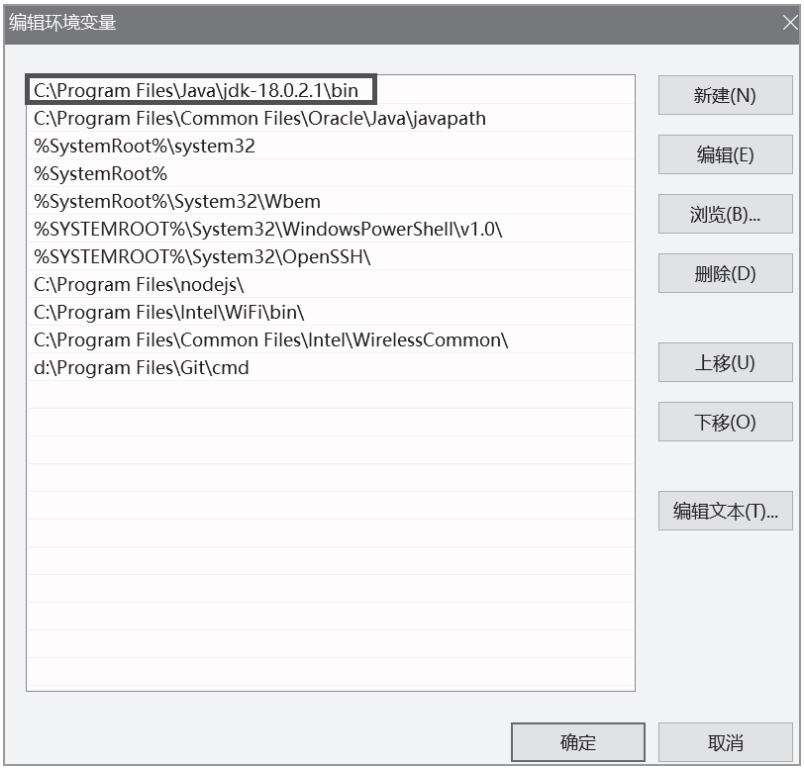

图 1.3 编辑系统变量的环境变量 Path

❷ Web 服务器

目前，比较常用的 Web 服务器有 Tomcat、JRun、Resin、WebSphere、WebLogic 等，本书采用的是 Tomcat 10.0。

登录 Apache 软件基金会的官方网站（http://jakarta.Apache.org/tomcat），下载 Tomcat 10.0 的免安装版（本书采用 apache-tomcat-10.0.23-windows-x64.zip）。登录网站之后，首先在 Download 中选择 Tomcat 10，然后在 Binary Distributions 的 Core 中选择相应版本即可。

在安装 Tomcat 之前需要先安装 JDK 并配置系统环境变量 Java_Home。将下载的 apache-tomcat-10.0.23-windows-x64.zip 解压到某个目录下，解压缩后将出现如图 1.4 所示的目录结构。

