

第一章 Spring 概述

1. spring 是什么

Spring 是分层的 Java SE/EE 应用 **full-stack** 轻量级开源框架，以 IoC（Inverse Of Control：反转控制）和 AOP（Aspect Oriented Programming：面向切面编程）为内核，提供了展现层 SpringMVC 和持久层 Spring JDBC 以及业务层事务管理等众多的企业级应用技术，还能整合开源世界众多著名的第三方框架和类库，逐渐成为使用最多的 Java EE 企业应用开源框架。

Spring 的发展历程

1997 年 IBM 提出了 EJB 的思想

1998 年，SUN 制定开发标准规范 EJB1.0

1999 年，EJB1.1 发布

2001 年，EJB2.0 发布

2003 年，EJB2.1 发布

2006 年，EJB3.0 发布

Rod Johnson (spring 之父)

Expert One-to-One J2EE Design and Development (2002)

阐述了 J2EE 使用 EJB 开发设计的优点及解决方案

Expert One-to-One J2EE Development without EJB (2004)

阐述了 J2EE 开发不使用 EJB 的解决方式 (Spring 雏形)

2017 年 9 月份发布了 spring 的最新版本 **spring 5.0 通用版 (GA)**

2. spring 的优势

方便解耦，简化开发

通过 Spring 提供的 IoC 容器，可以将对象间的依赖关系交由 Spring 进行控制，避免硬编码所造成的过度程序耦合。用户也不必再为单例模式类、属性文件解析等这些很底层的需求编写代码，可以更专注于上层的应用。

AOP 编程的支持

通过 Spring 的 AOP 功能，方便进行面向切面的编程，许多不容易用传统 OOP 实现的功能可以通过 AOP 轻松应付。

声明式事务的支持

可以将我们从单调烦闷的事务管理代码中解脱出来，通过声明式方式灵活的进行事务的管理，提高开发效率和质量。

方便程序的测试

可以用非容器依赖的编程方式进行几乎所有的测试工作，测试不再是昂贵的操作，而是

随手可做的事情。

方便集成各种优秀框架

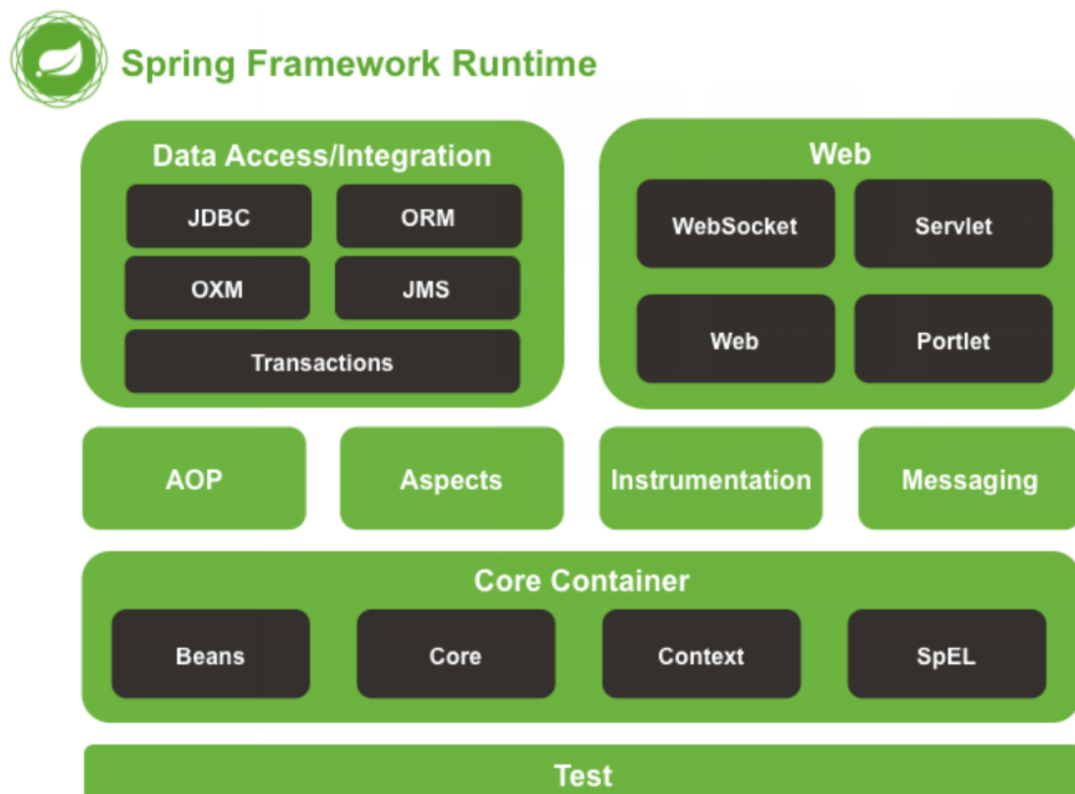
Spring 可以降低各种框架的使用难度，提供了对各种优秀框架（ Struts、 Hibernate、 Hessian、 Quartz 等）的直接支持。降低 **JavaEE API** 的使用难度

