

## 6.2 开发框架 Spring

---

- MVC 架构
- Spring 框架
- Maven



# 目录

1

MVC 架构

2

Spring 框架

3

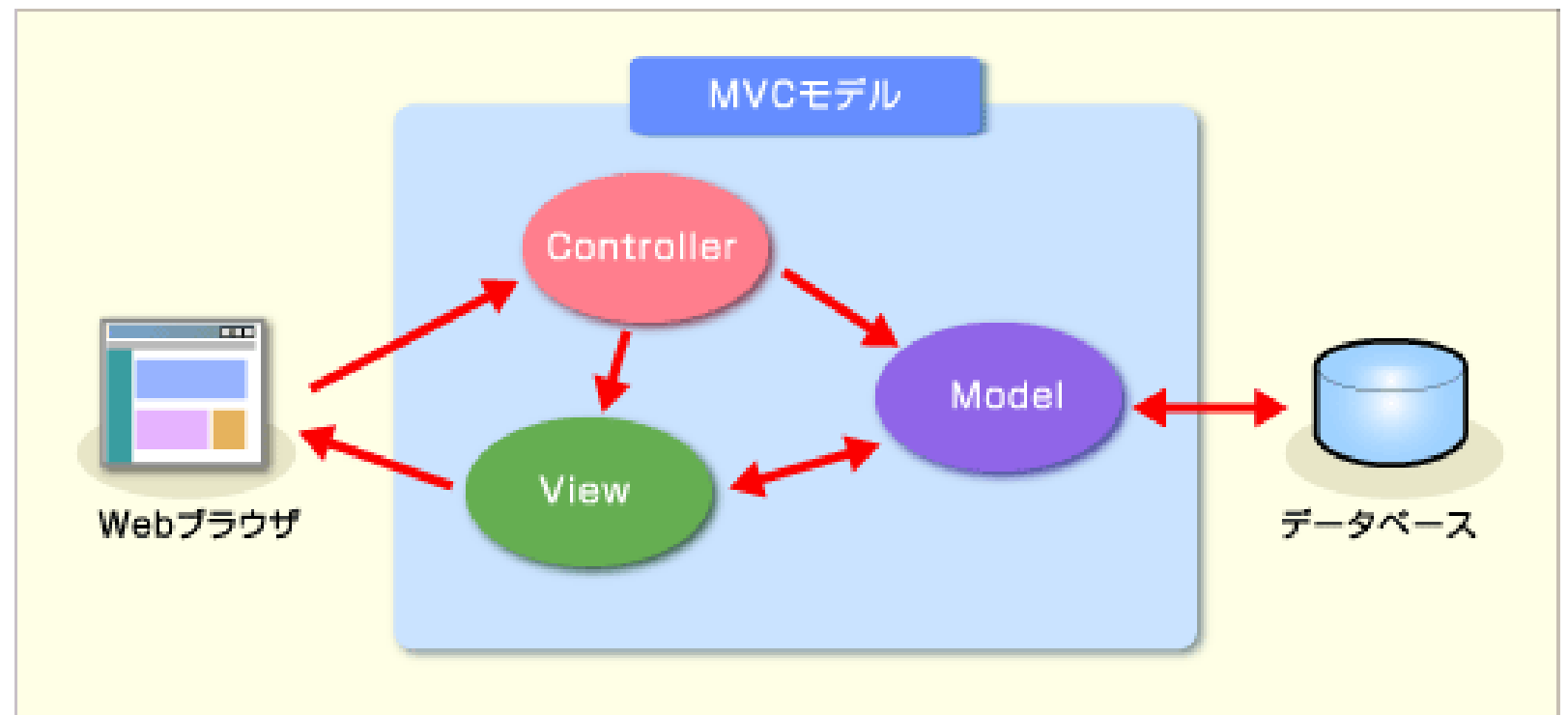
Maven

# 软件架构

- **软件架构**[ソフトウェアアーキテクチャ] 是对于软件整体结构与组件的抽象描述。我们在设计软件时不必从零开始，而是可以选择某一个（或多个）已经存在的**软件架构模式**[アーキテクチャパターン]，在其基础上具体实现各部分的功能。
- 常见的架构模式包括：
  - MVC 模式、MVP 模式、MVVM 模式：图形应用的架构模式；
  - 多层架构：解耦服务器系统；
  - 主从式架构：包含服务端与客户端的经典网络架构；
  - 管道与过滤器模式：维护进程间通信的统一接口；
  - 点对点网络（P2P）：去中心化的网络通信模式；
  - 黑板模式：经典的人工智能知识表现模型等。
- 本节将介绍 **MVC 架构模式**。

# MVC 架构模式

- **MVC 模式** (Model-view-controller) 把软件系统分为三个基本部分：模型 (Model)、视图 (View) 和控制器 (Controller)。
- MVC 模式早在 1978 年就被提出，是施乐帕罗奥多研究中心 (Xerox PARC) 为程序语言 Smalltalk 发明的一种桌面软件的架构模式。但它在互联网应用中也得到广泛运用，成为了最主流的网络应用架构之一。