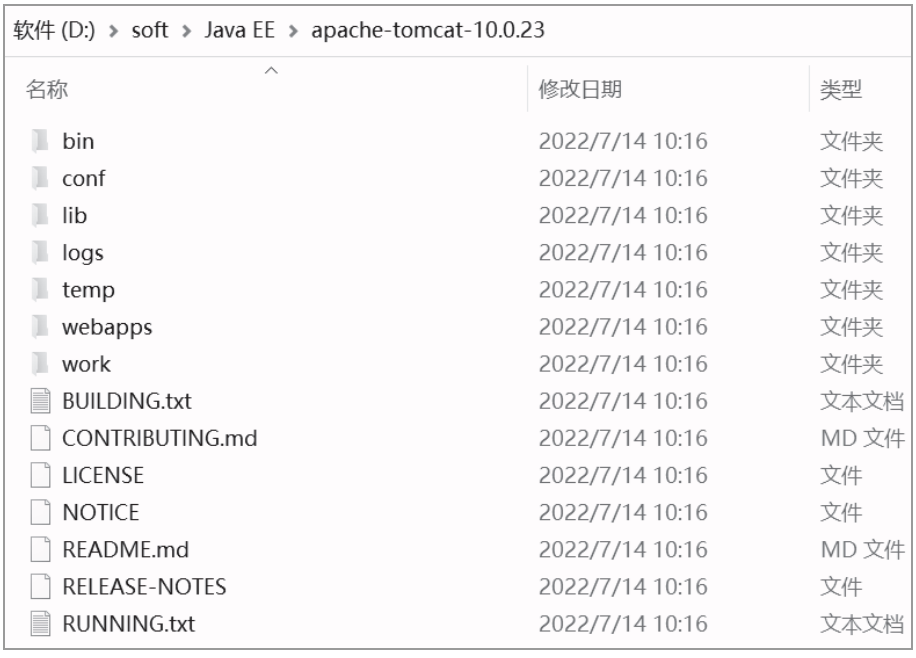

图 1.4　Tomcat 目录结构

通过执行 Tomcat 根目录下 bin 文件夹中的 startup.bat 来启动 Tomcat 服务器。执行 startup.bat 启动 Tomcat 服务器会占用一个 MS-DOS 窗口，如果关闭当前 MS-DOS 窗口将关闭 Tomcat 服务器。

Tomcat 服务器启动后，在浏览器的地址栏中输入“http://localhost:8080”，将出现如图 1.5 所示的 Tomcat 测试页面。

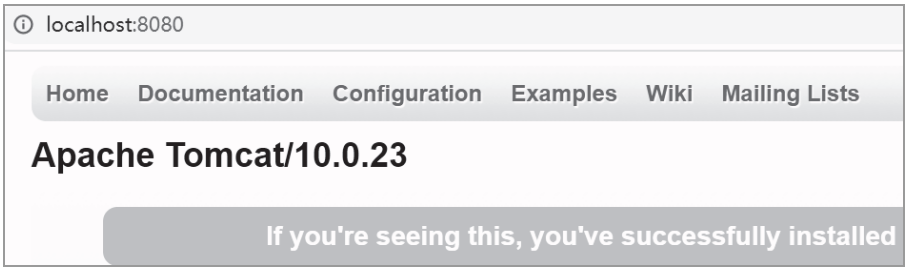

图 1.5　Tomcat 测试页面

❸ 集成 Tomcat

启动 IntelliJ IDEA，选择 File→Settings 命令，在弹出的对话框中选择 Build, Execution, Deployment 下的 Application Servers 选项，然后在右侧单击+按钮，在弹出的菜单中选择 Tomcat Server 命令，打开如图 1.6 所示的 Tomcat Server 界面，在其中选择 Tomcat 目录。

