LightHouseIT

2章 网络架构　　ウェブアーキテクチャ

- MVC架构　MVCアーキテクチャ
- Maven　Maven

目录

LightHouseIT

JavaBean规范

- 一个Java类满足以下条件，则被称为JavaBean：
  - 所有的属性均为private（每个属性都具有getter/setter方法）
  - 具有一个public的无参构造方法
  - 实现 Serializable 接口（实现此接口的类的对象可以被写入到IO流中）

- JavaBean是类的规范，如果称一个Java类为JavaBean，则它一定具有以上三种性质。

架构模式[アーキテクチャパターン]

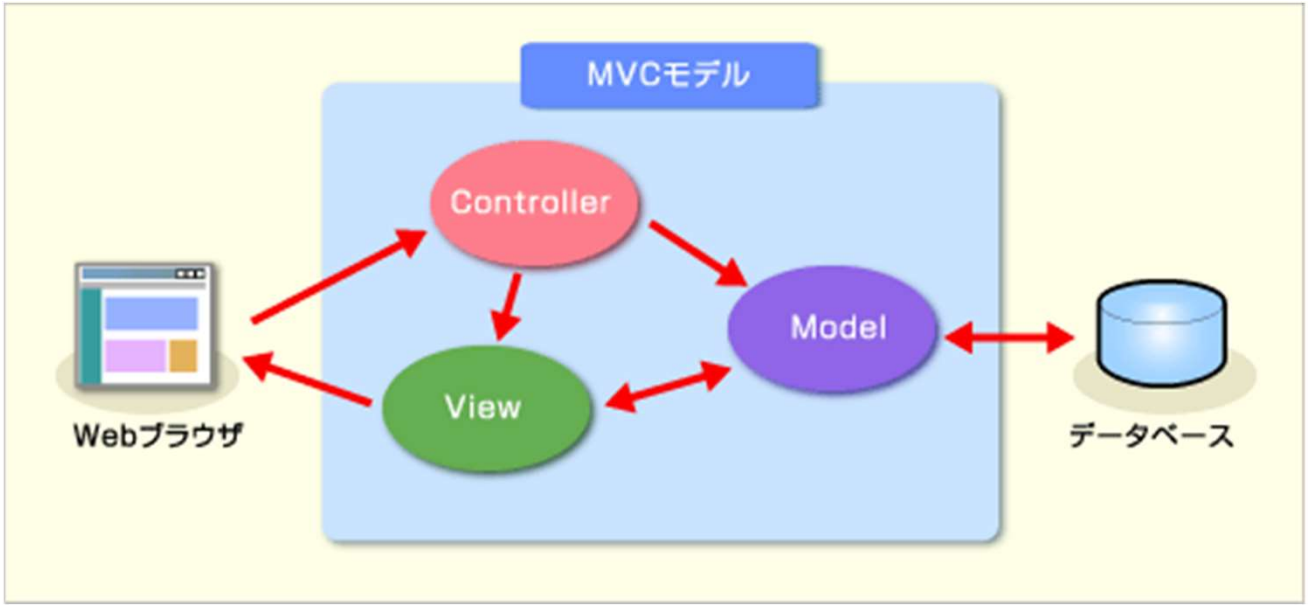
- **架构模式**（architectural pattern）是软件架构中在给定环境下常遇到问题的通用的、可重用的解决方案。

- 常见架构模式：
  - 模型—视图—控制器（MVC）
  - 分层
  - 管道和过滤器
  - 代理者
  - 黑板
  - 表示--抽象--控制（PAC）

- 本节课讲重点介绍 MVC 架构模式

MVC 简介

- **MVC模式**（Model–view–controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：**模型（Model）、视图（View）和控制器（Controller）**。
- 最早由Trygve Reenskaug在1978年提出，是施乐帕罗奥多研究中心（Xerox PARC）在20世纪80年代为程序语言Smalltalk发明的一种软件架构。

