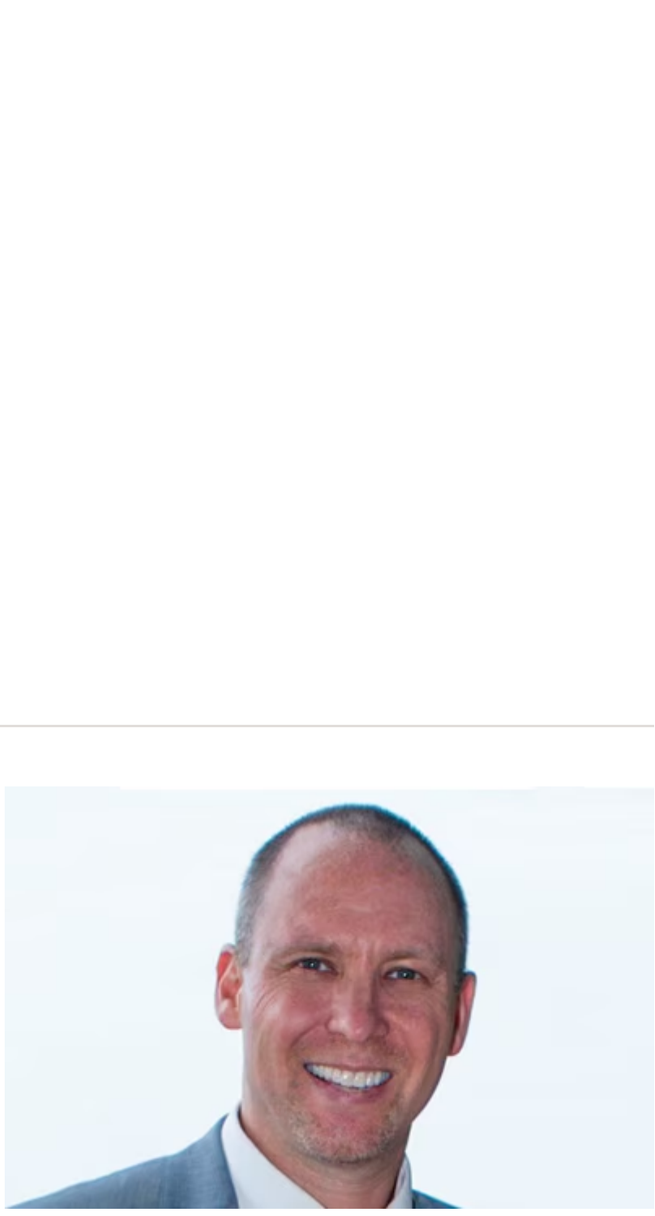
Java 9 | 摘要

了解 Java 9 模块

它们是什么以及如何使用它们

作者：Paul Deitel

在本文中，我将介绍 Java 9 Platform Module System (JPMS)，它是自 Java 诞生以来极重要的新软件工程技术。模块化是 Jigsaw 项目 的成果，可帮助各级开发人员在构建、维护和改进软件系统（尤其是大型系统）时提高工作效率。



Paul Deitel

什么是模块?

模块化在包之上提供更高层次的聚合。关键的新语言元素是**模块**，它是一组专属命名、可重用的相关包、资源（例如图像和 XML 文件）和一个**模块描述符**，用于指定

- 模块**名称**
- 模块的**依赖项**（即此模块依赖的其他模块）
- 明确可以让其他模块使用的包（其他模块**隐式不可用**模块中的所有其他包）
- **它提供的服务**
- **它使用的服务**
- 允许哪些其他模块使用**reflection**

历史记录

Java SE 平台自 1995 年推出以来，现大约有 1000 万名开发人员使用它来为物联网 (IoT) 和其他嵌入式设备等受资源限制的设备构建各种小型应用，到大规模业务关键型和任务关键型系统。市面上有大量的旧有代码，但到目前为止，Java 平台主要是单一的通用解决方案。多年来，业者针对模块化 Java 投入了各种努力，但没有一个被广泛使用，也没有一个可以用于模块化 Java 平台。

实现 Java SE 平台模块化是一项花了很多年且具有挑战性的努力。JSR 277：Java 模块系统当初于 2005 年针对 Java 7 推出，之后被 JSR 376：Java Platform Module System 取代，作为 Java 8 的目标，直到 Java 9 延迟到 2017 年 9 月推出之后，Java SE 平台在 Java 9 中进行了模块化处理。