Spring 对 JavaEE API（如 JDBC、 JavaMail、远程调用等）进行了薄薄的封装层，使这些 API 的使用难度大为降低。

Java 源码是经典学习范例

Spring 的源代码设计精妙、结构清晰、匠心独用，处处体现着大师对 Java 设计模式灵活运用以及对 Java 技术的高深造诣。它的源代码无意是 Java 技术的最佳实践的范例。

3. spring 的体系结构



第二章 SpringMVC 简介

MVC 全名是 Model View Controller，是模型(model) – 视图(view) – 控制器(controller)的缩写，是一种用于设计创建 Web 应用程序表现层的模式。MVC 中每个部分各司其职：

Model（模型）：

通常指的就是我们的数据模型。作用一般情况下用于封装数据。

View（视图）：

通常指的就是我们的 jsp 或者 html。作用一般就是展示数据的。

通常视图是依据模型数据创建的。

Controller（控制器）：

是应用程序中处理用户交互的部分。作用一般就是处理程序逻辑的。

它相对于前两个不是很好理解，这里举个例子：

例如：

我们要保存一个用户的信息，该用户信息中包含了姓名，性别，年龄等等。

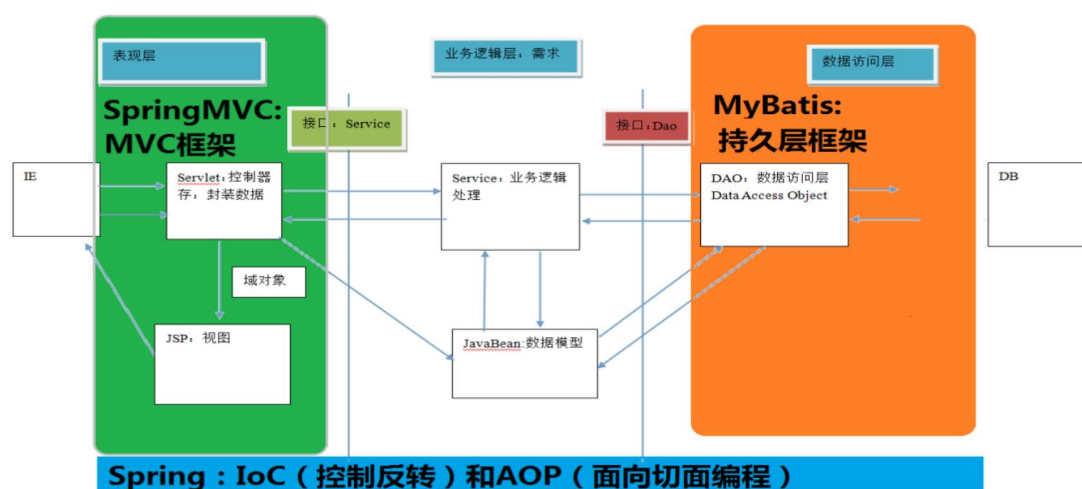
这时候表单输入要求年龄必须是 1~100 之间的整数。姓名和性别不能为空。并且把数据填充到模型之中。此时除了 js 的校验之外，服务器端也应该有数据准确性的校验，那么校验就是控制器的该做的。当校验失败后，由控制器负责把错误页面展示给使用者。如果校验成功，也是控制器负责把数据填充到模型，并且调用业务层实现完整的业务需求。