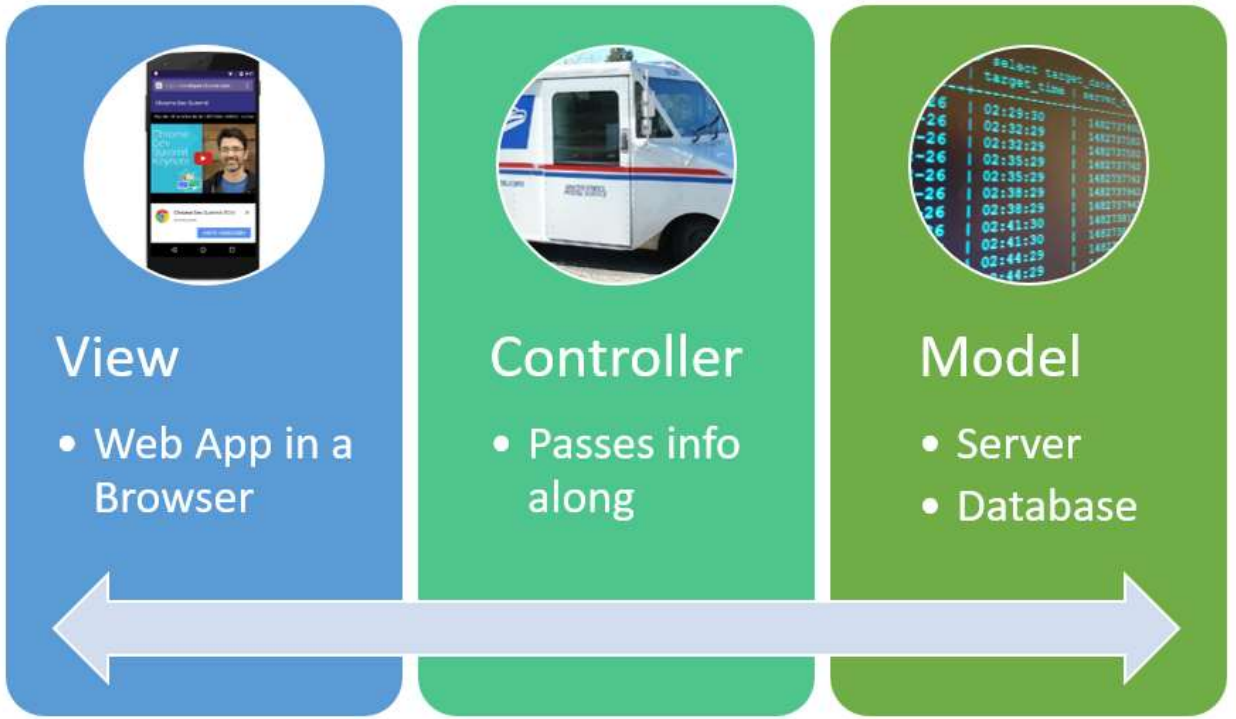


MVC 优势

- 使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。
- 使程序结构更加直观。软件系统透过对自身基本部分分离的同时也赋予了各个基本部分应有的功能。
- 专业人员可以依据自身的专长分组：
  - 模型（Model） - 数据库专家进行数据管理和数据库设计(可以实现具体的功能)。
  - 视图（View） - 界面设计人员进行前端界面设计。
  - 控制器（Controller） - 负责转发请求，对请求进行处理。