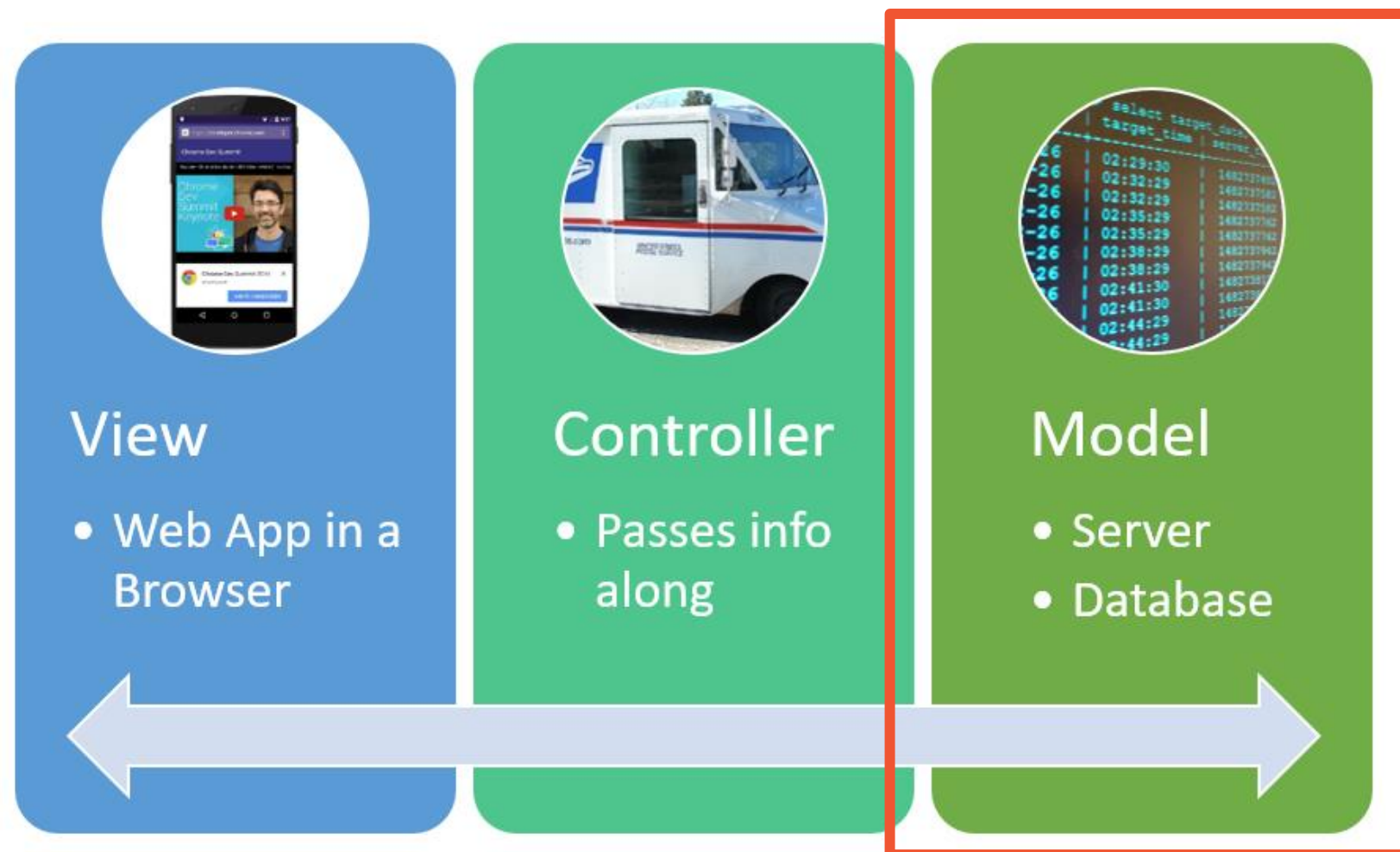
# MVC 模式的优点

- MVC 架构模式具有以下优点：

1. 它使后续对程序的**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。
2. 它使程序**结构**更加直观。每个基本部分只需实现它们应有的功能，而各个部分之间的**关系**也可以被清楚严格地规定下来。
3. 专业人员可以依据自身的专长分组，专注于实现 **1** 个部分的功能：
  - **模型**（Model）：由程序员进行数据结构与算法的设计、数据库专家进行数据管理和数据库设计。
  - **视图**（View）：由界面设计人员进行前端界面设计。
  - **控制器**（Controller）：由服务器开发人员设计对客户端请求的处理、转发逻辑。

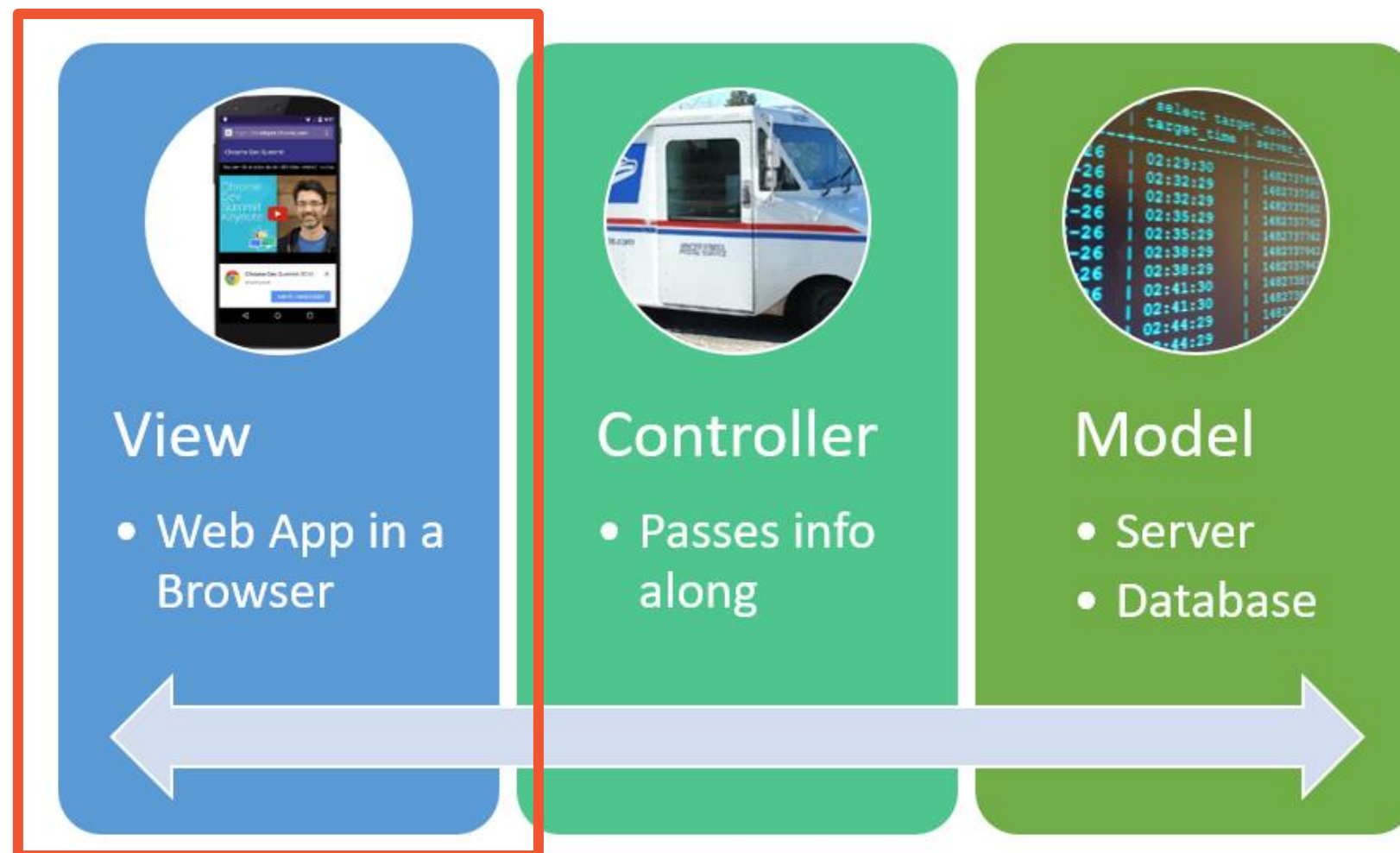
# MVC 组件：模型

- **模型**<sup>[モデル]</sup>部分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**。比如在 Java 中它可能代表一些用于存取数据的对象（如 JavaBeans），以及与存取数据有关的数据库功能与逻辑。