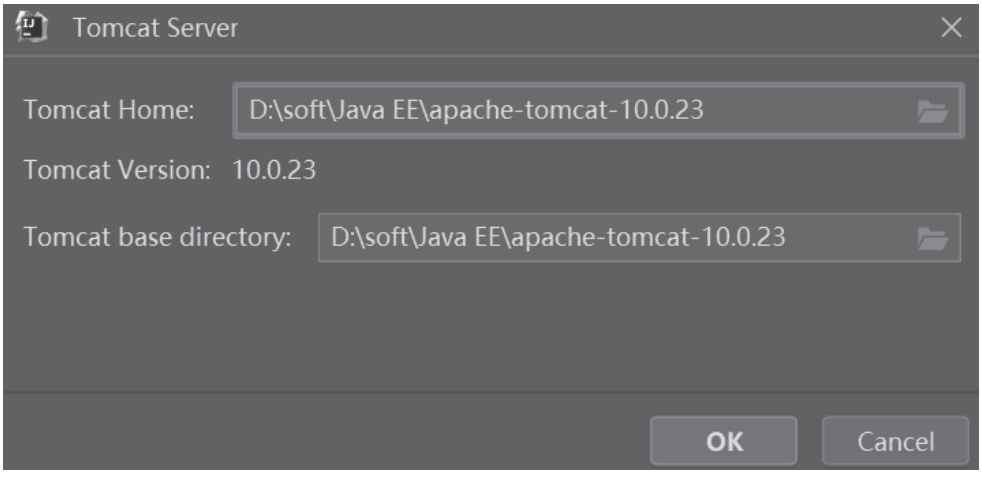

图 1.6　Tomcat 配置界面

在图 1.6 中单击 OK 按钮即可完成 Tomcat 配置。

至此可以使用 IntelliJ IDEA 创建 Java Web 应用，并在 Tomcat 下运行。

▶ 1.2.2　Spring 的下载

在使用 Spring 框架开发应用程序时需要引用 Spring 框架自身的 JAR 包。Spring Framework 6.0.0 的 JAR 包可以从 Maven 中央库获得。

在 Spring 的 JAR 包中有 4 个基础包 spring-core-6.0.0.jar、spring-beans-6.0.0.jar、spring-context-6.0.0.jar 和 spring-expression-6.0.0.jar，分别对应 Spring 核心容器的 4 个模块 spring-core 模块、spring-beans 模块、spring-context 模块和 spring-expression 模块。

对于 Spring 框架的初学者，在开发 Spring 应用时只需要将 Spring 的 4 个基础包和 Spring Commons Logging Bridge 对应的 JAR 包 spring-jcl-6.0.0.jar 复制到 Web 应用的 WEB-INF/lib 目录下即可。