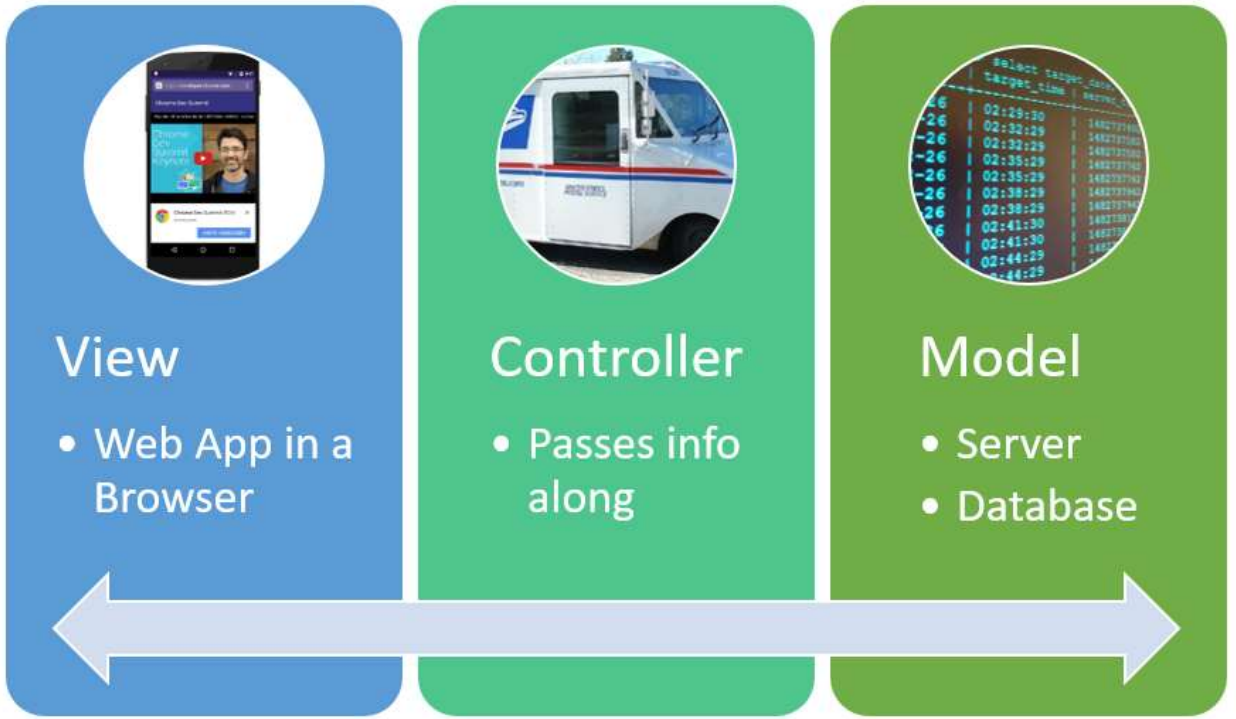
MVC 组件的互动—视图[ビュー]（View）

- 能够实现数据有目的的显示。在 View 中一般没有程序上的逻辑。为了实现 View 上的刷新功能，View 需要访问它监视的数据模型（Model），因此应该事先在被它监视的数据那里注册。