# MVC 组件：视图

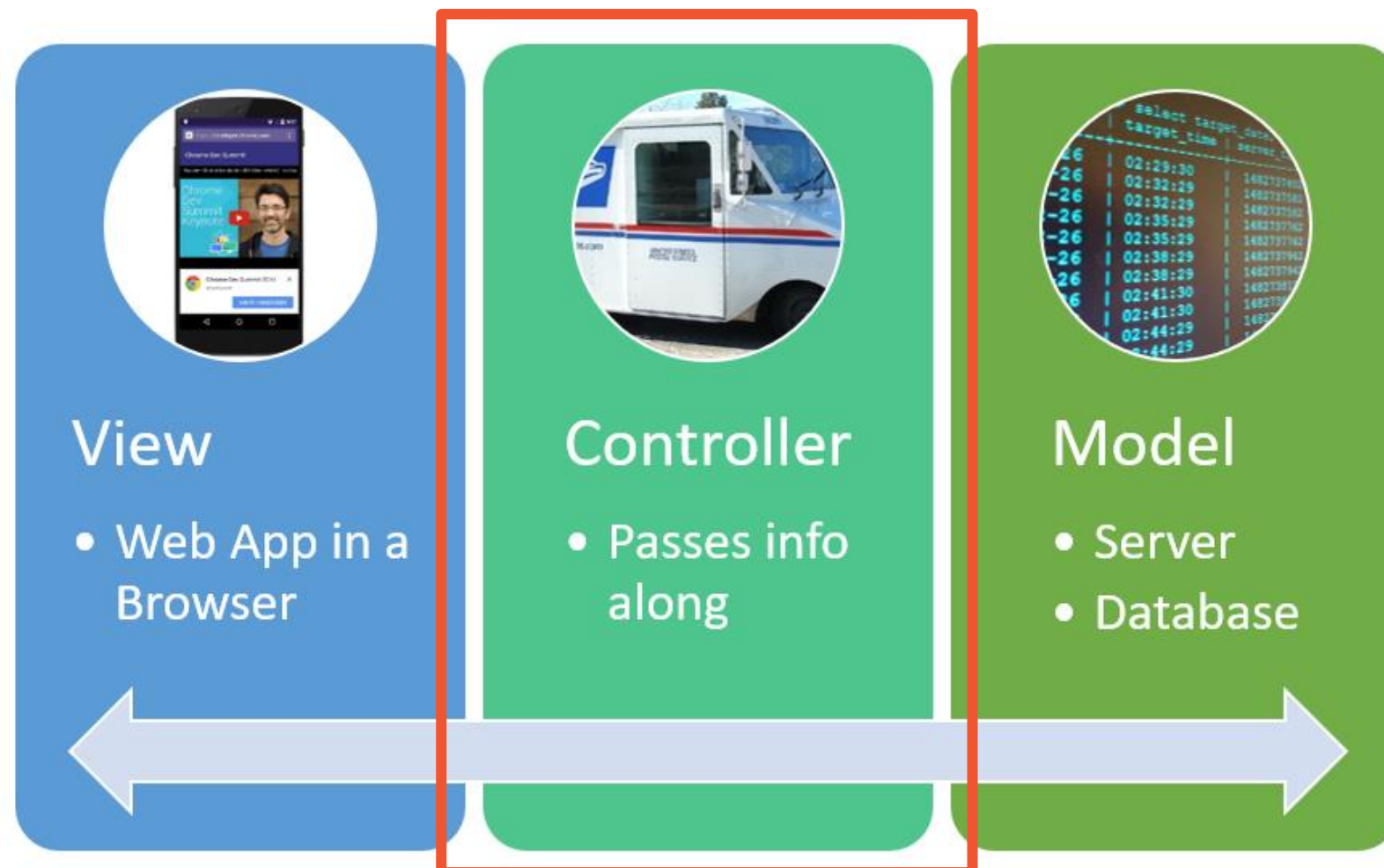
- **视图** [ビュー] 部分负责显示被传送到客户端的数据。它的主要职责就是把数据模型以用户看得懂的形式显示出来。此外，它也负责接受用户在图形界面上的**输入**，并把它们传递至控制器。