1. SpringMVC 是什么

SpringMVC 是一种基于 Java 的实现 MVC 设计模型请求驱动类型的轻量级 Web 框架，属于 SpringFramework 的后续产品，已经融合在 SpringWeb Flow 里面。Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。使用 Spring 可插入的 MVC 架构，从而在使用 Spring 进行 WEB 开发时，可以选择使用 Spring 的 Spring MVC 框架或集成其他 MVC 开发框架，如 Struts1(现在一般不用)，Struts2 等。

SpringMVC 已经成为目前最主流的 MVC 框架之一，并且随着 Spring3.0 的发布，全面超越 Struts2，成为最优秀的 MVC 框架。它通过一套注解，让一个简单的 Java 类成为处理请求的控制器，而无须实现任何接口。同时它还支持 RESTful 编程风格的请求。

2. SpringMVC 三层架构



3. SpringMVC 的优势

1、清晰的角色划分：

前端控制器 (DispatcherServlet)

请求到处理器映射 (HandlerMapping)

处理器适配器 (HandlerAdapter)

视图解析器 (ViewResolver)

处理器或页面控制器 (Controller)

验证器 (Validator)

命令对象 (Command 请求参数绑定到的对象就叫命令对象)

表单对象 (Form Object 提供给表单展示和提交到的对象就叫表单对象)。

2、分工明确，而且扩展点相当灵活，可以很容易扩展，虽然几乎不需要。

3、由于命令对象就是一个 POJO，无需继承框架特定 API，可以使用命令对象直接作为业务对象。

4、和 Spring 其他框架无缝集成，是其它 Web 框架所不具备的。

5、可适配，通过 HandlerAdapter 可以支持任意的类作为处理器。

6、可定制性，HandlerMapping、ViewResolver 等能够非常简单的定制。

7、功能强大的数据验证、格式化、绑定机制。

8、利用 Spring 提供的 Mock 对象能够非常简单的进行 Web 层单元测试。

9、本地化、主题的解析的支持，使我们更容易进行国际化和主题的切换。

10、强大的 JSP 标签库，使 JSP 编写更容易。

还有比如 RESTful 风格的支持、简单的文件上传、约定大于配置的契约式编程支持、基于注解的零配置支持等等。

4. SpringMVC 的组件

DispatcherServlet：前端控制器

用户请求到达前端控制器，它就相当于 mvc 模式中的 c，dispatcherServlet 是整个流程控制的中心，由

它调用其它组件处理用户的请求，dispatcherServlet 的存在降低了组件之间的耦合性。

HandlerMapping：处理器映射器

HandlerMapping 负责根据用户请求找到 Handler 即处理器，SpringMVC 提供了不同的映射器实现不同的

映射方式，例如：配置文件方式，实现接口方式，注解方式等。

Handler：处理器

它就是我们开发中要编写的具体业务控制器。由 DispatcherServlet 把用户请求转发到 Handler。由 Handler 对具体的用户请求进行处理。

HandlerAdapter：处理器适配器

通过 HandlerAdapter 对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

View Resolver：视图解析器

View Resolver 负责将处理结果生成 View 视图，View Resolver 首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成 View 视图对象，最后对 View 进行渲染将处理结果通过页面展示给用户。

View：视图

SpringMVC 框架提供了很多的 View 视图类型的支持，包括：jstlView、freemarkerView、pdfView 等。我们最常用的视图就是 jsp。一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

<mvc:annotation-driven> 说明

在 SpringMVC 的各个组件中，处理器映射器、处理器适配器、视图解析器称为 SpringMVC 的三大组件。使用 <mvc:annotation-driven> 自动加载 RequestMappingHandlerMapping（处理映射器）和 RequestMappingHandlerAdapter（处理适配器），可用在 SpringMVC.xml 配置文件中使用 <mvc:annotation-driven> 替代注解处理器和适配器的配置。

第三章 Mybatis 简介

Mybatis 是一个优秀的基于 java 的持久层框架，它内部封装了 jdbc，使开发者只需要关注 sql 语句本身，而不需要花费精力去处理加载驱动、创建连接、创建 statement 等繁杂的过程。

Mybatis 通过 xml 或注解的方式将要执行的各种 statement 配置起来，并通过 java 对象和 statement 中 sql 的动态参数进行映射生成最终执行的 sql 语句，最后由 mybatis 框架执行 sql 并将结果映射为 java 对象并返回。