目标

根据 JSR 376，对 Java SE 平台进行模块化的主要目标是

> 每个模块必须明确地指出其依赖项。

- 可靠的配置 — 模块提供了在编译期和执行期加以识别模块之间依赖项的机制，系统可以通过这些依赖项确保所有模块的子集合能满足程式的需求。
- 强封装 — 模块中的包只有在模块显式导出时可供其他模块访问，即使如此，其他模块也无法使用这些包，除非它明确指出它需要其他模块的功能。由于潜在攻击者只可以访问较少的类别，因此提高了平台的安全性。您会发现，模块化可帮助您做出更简洁、更合乎逻辑的设计。
- 可扩展的 Java 平台 — 在这之前，Java 平台是由大量的包组成的庞然大物，使其难以开发、维护和发展，还不能轻易地被子集化。现在，平台被模块化为 95 个模块（此数字可能随着 Java 的发展而变化），您可以创建定制运行时，其中仅包含应用所需的模块或目标设备。例如，如果设备不支持 GUI，您可以创建一个不包含 GUI 模块的运行时，显著降低运行时的大小。
- 更佳的平台完整性 — 在 Java 9 之前，您可以使用平台中许多不预期供应用使用的类别，通过强封装，这些内部 API 真正会被封装并隐藏在使用该平台的应用中。如果您的代码依赖这些内部 API，则可能会对转移旧有代码到以模块化的 Java 9 造成问题。
- 提高性能 — JVM 使用各种优化技术来提高应用性能。JSR 376 表明，当预先知道某技术仅在特定模块中被使用，这些技术会更有效。

JDK 模块列表

Java 9 的一个关键概念是将 JDK 切割成模块以支持各种配置。（参阅“JEP 200：对 JDK 进行模块化。”**表 1** 显示了与 Java 模块化有关的所有 JEP 和 JSR。)使用 JDK bin 文件夹中的 java 命令配搭 `--list-modules` 选项，如下所示：

```
java --list-modules
```

列出 JDK 的模块集，其中包括实施 Java SE 语言规范（以 `java` 开头的名称）、JavaFX 模块（以 `javafx` 开头的名称）、特定于 JDK 的模块（以 `jdk` 开头的名称）和特定于 Oracle 的模块（以 `oracle` 开头的名称）的标准模块。每个模块名称后都接着一个版本字符串 — `@9` 表示该模块属于 Java 9。

模块声明

正如我们所提到的，模块必须提供一个模块描述符，也就是用于指定模块的依赖项、让其他模块使用的包等的元数据。模块描述符是一个经过编译的模块声明，定义于 `module-info.java` 档案中。每个模块声明以关键字 `module` 开头，紧接着一个专属的模块名称，以及括在括号中的模块主体内容，如下所示：

```
module

modulename {
}
```

模块系统的关键动力是强封装。

JEP 200：模块化 JDK
JEP 201：模块化源代码
JEP 220：模块化运行时镜像
JEP 260：封装大多数内部 API
JEP 261：模块系统
JEP 275：JAVA 应用模块化打包
JEP 282：JAVA 连接器 JLINK
JSR 376:JAVA 平台模块系统
JSR 379：JAVA SE 9
表 1 Java 模块化 JEP 和 JSR

模块声明的正文可以留空，也可以包含各种 **module 指令**，包括 `requires`、`exports`、`provides…with`、`uses` 和 `opens`（我们讨论的每个指令）。您稍后会看到，编译模块声明会创建模块描述符，存储在模块根文件夹中名为 `module-info.class` 的文件中。在这里，我们简要介绍每个模块指令。过后，我们将展示实际的模块声明。

关键字 `exports`、`module`、`open`、`opens`、`provides`、`requires`、`uses`、`with` 以及后面介绍的 `to` 和 `transitive` 是受限关键字。这些仅是模块声明中的关键字，可用作代码中的标识符。

requires。`requires` 模块指令指定此模块依赖于另一个模块，此关系称为**模块依赖关系**。每个模块必须明确地指出其依赖项。当模块 A `requires` 模块 B 时，模块 A 称为**读取**模块 B，模块 B 则是给模块 A **读取**。如需指定对其他模块的依赖性，请使用 `requires`，如下所示：

```
requires modulename;
```

还有一个 `requires static` 指令，用于指示在编译时必需要有模块，在运行时则不是必须的。这称为**可选相关项**，但不会在本介绍中讨论。

requires transitive — 隐式可读性。指定对其他模块的依赖性并确保其他模块读取您的模块时也能读取这依赖性，所谓的**隐式可读性**，请使用 `requires transitive`，

```
requires transitive modulename;
```