# MVC 组件：控制器

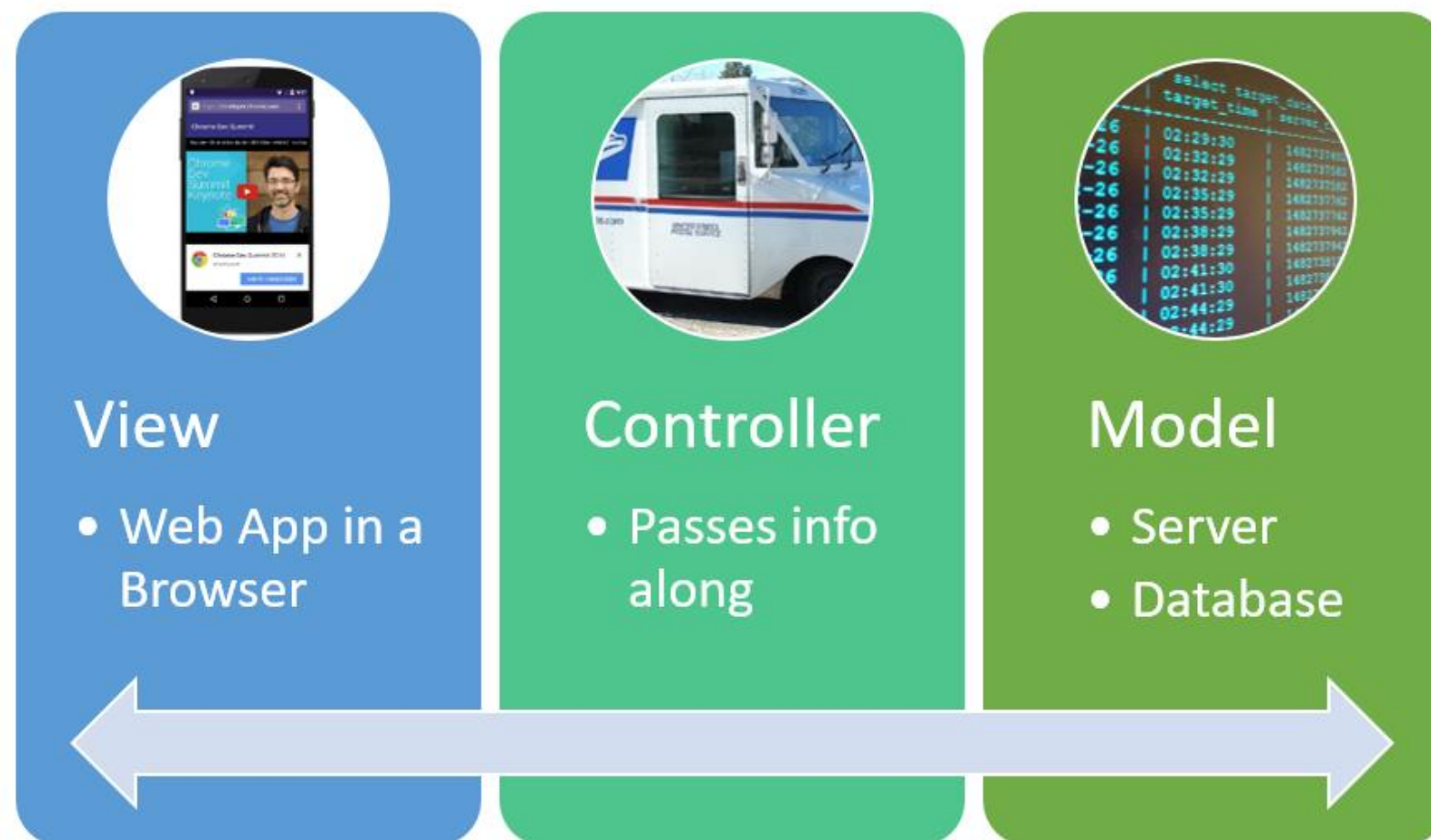
- **控制器**[コントローラー]部分负责组织不同部分间的交互，控制应用程序的主逻辑。它处理事件并作出响应。比如，在网络应用中，控制器一般部署在服务端：它需要处理前端传来的 HTTP 请求、产生相应的数据模型、最终将正确的响应发回前端。