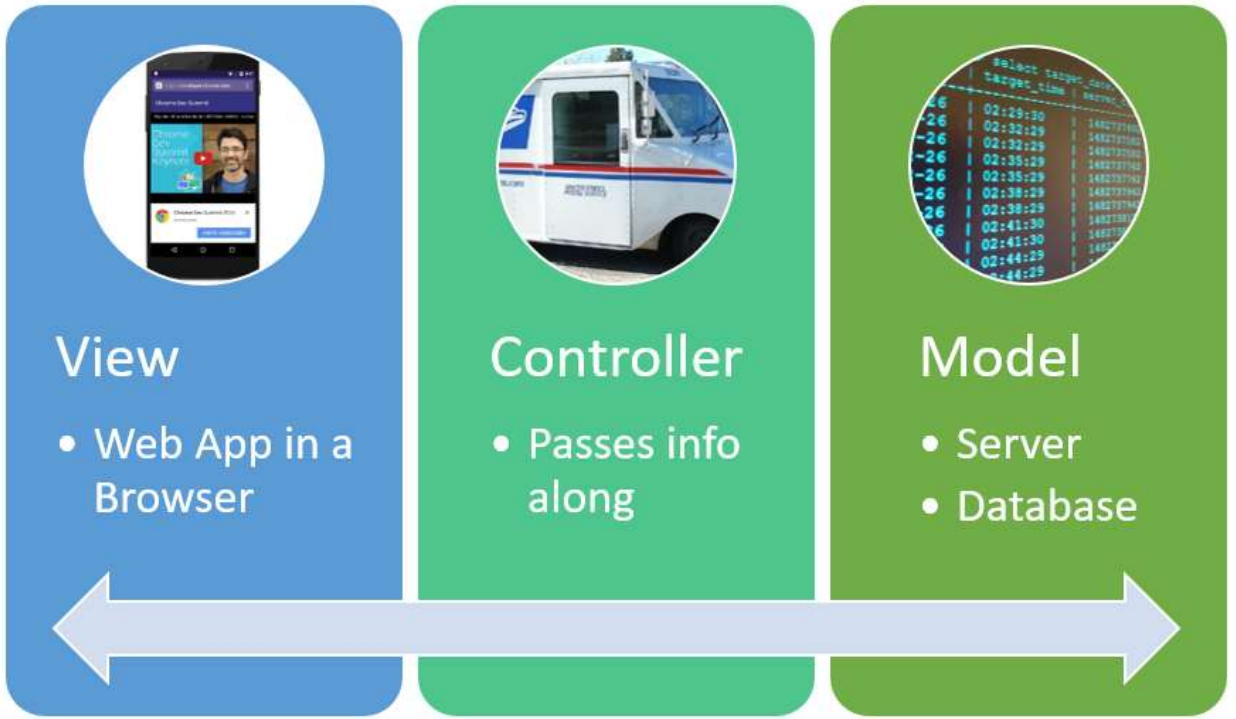
MVC 组件的互动—控制器[コントローラー]（Controller）

- 起到不同层面间的组织作用，用于控制应用程序的流程。它处理事件并作出响应。“事件”包括用户的请求行为和数据 Model 上的改变。



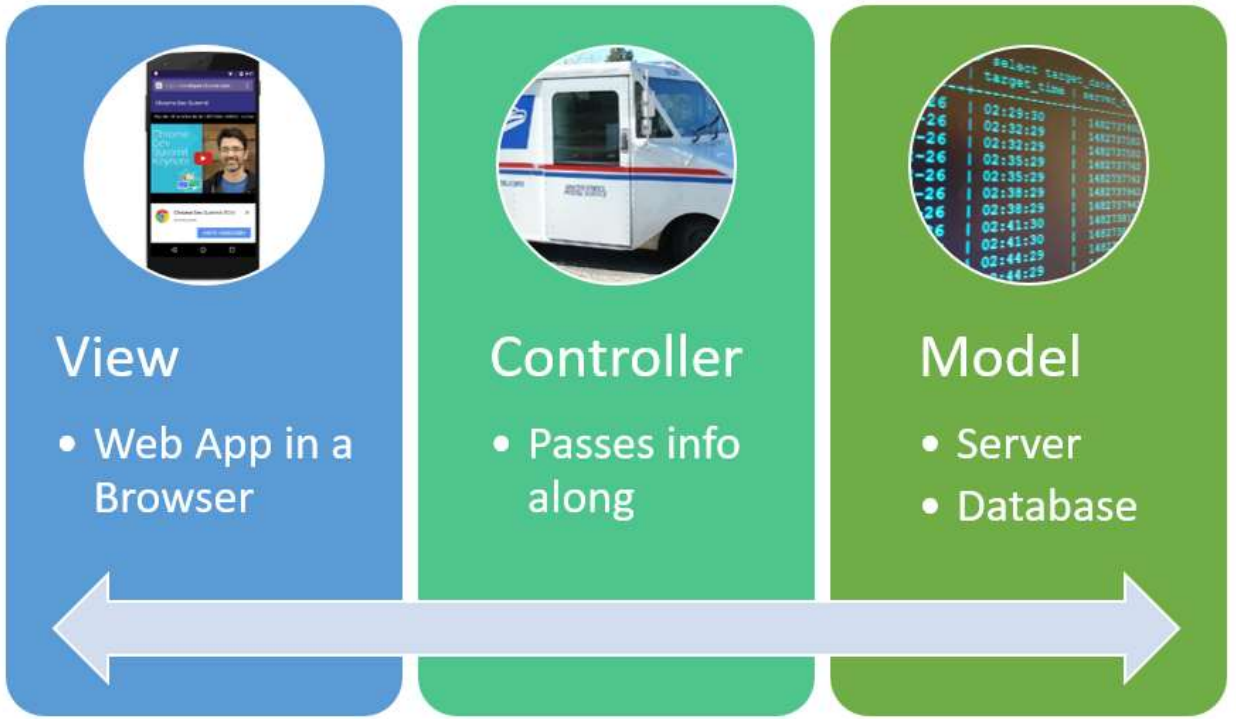
MVC 组件的互动—模型[モデル]（Model）

- 模型（Model） 用于封装与应用程序的业务逻辑相关的数据以及对数据的处理方法。代表一个存取数据的对象（JavaBean），也包括数据处理和业务逻辑。

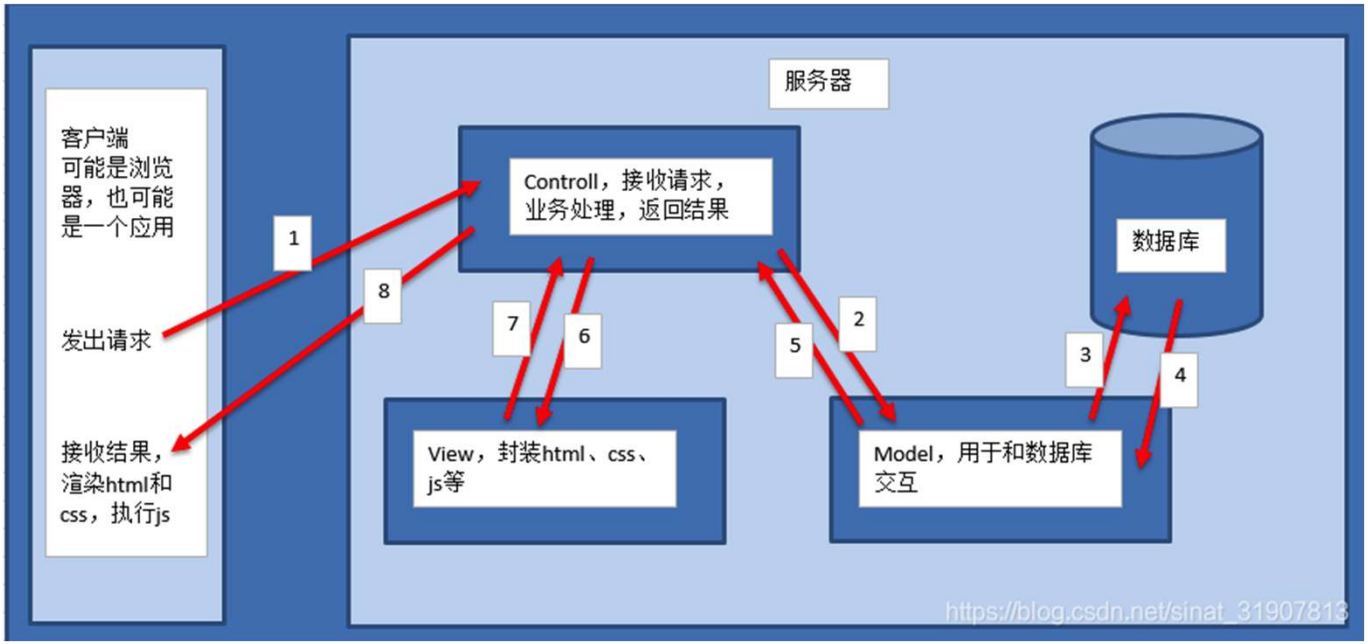