▶ 1.2.3　第一个 Spring 入门程序

本小节通过一个简单的入门程序向读者演示 Spring 框架的使用过程，具体见例 1-1。

【例 1-1】一个简单的入门程序。

其具体实现步骤如下：

❶ 使用 IntelliJ IDEA 创建模块并导入 JAR 包

首先使用 IntelliJ IDEA 创建一个名为 ch1 的项目，如图 1.7 所示。

然后在项目 ch1 中创建一个名为 ch1_1 的模块，如图 1.8 所示。

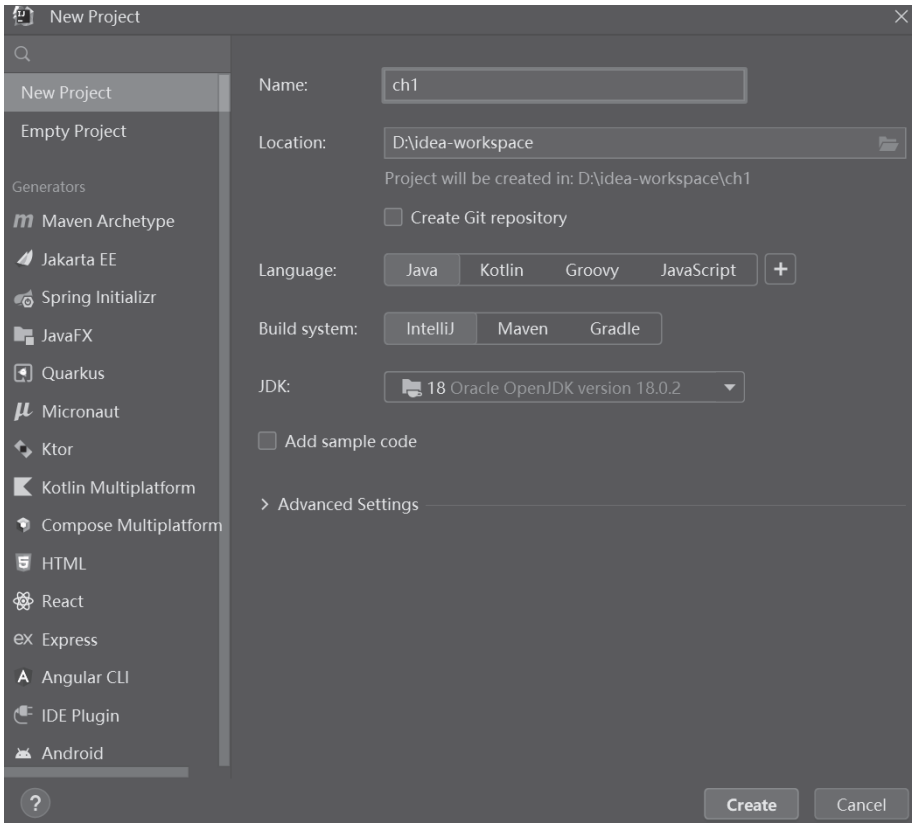

图 1.7　使用 IntelliJ IDEA 创建项目

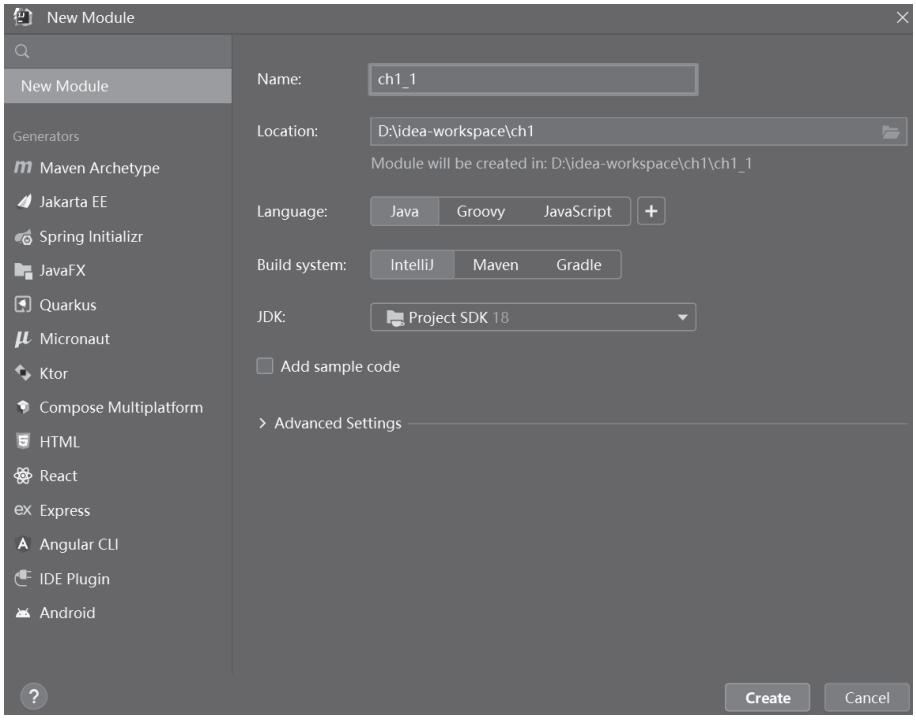

图 1.8　使用 IntelliJ IDEA 创建模块

接着右击模块名 ch1_1，在弹出的快捷菜单中选择 Add Framework Support 命令，给模块 ch1_1 添加 Web Application，如图 1.9 所示。