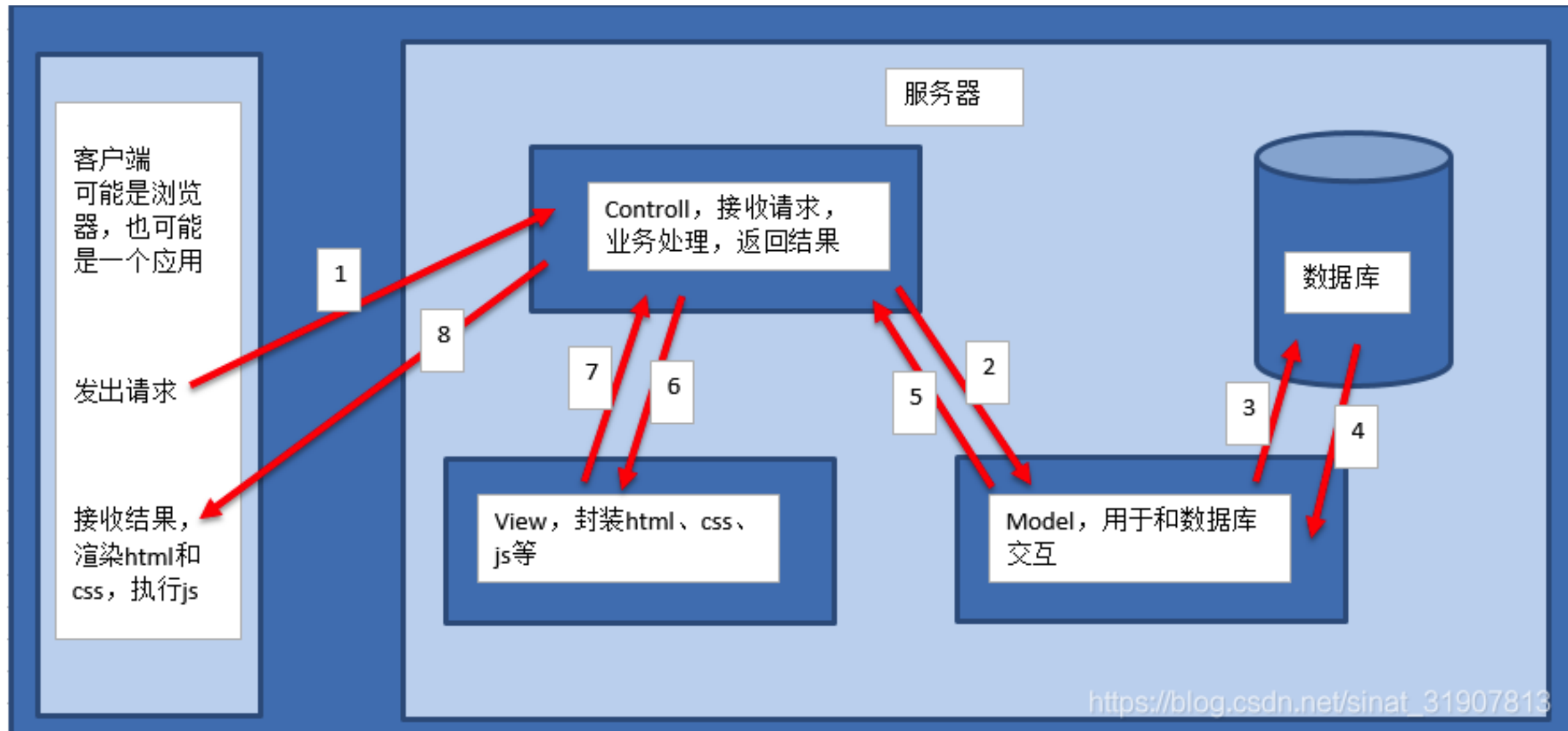


# MVC 总结

- Model 是项目的**数据模型**，View 是视图展示，而 Controller 则控制数据在这些部分之间的**传输**。



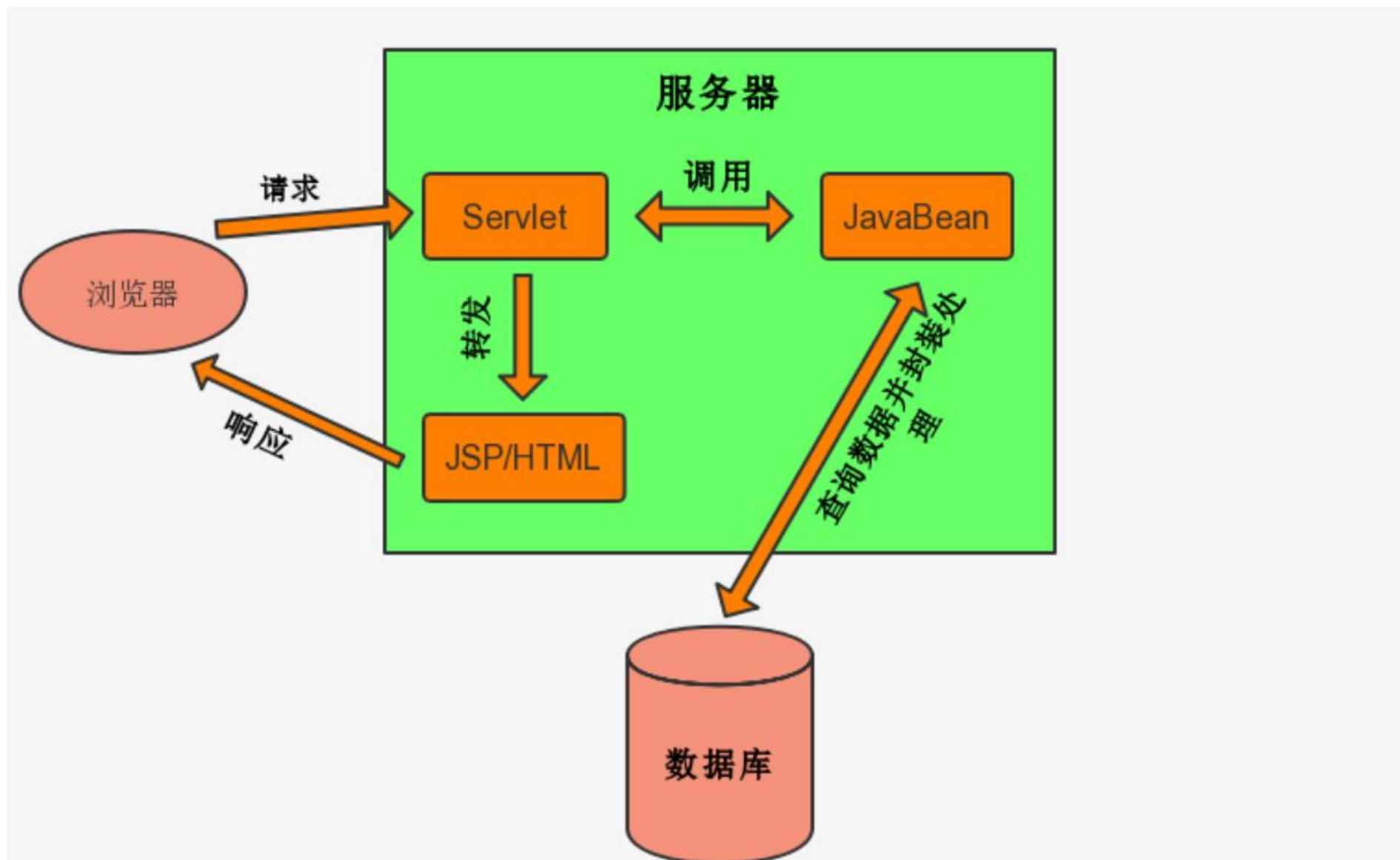
# MVC 流程





# 练习

- 这是一个基于 Servlet 实现的 MVC 架构模式图。在这个应用中，M、V、C 分别对应哪些部分？



Q & A

*Question and answer*



# 目录

1

MVC 模型

2

Spring 框架

3

Maven

# 开发框架

- **框架**[フレームワーク]一般指一种工具软件，在开发应用程序时预先提供常用的通用功能，作为实际应用程序开发的基础。
- 换句话说，当我们使用软件框架时，各个应用共通的部分从一开始就被创建好了。要创建一个特定的应用，我们只需要书写它和其它应用**不同**的部分。



# Struts





# 使用开发框架的好处

- 使用软件开发框架一般具有以下好处：
  - **提高生产力**：通过省略基础部分的开发，我们可以更集中于软件的主要逻辑，不必重复造轮子。
  - **同构开发**：通过遵循框架使用规则，我们可以开发出保持一定**质量水平**的应用程序。
  - **缩短测试过程**：某些功能框架已经帮我们实现并测试。我们**不需要再**对它们进行单元测试（ ➡ § 6.6.1 ）。
  - **可维护性提高**：由于软件是根据规则创建的，因此我们更容易掌握应用程序的整体情况，软件的**可维护性也会相应提高**。
- 当然，使用开发框架也有一定的缺陷，比如学习成本高、需要花时间选择框架、框架本身就存在错误等等。但整体来说还是利大于弊的。

# 常用开发框架

## 1. Spring

Spring 是专为 Java 设计的、开源的全栈[フルスタック]应用程序框架。Spring 框架的核心功能之一**控制反转**（Inversion of Control, IoC），理论上可用于任何 Java 应用，但 Spring 还为基于 Java 企业版平台构建的 Web 应用提供了大量的拓展支持。Spring 没有直接实现任何的编程模型，但它已经在 Java 社区中广为流行，成为目前最主流的框架之一。

## 2. Spring Boot

Spring Boot 是专门用于 Web 应用程序开发的 Spring 的派生框架。相较于 Spring 框架，Spring Boot 帮我们设置好了 Web 应用开发所需的配置文件，使我们能立刻开始简单的 Web 应用的开发。

## 3. Apache Struts

Struts 是自 2001 年以来一直在使用的 Java 中最著名的框架之一，是 Apache 软件基金会赞助的一个开源项目。近年来，由于其漏洞处理机制的不足，越来越多的用户正在迁移到别的框架。