思考 `java.desktop` 模块声明中的以下指令：

```
requires transitive java.xml;
```

在这种情况下，任何读取 `java.desktop` 的模块也会隐式读取 `java.xml`。例如，如果 `java.desktop` 模块中的方法返回 `java.xml` 模块中的类型，则读取 `java.desktop` 的模块中的代码将依赖于 `java.xml`。如果 `java.desktop` 模块声明中没有 `requires transitive` 指令，除非这些相关模块**显式**读取 `java.xml`，否则这些模块将不会编译。

根据 JSR 379，Java SE 的标准模块必须在所有情况下（如此处所述）授予隐含的可读性。此外，虽然 Java SE 标准模块可能依赖于非标准模块，但是它**不能**授予隐含的可读性。这可确保代码仅依赖于 Java SE 标准模块，可以执行于任何 Java SE 的实作版本中。

exports 及 exports…to。`exports` 模块指令指定模块的一个包，其 `public` 类型（及其嵌套的 `public` 和 `protected` 类型）应可供所有其他模块的代码访问。通过 `exports…to` 指令，您可以在逗号分隔的列表中指定哪个模块或模块的代码可以访问导出的包，这就是所谓的**限定**导出。

uses。`uses` 模块指令指定此模块所使用的服务，使此模块成为服务使用者。**service** 是类的对象，用于实施接口或扩展 `uses` 指令中指定的 `abstract` 类。

provides…with。`provides…with` 模块指令指定模块提供服务实施，使模块成为**服务提供者**。指令的 `provides` 部分指定模块的 `uses` 指令指令中列出的接口或 `abstract` 类；指令的 `with` 部分指定 `implements` 接口或 `extends abstract`类的服务提供类的名称。

open、opens 及 opens…to。在 Java 9 之前，reflection 可用于了解包中的所有类型以及类型的所有成员，包括其 私有成员，无论您是否允许此功能。因此，没有任何东西是真正封装的。

模块系统的关键优势是强封装。默认情况下，模块中的某个类型是无法让其他模块访问的，除非它是公开类型**并且**您导出它的包，您只需要公开您希望公开的包，在 Java 9 中，这也适用于 reflection。

只允许在运行时访问特定的包。open 模块指令的形式

```
opens package
```

表示特定 **包** 的 `public` 类型（及其嵌套的 `public` 和 `protected` 类型）只能在运行时可供其他模块中的代码访问。同样，指定包中的所有类型（以及所有类型的成员）都可通过 reflection 进行访问。

只允许在运行时让特定模组访问特定的包。`opens…to` 模块指令的形式

```
opens package to comma-separated-list-of-modules
```

表示特定**包** 的 `public` 类型（及其嵌套的 `public` 和 `protected` 类型）只能在运行时可供列出的模块中的代码访问。同样，指定包中的所有类型（以及所有类型的成员）都可通过 reflection 进行访问。

只允许在运行时能访问模块中的所有包。如果某模块中的所有包能在运行时并通过 reflection 让所有其他模块访问，您可以公开整个模块，如下所示：

```
open module modulename {
   // module directives
}
```

Reflection 默认值

默认情况下，一个在运行时拥有反射访问权限的模块可以查看该包的 `public` 类型（及其嵌套的 `public` 和 `protected` 类型）。但是，其他模块中的代码可以访问公开的包的**所有**类型以及这些类型中的**所有**成员，包括通过 `setAccessible` 访问

私有成员，和先前的 Java 版本相同。

Paul Deitel 是 Deitel & Associates 首席执行官兼首席技术官，毕业于麻省理工学院，拥有 35 年的计算经验。他拥有超过 22 年的 Java 编程经验，是 Java Champion 获得者。他和合著者 Harvey M. Deitel 博士同为全球畅销编程语言书籍的作者。Paul 在国际上为工业界、政府和学术界的客户提供 Java、Android、iOS、C#、C++、C 和互联网编程课程。

注：本文摘自 **Java Magazine** 2017 年 9 月/10 月刊。

更多资源

Jigsaw 项目：模块系统快速入门指南　　Java 9 模块化：开发可维护应用的模式和练习　　Java Magazine

注：为免疑义，本网页所用以下术语专指以下含义：

1. 除Oracle隐私政策外，本网站中提及的“Oracle”专指Oracle境外公司而非甲骨文中国。
2. 相关Cloud或云术语均指代Oracle境外公司提供的云技术或其解决方案。