MVC 总结

- Model是项目的数据模型，View是视图展示，而Controller则是控制数据在视图上展示的逻辑。

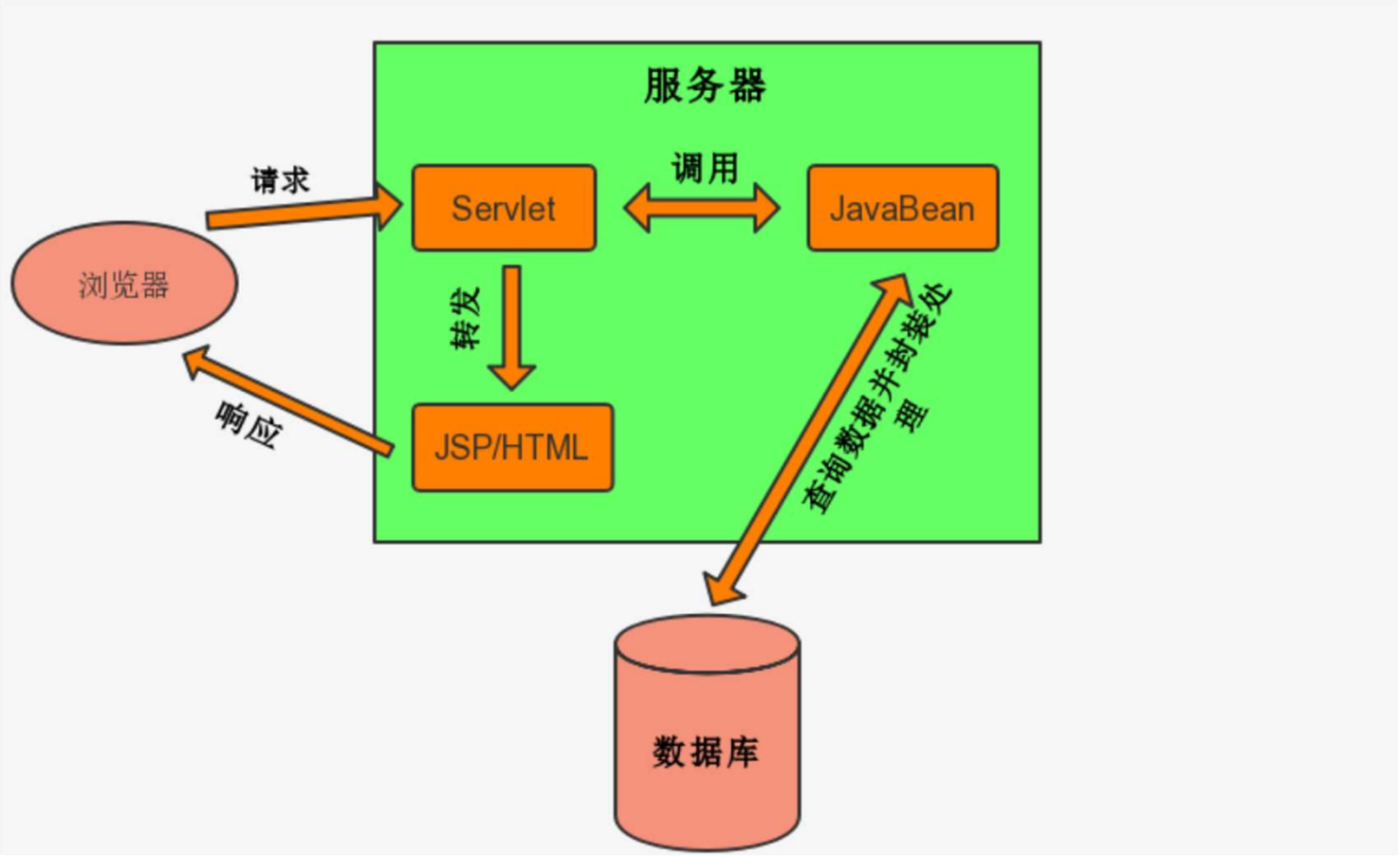


MVC流程图



练习

- 这是一个基于Servlet实现的MVC架构模式图，这里的M，V，C分别对应哪些部分？



LightHouseIT

Maven概述

- Maven是一个Java项目管理和构建工具，它可以定义项目结构、项目依赖，并使用统一的方式进行自动化构建，是Java项目不可缺少的工具。理工具

为什么需要Maven?

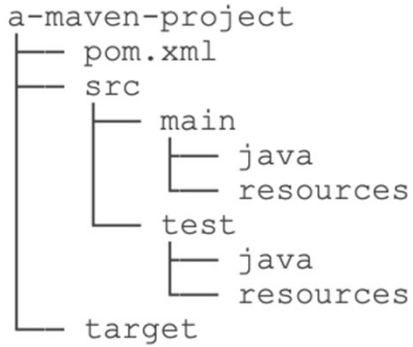
- 在了解Maven之前，我们先来看看一个Java项目需要的东西
- 首先，我们需要确定引入哪些依赖包。例如，如果我们需要用到servlet，我们就必须把servlet的jar包放入classpath。如果我们还需要tomcat，就需要把tomcat相关的jar包都放到classpath中。这些就是依赖包的管理。
- 其次，我们要确定项目的目录结构。例如，src目录存放Java源码，resources目录存放配置文件，bin目录存放编译生成的.class文件。
- 此外，我们还需要配置环境，例如JDK的版本，编译打包的流程，当前代码的版本号。
- 最后，除了使用Eclipse这样的IDE进行编译外，我们还必须能通过命令行工具进行编译，才能够让项目在一个独立的服务器上编译、测试、部署。

为什么需要Maven?