接次页 



## 4. JSF

JSF (JavaServer Faces)，是 2004 年开发的 Java 标准框架，被 Java EE 规范采用。它采用类似于 *Apache Struts* 的 MVC 架构模式，但有一些差异，例如基于组件和事件驱动的、更接近传统静态网页开发的开发模式；XML 文件和 AJAX 技术的支持等。

## 5. Play Framework

Play Framework 是使用 Scala 语言开发的 Web 应用框架。因此，它不仅可以在 Java 中使用，还可以在 Scala 中使用。它深受 *Ruby on Rails* 和 *Django* 等其它框架的影响，是一个轻量级、高效的 Web 开发框架。

- 总结：除了上面介绍的框架之外，还有许多其他框架，也不存在所谓的“最好的框架”。每种框架各有各的特点，需要灵活地根据系统开发的目的是来决定选择哪个框架。

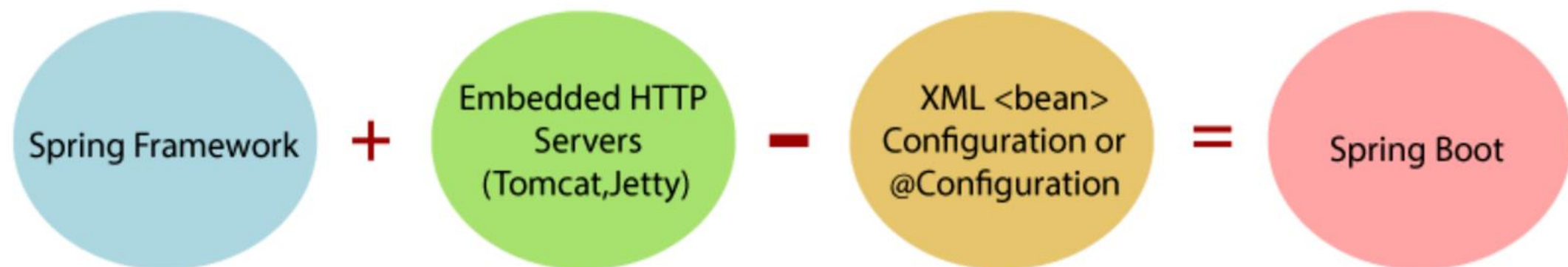
# Spring 框架

- **Spring** 是一个支持快速开发 Java EE 应用程序的框架。它提供了一系列底层容器和基础设施，并可以和大量常用的开源框架无缝集成。
- Spring 框架提供的一个基本的 MVC 框架，以便我们开发图形应用。除此之外，Spring 还提供了数据库支持、云计算、通信安全、图像处理等各种丰富的组件。



# Spring Boot

- 尽管 Spring 框架功能强大，但对一个最基本的 Web 应用来说能用到的功能有限。繁杂的组件和过于专业的服务器配置方式可能会使新开发者们望而却步。于是，Spring Boot 出现了。
- **Spring Boot** 是一个基于 Spring 的套件，它帮我们预组装了 Spring 的一系列组件，以便以**尽可能少**的代码和配置来开发基于 Spring 的 Java Web 应用程序。
- Spring Boot 的目标就是提供一个开箱即用的应用程序架构。我们可以基于 Spring Boot 的预置结构继续开发，省时省力。





# Spring Tool Suite

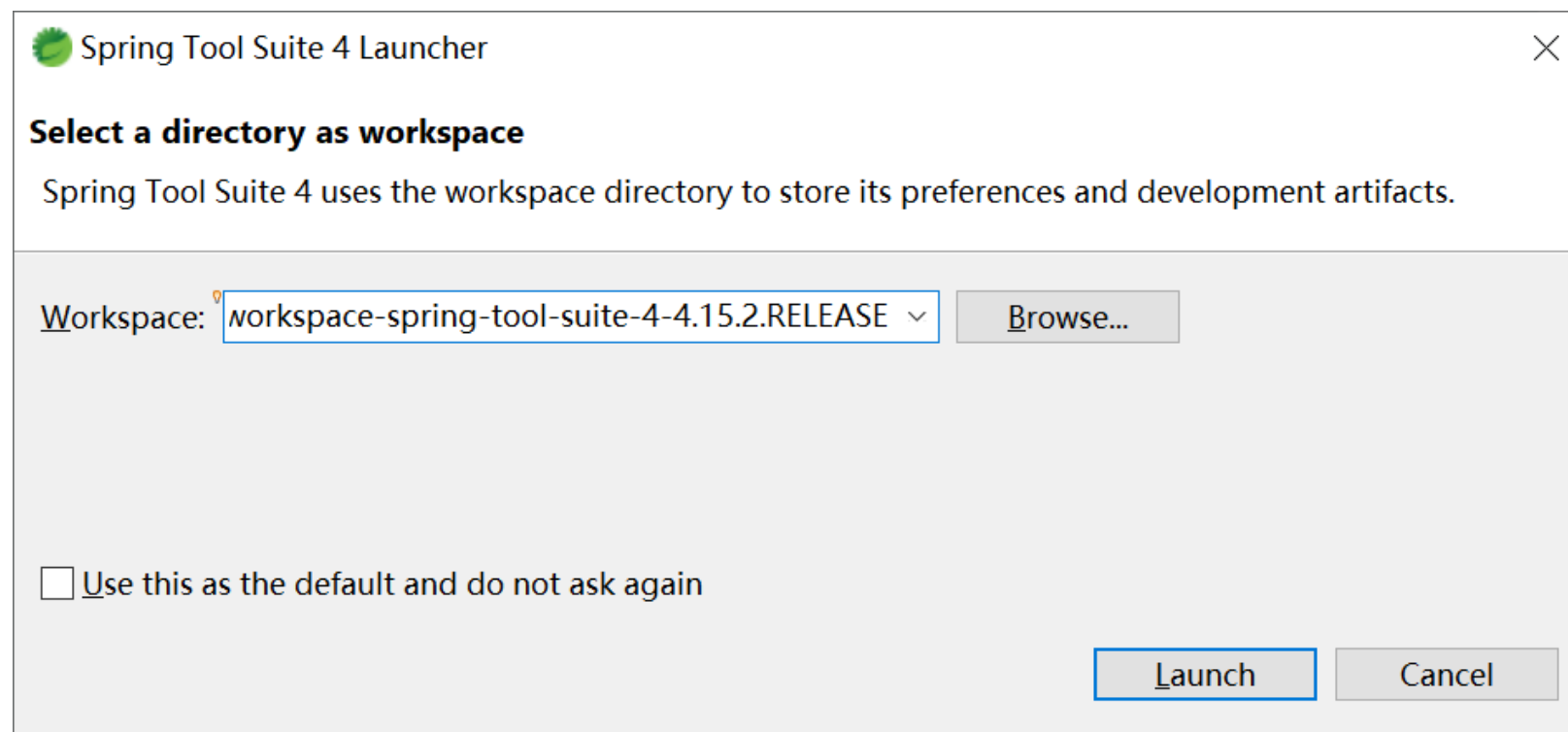
- **STS** (Spring Tool Suite 或 Spring Tools) 是一个 Spring 为 Spring Boot 开发者提供的类似 Eclipse 的集成开发环境。