采用 ORM 思想解决了实体和数据库映射的问题，对 jdbc 进行了封装，屏蔽了 jdbc api 底层访问细节，使我们不用与 jdbc api 打交道，就可以完成对数据库的持久化操作。

Mybatis 的功能架构分为三层：

API 接口层：提供给外部使用的接口 API，开发人员通过这些本地 API 来操纵数据库。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理。

数据处理层：负责具体的 SQL 查找、SQL 解析、SQL 执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作。

基础支撑层：负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这

些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑。

MyBatis 的优缺点

优点：

- 简单易学：本身就很小且简单。没有任何第三方依赖，最简单安装只要两个 jar 文件+配置几个 sql 映射文件易于学习，易于使用，通过文档和源代码，可以比较完全的掌握它的设计思路和实现。
- 灵活：mybatis 不会对应用程序或者数据库的现有设计强加任何影响。sql 写在 xml 里，便于统一管理和优化。通过 sql 基本上可以实现我们不使用数据访问框架可以实现的所有功能，或许更多。
- 解除 sql 与程序代码的耦合：通过提供 DAL 层，将业务逻辑和数据访问逻辑分离，使系统的设计更清晰，更易维护，更易单元测试。sql 和代码的分离，提高了可维护性。
- 提供映射标签，支持对象与数据库的 orm 字段关系映射
- 提供对象关系映射标签，支持对象关系组建维护
- 提供 xml 标签，支持编写动态 sql。

缺点：

- 编写 SQL 语句时工作量很大，尤其是字段多、关联表多时，更是如此。
- SQL 语句依赖于数据库，导致数据库移植性差，不能更换数据库。
- 框架还是比较简陋，功能尚有缺失，虽然简化了数据绑定代码，但是整个底层数据库查询实际还是要自己写的，工作量也比较大，而且不太容易适应快速数据库修改。
- 二级缓存机制不佳

第四章 SSM 整合

SSM（Spring+SpringMVC+MyBatis）框架集由 Spring、MyBatis 两个开源框架整合而成（SpringMVC 是 Spring 中的部分内容）。常作为数据源较简单的 web 项目的框架。

整合步骤：

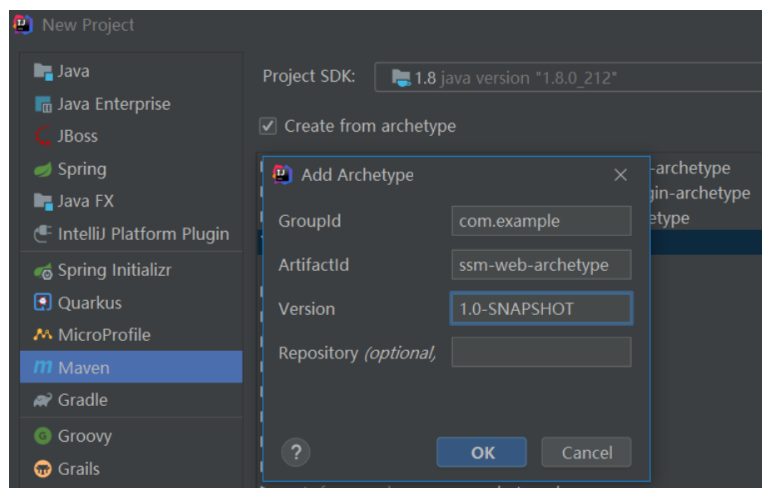
1. Maven 引入需要的 JAR 包
2. 配置 SpringMVC
3. Spring 与 MyBatis 的整合

这里我们使用自定义的脚手架(项目模板)创建项目



ssm-web-archetype.zip

放到 本地仓库 repository\com\example 下解压 然后使它创建项目.