- 以上这些工作难度不大，但是非常琐碎且耗时。如果每一个项目都自己搞一套配置，肯定会一团糟。我们需要的是一个标准化的Java项目管理和构建工具。
- Maven就是是专门为Java项目打造的管理和构建工具，它的主要功能有：
  - 提供了一套标准化的项目结构；
  - 提供了一套标准化的构建流程（编译，测试，打包，发布……）；
  - 提供了一套依赖管理机制。

Maven项目结构

- 一个使用Maven管理的普通的Java项目，它的目录结构默认如下：



- a-maven-project：项目的根目录，项目名
- pom.xml：项目描述文件
- src/main/java：存放Java源码的目录
- src/main/resources：存放资源文件的目录
- src/test/java：存放测试源码的目录
- src/test/resources：存放测试资源的目录
- target：所有编译、打包生成的文件

项目描述文件pom.xml

- 最关键的一个项目描述文件pom.xml，示例如下：

```
<project ...>
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.lighthouseit.java</groupId>
   <artifactId>hello</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>
   <properties> ... </properties>
   <dependencies>
      <dependency>
         <groupId>javax.servlet</groupId>
         <artifactId>javax.servlet-api</artifactId>
         <version>3.1.0</version>
      </dependency>
   </dependencies>
</project>
```

项目描述文件pom.xml

- groupId : 类似于Java的包名，通常是公司或组织名称
- artifactId: 类似于Java的类名，通常是项目名称
- Version: 项目版本号
- 一个Maven工程就是由groupId，artifactId和version作为唯一标识。

声明依赖

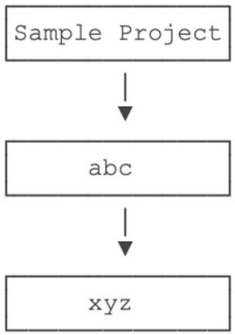
- 我们在引用其他第三方库的时候，也是通过这3个变量确定。例如，依赖servlet：

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
```

- 对于某个依赖，Maven只需要3个变量即可唯一确定某个jar包：
  - groupId：属于组织的名称，类似Java的包名；
  - artifactId：该jar包自身的名称，类似Java的类名；
  - version：该jar包的版本。
- 使用<dependency>声明一个依赖后，Maven就会自动下载这个依赖包并把它放到classpath中。

Maven 依赖管理

- Maven解决了依赖管理问题。例如，我们的项目依赖abc这个jar包，而abc又依赖xyz这个jar包：



- 当我们声明了abc的依赖时，Maven自动把abc和xyz都加入了我们的项目依赖，不需要我们自己去研究abc是否需要依赖xyz。
- 我们声明了自己的项目需要abc，Maven会自动导入abc的jar包，再判断出abc需要xyz，又会自动导入xyz的jar包，这样，最终我们的项目会依赖abc和xyz两个jar包。

Maven 依赖管理例子

- 我们来看一个复杂依赖示例：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>1.4.2.RELEASE</version>
</dependency>
```

- 当我们声明一个spring boot 依赖时，Maven会自动解析并判断最终需要大概二三十个其他依赖：

```
spring-boot-starter-web
  spring-boot-starter
    spring-boot
    sprint-boot-autoconfigure
    spring-boot-starter-logging
      logback-classic
        logback-core
        slf4j-api
      jcl-over-slf4j
        slf4j-api
      jcl-over-slf4j
        slf4j-api
      jul-to-slf4j
        slf4j-api
      log4j-over-slf4j
        slf4j-api
  spring-core
  snakeyaml
spring-boot-starter-tomcat
  tomcat-embed-core
  tomcat-embed-el
  tomcat-embed-websocket
    tomcat-embed-core
jackson-databind
...
```

- 如果我们自己手动管理这些依赖是非常费时费力的，而且出错的概率很大。

Maven POM

- POM，Project Object Model，工程对象模型
- 它是使用 Maven 工作时的基本组建，是一个 xml 文件。它被放在工程根目录下，文件命名为 pom.xml。
- POM 包含了关于工程和各种配置细节的信息，Maven 使用这些信息构建工程。
- 能够在 POM 中设置的一些配置如下:
  - project dependencies
  - plugins
  - goals
  - build profiles
  - project version
  - developers
  - mailing list

搜索第三方依赖

- 最后一个问题：如果我们要引用一个第三方依赖，比如Tomcat，如何确切地获得它的groupId、artifactId和version？
- 方法是在网站 search.maven.org 上搜索关键字，就能找到对应的依赖的三个信息

Group ID	Artifact ID	Latest Version
org.apache.tomcat	tomcat	10.0.5