1. Java平台模块系统
2. 概述

封装是面向对象编程的一个特性，类的声明由公有接口和私有实现构成，类可以通过只修改实现而不影响其用户的方式得以演化。模块系统使类和包可以有选择性地获取，从而可以控制模块的演化。多个现有的Java模块系统都依赖于类加载器来实现类之间的隔离，Java9引入了一个由Java编译器和虚拟机支持的新系统，称为Java平台模块系统。

1. 模块

包是类的集合，包提供了一种封装级别，具有包访问权限的所有特性都只能被同一个包中的方法访问，但是在大型系统中，这种级别的访问能控制依然不够，对于所有公有特性可以从任何位置访问，加入我们想要修改或剔除一个很少使用的特性，那么我们无从得知有哪些类依赖了这个公有特性。Java平台模块系统，成功地用于Java API的模块化。一个Java平台模块包含：

1. 一个包集合
2. 可选地包含资源文件和像本地库这样的文件
3. 一个有关模块中可访问的包的列表
4. 一个有关这个模块依赖的所有其他模块的列表
5. 模块的命名

模块是包的集合，模块中的包名无需彼此相关，就像路径名一样，模块名是由字母，数字，下划线和句点构成的，而且和路径名一样，模块之间是没有任何层次关系的。当构建供其他人使用的模块时，重要的是保证它的名字是全局唯一的。命名模块最简单的方式是按照模块提供的顶级包来命名，如org.slf4j模块，其中的包围org.slf4j, org.slf4j.spi等等，这个习惯可以防止模块中产生包名冲突。因为任何给定的模块只能放到一个模块之中，如果模块名是唯一的，并且包名以模块名开头，那么包名定是唯一的。且模块是没有作用域的概念的，因此不能在不同的模块中放置两个具有相同名字的包。

模块名只用于模块声明中，在Java类的源文件中，只是使用包名。

1. 模块化的程序示例
2. 创建包com.hello

在包下创建一个类HelloWorld

1. 创建包含这个包的模块hellomode

需要在模块中添加一个模块声明，可以将其置于名为module-info.java文件中，该文件位于基目录中（包含com目录的目录），按照惯例，基目录的名字与模块的名字相同。

module-info.java内容：

module hellomode

{}

这个模块声明之所以为空，是因为该模块没有任何可以向其他人提供的内容，也不需要依赖任何东西。

1. 编译模块

将模块声明和原始的类一起编译：

javac hellomode/module-info.java hellomode/com/hello/HelloWorld.java

模块声明文件最终会以二进制形式编译到包含该模块定义的类文件module-info.class中。

1. 模块化运行程序

为了让这个程序以模块化应用程序来运行，需要指定模块路径，与类路径相似，但是包含的是模块，还需要以模块名/类名的形式指定主类：

java --module-path hellomode –module hellomode/com.hello.HelloWorld

1. 对模块的需求

模块系统的设计目标之一就是模块需要明确它们的需求，使得虚拟机可以确保在启动程序之前所有的需求得到满足，即模块依赖哪些模块需要在模块声明中显示地写出来，使用requires关键词依赖其他模块，一个模块不能直接或者间接地对自己产生依赖，同时模块也不会自动地将访问权限传递给其他模块，即依赖是不会有传递性的。

module hellomode

{

requires java.desktop;

}

1. 模块导出的包

模块可以使用exports关键词来声明它的哪些包是可以导出的，其他的包就被隐藏了，当包被导出时，它的public和protected的类和接口以及成员在模块外部也是可以访问的。

module java.xml

{

exports javax.xml;

exports javax.xml.catelog;

…

}

1. 模块和反射

反射可以访问任何类的私有成员，但是在模块中，这条路是行不通的，如果一个类在模块中，那么对非公有成员的反射式访问将会失败。之前我们通过反射访问私有域，除非安全管理器不允许对私有域地访问，有很多使用反射式访问的库，如JPA这样的对象-关系映射器，他们会自动地将对象持久化到数据库中。如下就是访问私有域的方式：

Field f = obj.getClass().getDeclaredField(“salary”);

f.setAccessible(true);

double value = f.getDouble(obj);

f.setDouble(obj, value \* 1.1);

在模块中，使用opens关键词，打开指定的包，从而启动对给定包中的类的所有实例进行反射式访问，这样该包中的所有内容就可以被访问了，但是在运行时只有显式导出的包是可访问的。如下所示：

module hellomode

{

opens com.hello

}

1. 自动模块和不具名模块

如果从全新的项目开始，其中所有的代码都由我们自己编写，那么就可以设计模块，声明模块依赖关系，并将应用程序打包成模块化的JAR文件。但是这种情况目前还是达不到的，因为我们的项目肯定要依赖第三方库，第三方库很有可能没有模块化，因此Java平台模块系统提供了两种机制解决现在的情况：自动化模块和不具名模块。

1. 自动化模块

模块路径上美哦与module-info.class文件的JAR被称为自动模块，自动模块的属性为：

1. 隐式地包含对其他所有模块的requires子句；
2. 其所有包都被导出，且是开放的；
3. 如果在JAR文件清单META-INF/MANIFEST.MF中具有键为Automatic-Module-Name的项，那么它的值会变为模块名；
4. 否则，模块名将从JAR文件名中获得，将文件名中尾部的版本号删除
5. 不具名模块

任何不在模块路径中的类都是不具名模块的一部分，和自动模块一样，不具名模块可以访问所有其他的模块，它的所有包都会被导出，并且都是开放的。但是没有任何明确模块（非自动模块，非不具名模块）可以访问不具名模块。

1. 服务加载

在模块系统之前，使用ServiceLoader类将服务接口与实现匹配起来，而在Java平台模块系统中这种机制将会更加简单。服务消费者必须基于其认为合适的标准在所有的实现中选择一个，过去是通过将文本文件放置到包含实现类的JAR文件的META-INF/services目录中提供给服务消费者的，模块系统直接添加语句到模块描述符中。提供服务实现的模块可以添加一条provides语句，它列出了服务接口以及实现类，如下所示：

module jdk.security.auth

{

provides javax.security.auth.api.LoginModule with

com.sun.security.auth.module.UnixLoginModule,

…

}

这与META-INF/services文件是等价的，而使用它的消费模块包含一条uses语句：

module java.base

{

uses javax.security.auth.spi.LoginModule;

}

当消费模块中的代码调用ServiceLoader.load(ServiceInterface.class)时，匹配的提供者类将被加载。