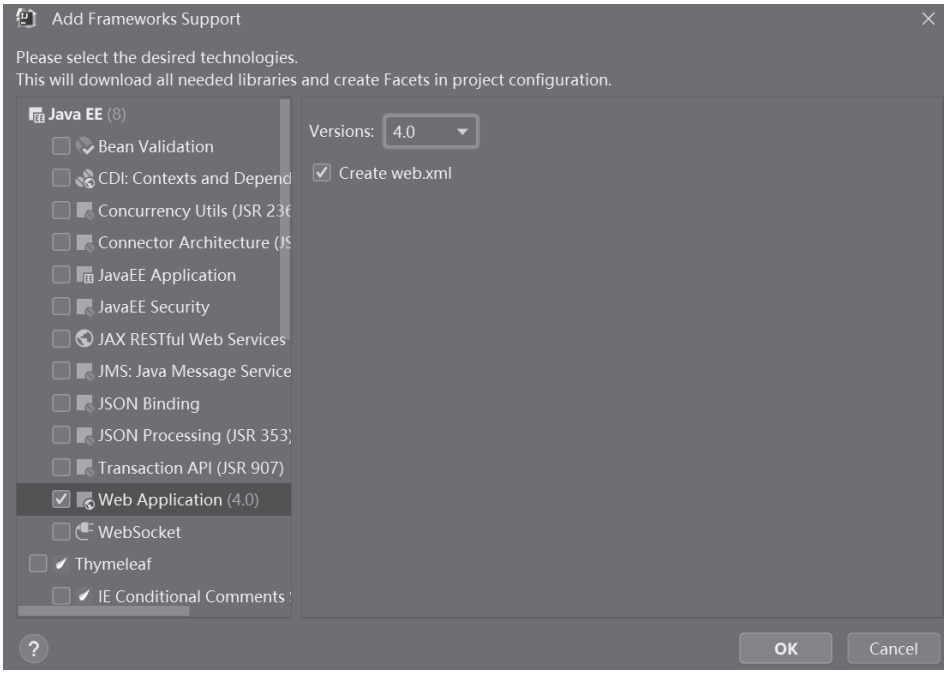

图 1.9　给模块添加 Web Application

最后将 Spring 的 4 个基础包和 Spring Commons Logging Bridge 对应的 JAR 包 spring-jcl-6.0.0.jar 复制到 ch1_1 的 WEB-INF/lib 目录中。具体做法是在 WEB-INF 目录下创建 lib 目录，将 JAR 包复制到 lib 中，然后右击选择 Add as Library 命令，添加为模块依赖，如图 1.10 所示。

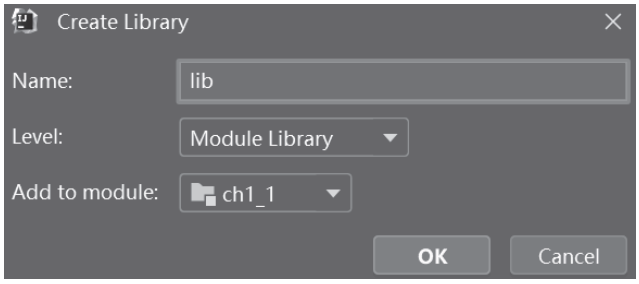

图 1.10　给模块添加依赖

注意：在讲解 Spring MVC 框架之前，本书的实例并没有真正运行 Java Web 应用。创建 Java Web 应用是为了方便添加相关 JAR 包。

❷ 创建接口 TestDao

Spring 解决的是业务逻辑层和其他各层的耦合问题，因此它将面向接口的编程思想贯穿于整个系统应用。

在模块 ch1_1 的 src 目录下创建一个名为 dao 的包，并在 dao 包中创建接口 TestDao，在

该接口中定义一个 sayHello()方法，代码如下：

```
package dao;
public interface TestDao {
    public void sayHello();
}
```

❸ 创建接口 TestDao 的实现类 TestDaoImpl

在 dao 包下创建接口 TestDao 的实现类 TestDaoImpl，代码如下：

```
package dao;
public class TestDaoImpl implements TestDao{
    @Override
    public void sayHello() {
        System.out.println("Hello, Study hard!");
    }
}
```

❹ 创建配置文件 applicationContext.xml

在模块 ch1_1 的 src 目录下创建 Spring 的配置文件 applicationContext.xml，并在该文件中使用实现类 TestDaoImpl 创建一个 id 为 test 的 Bean，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!--将指定类 TestDaoImpl 配置给 Spring，让 Spring 创建其实例-->
    <bean id="test" class="dao.TestDaoImpl" />
</beans>
```

注意：配置文件的名称可以自定义，但习惯上命名为 applicationContext.xml，有时候也命名为 beans.xml。有关 Bean 的创建将在本书后续章节详细讲解，在这里读者只需了解即可。另外，配置文件信息不需要读者手写，可以从 Spring 的帮助文档中复制（首先使用浏览器打开“https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core”，在 1.2.1Configuration Metadata 小节下即可找到配置文件的约束信息）。

❺ 创建测试类

在模块 ch1_1 的 src 目录下创建一个名为 test 的包，并在 test 包中创建 Test 类，代码如下：

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.TestDao;
public class Test {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        appCon=new ClassPathXmlApplicationContext("applicationContext.xml");
        //从容器中获取 test 实例
        TestDao tt = appCon.getBean("test", TestDao.class);
        //test 为配置文件中的 id
        tt.sayHello();
    }
}
```