- STS 的基本使用和 Eclipse 几乎完全一致，可以理解为是为了 Spring Boot 开发准备的“升级版”Eclipse。
- STS 无需专门安装，从官网  <https://spring.io/tools> 上下载对应的压缩包双击解压即可使用。

# 启动 Spring Tool Suite

- 解压后，你可以把整个文件夹（sts-版本号.RELEASE，如 **sts-4.15.2.RELEASE**）复制到你电脑上的任何位置，然后双击其中的 **SpringToolSuite4.exe** 启动。
- 初次启动时，会出现和 Eclipse 类似的工作区选择画面，根据需求选择即可。



Q & A

*Question and answer*



# 目录

1

MVC 模型

2

Spring 框架

3

Maven

# Java 项目构建的难题

- 让我们一起来考虑一下，开发一个完整的 Java 应用项目需要做哪些准备：
  1. 首先，我们需要确定导入哪些**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。
  2. 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。
  3. 此外，我们还需要配置一些和**构建流程**有关的信息，例如 JDK 的版本、编译打包的流程、当前代码的版本号等。
  4. 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，才能够保证项目能在一个独立的服务器上被编译、测试和部署。

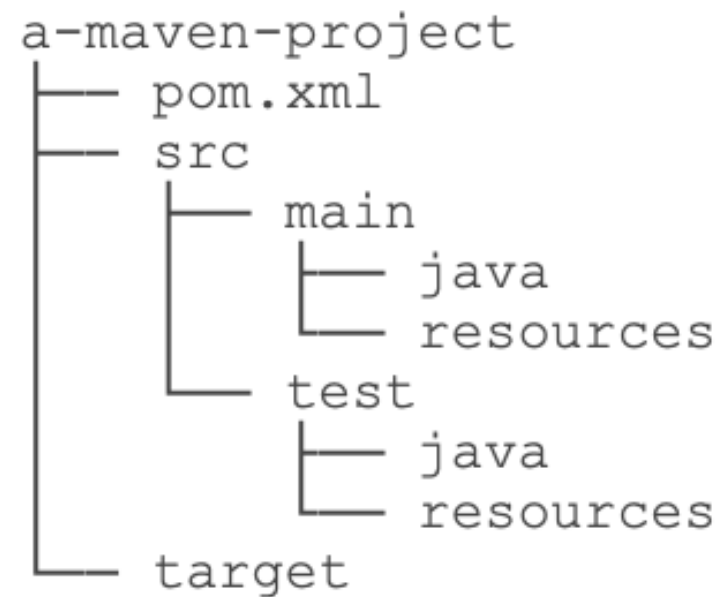
# Maven

- 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。Maven 的主要功能包括：
  - 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**。
- 我们接下来要创建的 Spring Boot 项目就是基于 Maven 进行管理的。让我们一起来了解 Maven 项目的结构和配置方法。



# Maven 项目结构

- 假定我们使用 Maven 管理的项目名为 a-maven-project, 那么它的目录结构默认如下:



- 其中:
  - pom.xml 是 Maven 项目的描述文件;
  - src/main/java 文件夹存放程序的 Java 源代码;
  - src/main/resources 文件夹存放各种资源文件;
  - src/test/java 文件夹存放测试源代码;
  - src/test/resources 文件夹存放测试资源;
  - target 文件夹所有编译、打包生成的文件。

# POM 文件

- **POM** (Project Object Model, 工程对象模型) 是使用 Maven 工作时的基本组件, 由一个 XML (一种语法类似 HTML 的文档标记语言) 文件进行描述。该文件被放在工程根目录下, 名为 **pom.xml**。
- POM 文件中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 中设置配置信息包括:
  - 项目基本信息, 如名称、版本等;
  - 项目作者 / 团队的相关信息;
  - 项目依赖信息;
  - 项目构建流程配置信息;
  - 其它插件信息等。

# pom.xml 的例子

- 一个典型的 pom.xml 文件结构如下：

```
1 <project ...>
2   <modelVersion>4.0.0</modelVersion>
3
4   <groupId>com.lighthouseit.java</groupId>
5   <artifactId>hello</artifactId>
6   <version>1.0</version>
7   <packaging>jar</packaging>
8
9   <properties> ... </properties>
10
11  <dependencies>
12    <dependency>
13      <groupId>javax.servlet</groupId>
14      <artifactId>javax.servlet-api</artifactId>
15      <version>3.1.0</version>
16    </dependency>
17    ...
18  </dependencies>
19 </project>
```

POM 版本号（一般都是 4.0.0）

项目基本信息

Maven 的配置信息

项目依赖信息



# 项目基本信息

- 一些最重要的项目基本信息如下：
  - `groupId`: 开发公司或组织的名称, 类似于 Java 的包名。
  - `artifactId`: 项目名称, 类似于Java的类名。
  - `version`: 项目版本号。
- 每一个 Maven 工程可以通过 **`groupId`**、**`artifactId`** 和 **`version`** 作为唯一标识确定下来。

