Atitit 依赖管理之道

[1. 概念 依赖管理，是指在什么地方以什么形式引入外部代码。 1](#_Toc11141)

[1.1.1. 理解模块化和依赖管理： 1](#_Toc24635)

[1.2. 依赖管理，有三个层面。 单一职责原则，协议对象引用，依赖逻辑关系 2](#_Toc7388)

[2. 五大数据解耦 视图、数据模型、网络请求、本地存储、页面跳转五块代码 2](#_Toc9564)

[3. Maven 2](#_Toc9376)

[4. Maven最佳实践：管理依赖 1 3](#_Toc8379)

[4.1. 最简单的依赖 2 4](#_Toc12839)

[4.2. 依赖归类 2 4](#_Toc21081)

[4.3. 依赖范围(scope) 3 4](#_Toc846)

[4.4. 分类器（classifer） 5 4](#_Toc20222)

[4.5. 依赖管理（dependencyManagement） 6 4](#_Toc22196)

[5. 与其他工具的区别 4](#_Toc11255)

[5.1. Ruby的gem，Node的npm，Python的pip，iOS的CocoaPods都类似，只是配置文件语法和坐标命名规则有些差异。 4](#_Toc8427)

[5.2. App store 4](#_Toc30211)

[6. 依赖管理工具，设计 4](#_Toc25680)

[7. 组合层次 fun》jobj,pkg,namespae,modu 5](#_Toc29786)

[8. 参考资料 5](#_Toc2137)

# 概念 依赖管理，是指在什么地方以什么形式引入外部代码。

依赖管理为什么重要呢？ 因为，依赖在哪里，处理代码就会那里，而bug也就会出现在那里。 于是，反过来，为了让bug集中，就需要让处理代码集中，也就需要管理好代码的依赖关系。

### **理解模块化和依赖管理：**

* 应用一般都会依赖外部库(jQuery和AngularJs),这些库应该使用包管理器处理和更新，而不是手动下载。
* 应用本身也可以分解成为多个相互交互的小部分,学习封装代码的技能,把代码视作自成一体的组件,学习如何设计优秀的接口,如何安排接口;学习如何隐藏数据,只开放用户需要的部分。
* 如何解析依赖,避免手动维护一组有序的script标签。

## 依赖管理，有三个层面。 单一职责原则，协议对象引用，依赖逻辑关系

1､相同的代码，只出现在一个地方，也叫单一职责原则。

2､通过引用协议对象，让依赖关系中的组件更容易更换。

3､从逻辑上安排依赖关系，让依赖关系更容易被人理解。

单一职责原则，是最基本的原则。它是代码模块化，设计模式的根本。

协议对象引用，使得功能变更时，只需要在一个统一的地方做最少的修改。

依赖逻辑关系，则主要是为了更方便人脑去理解代码之间的关系。

# 五大数据解耦 视图、数据模型、网络请求、本地存储、页面跳转五块代码

# Maven

Atitit maven使用总结attilax总结.docx

atitit 项目管理 package 模块管理 maven attilax总结.docx

Maven诞生于2004年(来源维基)，查询了下，应该是各语言的依赖管理工具中早的。Ruby的gem也是2004年出现的，但gem离完备的依赖管理工具还差些，直到Ruby的bundler出现。Python的pip出现的更晚。

Maven的习惯是通过 groupID(一般是组织的域名倒写，遵循Java package的命名习惯)+ artifactId(库本身的名称) + version(版本)来定义坐标，通过xml来做配置文件，提供了中心仓库(repo.maven.org)以及本地工具(mvn)。

依赖定义：  <dependency>  <groupId>com.google.guava</groupId>  <artifactId>guava</artifactId>  <version>18.0</version>  </dependency>  repo定义：  <repository>  <id>repo.default</id>  <name>Internal Release Repository</name>  <url>http://repo.xxxxxx.com/nexus/content/repositories/releases</url>  <releases>  <enabled>true</enabled>  <updatePolicy>interval:60</updatePolicy>  <checksumPolicy>warn</checksumPolicy>  </releases>  <snapshots>  <enabled>false</enabled>  <updatePolicy>always</updatePolicy>  <checksumPolicy>warn</checksumPolicy>  </snapshots>  </repository>

同时，为了避免依赖冲突的问题，Maven的依赖配置提供了exclude配置机制，用于阻断部分库的传递依赖。

# Maven最佳实践：管理依赖 1

Maven最佳实践：管理依赖 1

## 最简单的依赖 2

## 依赖归类 2

## 依赖范围(scope) 3

## 分类器（classifer） 5

## 依赖管理（dependencyManagement） 6

# 与其他工具的区别

## Ruby的gem，Node的npm，Python的pip，iOS的CocoaPods都类似，只是配置文件语法和坐标命名规则有些差异。

## App store

# 依赖管理工具，设计

1.要有一种依赖库的命名规则，或者叫坐标(Coordinates)的定义规则，可以通过坐标准确找到依赖的库。

2.要有对应的配置文件规则，来描述和定义依赖。

3.要有中心仓库保存这些依赖库，以及依赖库的元数据(metadata)，供使用方拉取。

4.还需要一个本地工具去解析这个配置文件，实现依赖的拉取。

以上其实就是各依赖管理工具的核心要素。

# 组合层次 fun》jobj,pkg,namespae,modu

function用于组合代码块逻辑，有了object用于组合一组方法，有了package，namespace用于组合一组相关对象，但其实还需要有更高一个层次的组合定义 —– module，或者叫子项目

# 可选择性依赖

的另一个原因，是Android 框架已经提供了一种解决方案，但是某个外部库提供的解决方案性能更好。如果用户本就依赖于该外部库，或者愿意增加方法数量以获得更好的性能，就可以添加可选择性依赖。

我最近看到的[PlacesAutocompleteTextView](https://github.com/seatgeek/android-PlacesAutocompleteTextView" \o "PlacesAutocompleteTextView" \t "https://yq.aliyun.com/articles/_blank)库，就属于这种情况。该库使用的内部 HTTP 客户端，既可以是 OkHttpClient，也可以是 HttpURLConnection。通常，前者的性能更好，但是需要添加 [OkHttp](http://square.github.io/okhttp/" \o "OkHttp" \t "https://yq.aliyun.com/articles/_blank) 作为依赖。 如果用户不想包含该依赖，可以自动从标准库回退到 HttpURLConnection。

该类被加载时，会检查 OkHttpClient 的完全限定类名是否可用。如果抛出 ClassNotFoundException，我们就知道用户没有添加 OkHttp，于是回退到 HttpURLConnection。PlacesHttpClient 是包装以上两种实现方式的公共接口，因此在整个代码库中，这两种实现方式可以交换使用。JSON 解析也采用了同样的方法，[Gson](https://github.com/google/gson" \o "Gson" \t "https://yq.aliyun.com/articles/_blank) 可选择性地作为依赖包含在库中。

如果性能表现与库的大小之间的权衡系数很大，这个方法确实不错。但是，如果回退的实现方式比较困难（比如 JSON 解析就是这种情况），笔者建议你先使用外部库来节省时间，在后续的版本中再考虑添加回退实现。

# 参考资料

漫谈依赖管理工具：从Maven,Gradle到Go\_TechWeb.mhtml

漫谈依赖管理工具：从Maven,Gradle到Go\_TechWeb.mhtml

Maven最佳实践：管理依赖 - 飞龙在天001 - 博客园.mhtml