执行上述 main 方法以后，将在控制台中输出“Hello, Study hard!”。在上述 main 方法中并没有使用 new 运算符创建 TestDaoImpl 类的对象，而是通过 Spring IoC 容器来获取实现类的对象，这就是 Spring IoC 的工作机制。

扫一扫

视频讲解

1.3 Spring IoC

1.3.1 Spring IoC 的基本概念

控制反转（Inversion of Control，IoC）是一个比较抽象的概念，是 Spring 框架的核心，用来消减计算机程序的耦合问题。依赖注入（Dependency Injection，DI）是 IoC 的另外一种说法，只是从不同的角度描述相同的概念。下面通过实际生活中的一个例子解释 IoC 和 DI。

当人们需要一件东西时，第一反应就是找东西，比如想吃面包，在没有面包店时，最直接的做法可能是按照自己的口味制作面包，也就是需要主动制作面包。然而，现在各种网店、实体店盛行，已经没有必要自己制作面包，想吃面包了，去网店或实体店购买即可。注意这里自己并没有制作面包，而是由店家制作，但是完全符合自己的口味。

上面只是列举了一个非常简单的例子，但包含了控制反转的思想，即把制作面包的主动权交给店家。下面通过面向对象编程思想继续探讨这两个概念。

当某个 Java 对象（调用者，比如自己）需要调用另一个 Java 对象（被调用者，即被依赖对象，比如面包）时，在传统编程模式下，调用者通常会采用“new 被调用者”的代码方式来创建对象（比如自己制作面包）。这种方式会增加调用者与被调用者之间的耦合性，不利于后期代码的升级与维护。

当 Spring 框架出现以后，对象的实例不再由调用者来创建，而是由 Spring 容器（比如面包店）来创建。Spring 容器会负责控制程序之间的关系（比如面包店负责控制顾客与面包的关系），而不是由调用者的程序代码直接控制，这样控制权由调用者转移到 Spring 容器，控制权发生了反转，这就是 Spring 的控制反转。

从 Spring 容器角度来看，Spring 容器负责将被依赖对象赋值给调用者的成员变量，相当于为调用者注入它所依赖的实例，这就是 Spring 的依赖注入，主要目的是解耦，体现一种“组合”的理念。

综上所述，控制反转是一种通过描述（在 Spring 中可以是 XML 或注解）并通过第三方去产生或获取特定对象的方式。在 Spring 中实现控制反转的是 IoC 容器，其实现方法是依赖注入。

1.3.2 Spring 的常用注解

在 Spring 框架中，尽管使用 XML 配置文件可以很简单地装配 Bean，但是如果应用中有大量的 Bean 需要装配，会导致 XML 配置文件过于庞大，不方便以后的升级与维护，因此更多的时候推荐开发者使用注解（annotation）的方式去装配 Bean。

需要注意的是，基于注解的装配需要使用<context:component-scan>元素或@ComponentScan 注解定义包（注解所在的包）扫描的规则，然后根据定义的规则找出哪些类（Bean）需要自动装配到 Spring 容器中，再交由 Spring 进行统一管理。

Spring 框架基于 AOP 编程（面向切面编程）实现注解解析，因此在使用注解编程时需要导入 spring-aop-6.0.0.jar 包。

在 Spring 框架中定义了一系列的注解，常用注解如下。

❶ 声明 Bean 的注解

1）@Component

该注解是一个泛化的概念，仅表示一个组件对象（Bean），可以作用在任何层次上，没有明确的角色。