# 依赖管理

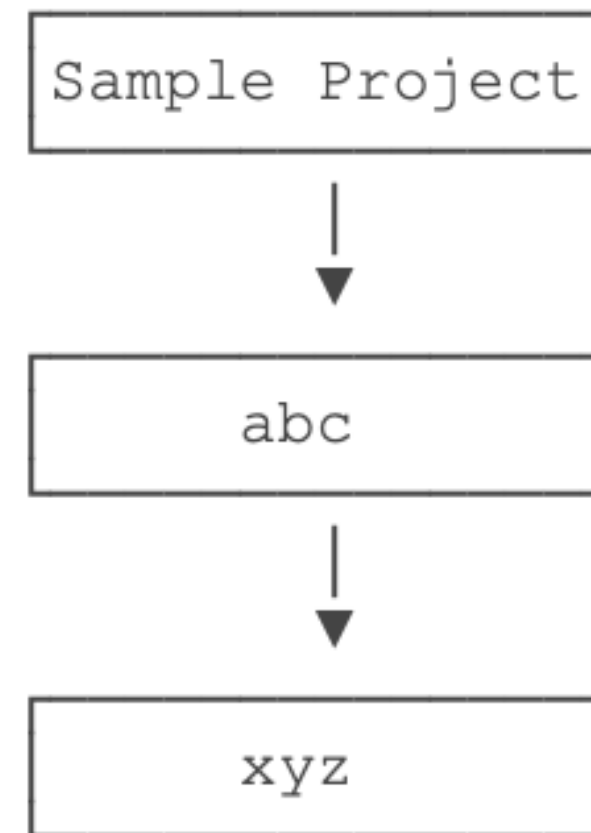
- 我们在引用其他第三方库的时候，也是通过这 3 个信息确定。例如，依赖 *Servlet*:

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

- 使用 `<dependency>` 标签声明一个依赖后，Maven 就会自动下载这个项目的 Jar 包并把它放到 classpath 中。

# 依赖的传递

- Maven 简化了我们管理依赖的流程。例如，我们的项目依赖 `abc` 这个 Jar 包，而 `abc` 又依赖 `xyz` 这个 Jar 包：
- 此时，我们的项目便需要依赖 `xyz` 包。这就是依赖的**传递**：
- 当我们声明了 `abc` 的依赖时，Maven 会自动把 `abc` 和 `xyz` 都加入我们的项目依赖，不需要我们自己去研究 `abc` 是否需要依赖 `xyz`。
- 当然，如果 `abc` 依赖别的 Jar 包，或是 `xyz` 依赖了某些 Jar 包，Maven 都会自动帮我们确认并下载。





# 依赖传递的例子

- 来看一个具体的例子：

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-web</artifactId>
4   <version>1.4.2.RELEASE</version>
5 </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约二、三十个其他依赖：

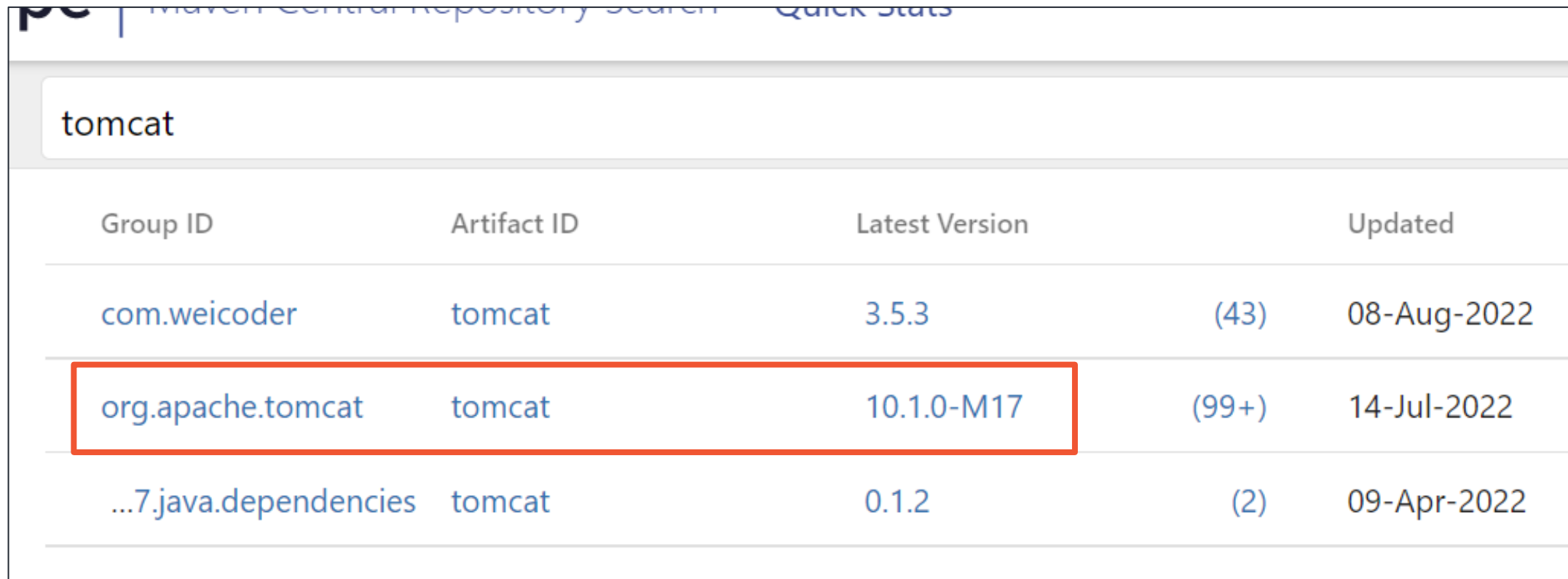
```
spring-boot-starter-web
spring-boot-starter
spring-boot
spring-boot-autoconfigure
spring-boot-starter-logging
logback-classic
logback-core
slf4j-api
jcl-over-slf4j
slf4j-api
```

```
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
...
```

- 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。

# 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？
- 我们可以在网站  <https://search.maven.org> 上搜索关键字，以找到对应的依赖的 3 个信息：



The screenshot shows the Maven Central Repository search results for the keyword 'tomcat'. The search bar at the top contains 'tomcat'. Below the search bar, there is a table with the following columns: Group ID, Artifact ID, Latest Version, (number of artifacts), and Updated. The table lists several results, with the first three being highlighted by a red box.

| Group ID               | Artifact ID | Latest Version |       | Updated     |
|------------------------|-------------|----------------|-------|-------------|
| com.weicoder           | tomcat      | 3.5.3          | (43)  | 08-Aug-2022 |
| org.apache.tomcat      | tomcat      | 10.1.0-M17     | (99+) | 14-Jul-2022 |
| ...7.java.dependencies | tomcat      | 0.1.2          | (2)   | 09-Apr-2022 |

Q & A

*Question and answer*



# 总结

## Sum Up

1. MVC 架构模式的概念：
  - ① Model 指数据模型；
  - ② View 指视图显示；
  - ③ Controller 指控制逻辑。
2. 软件开发框架：Spring 和 Spring Boot 框架。
3. Maven 的基本使用方法。

**THANK YOU!**