第五章 SpringBoot

1. Spring 的缺点分析

虽然 Spring 的组件代码是轻量级的，但它的配置却是重量级的。一开始，Spring 用 XML 配置，而且是很多 XML 配置。Spring 2.5 引入了基于注解的组件扫描，这消除了大量针对应用程序自身组件的显式 XML 配置。Spring 3.0 引入了基于 Java 的配置，这是一种类型安全的可重构配置方式，可以代替 XML。

所有这些配置都代表了开发时的损耗。因为在思考 Spring 特性配置和解决业务问题之间需要进行思维切换，所以编写配置挤占了编写应用程序逻辑的时间。和所有框架一样，Spring 实用，但与此同时它要求的回报也不少。

除此之外，项目的依赖管理也是一件耗时耗力的事情。在环境搭建时，需要分析要导入哪些库的坐标，而且还需要分析导入与之有依赖关系的其他库的坐标，一旦选错了依赖的版本，随之而来的不兼容问题就会严重阻碍项目的开发进度。

SpringBoot 对上述 Spring 的缺点进行的改善和优化，基于约定优于配置的思想，可以让开发人员不必在配置与逻辑业务之间进行思维的切换，全身心的投入到逻辑业务的代码编写中，从而大大提高了开发的效率，一定程度上缩短了项目周期。

2. SpringBoot 的特点

- 为基于 Spring 的开发提供更快入门体验
- 开箱即用，没有代码生成，也无需 XML 配置。同时也可以修改默认值来满足特定的需求
- 提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标、健康检测、外部配置等
- SpringBoot 不是对 Spring 功能上的增强，而是提供了一种快速使用 Spring 的方式

3. SpringBoot 的核心功能

- 启动器依赖

启动器依赖本质上是一个 Maven 项目对象模型（Project Object Model, POM），定义了对其他库的传递依赖，这些东西加在一起即支持某项功能。

简单的说，启动器依赖就是将具备某种功能的坐标打包到一起，并提供一些默认的功能。

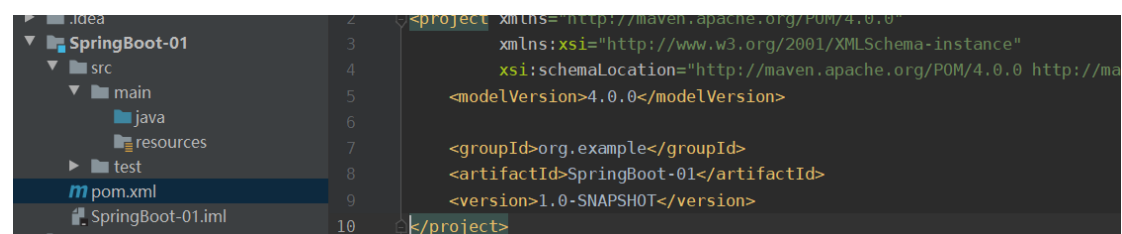
- 自动配置

Spring Boot 的自动配置是一个运行时（更准确地说，是应用程序启动时）的过程，考虑了众多因素，才决定 Spring 配置应该用哪个，不该用哪个。该过程是 Spring 自动完成的。

4. SpringBoot 快速入门

● 创建 Maven 工程

使用 idea 工具创建一个 maven 工程，该工程为普通的 java 工程即可



● 添加 SpringBoot 的启动器依赖

SpringBoot 要求，项目要继承 SpringBoot 的启动器依赖 spring-boot-starter-parent


```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.3.RELEASE</version>
</parent>
```

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.3.RELEASE</version>
</parent>
```

SpringBoot 要集成 SpringMVC 进行 Controller 的开发，所以项目要导入 web 的启动依赖

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

● 编写 SpringBoot 引导类

要通过 SpringBoot 提供的引导类启动器 SpringBoot 才可以进行访问

```
@RestController
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @RequestMapping("/")
    String home() {
        return "Hello World!";
    }
}
```

- 执行 SpringBoot 启动器类的主方法，控制台打印日志如下：

```

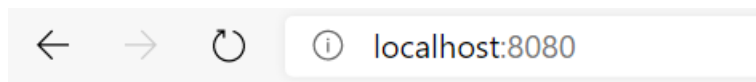
      ^__^
      (oo)\_______
      (__)\       )\/\
      ||----w |
      ||     ||

:: Spring Boot ::                (v2.3.3.RELEASE)

2020-08-20 17:27:33.794 INFO 17176 --- [           main] com.example.Application : Starting Application on LAPTOP-A0KG03MV with PID 17176

```

打开浏览器访问 url 地址为: **http://localhost:8080/**



Hello World!

代码解析:

@SpringBootApplication 注释可用于启用这三个功能，即：

@EnableAutoConfiguration: 启用 Spring Boot 的自动配置机制

@ComponentScan: 启用@Component 对应用程序所在的软件包的扫描

@Configuration: 允许在上下文中注册额外的 bean 或导入其他配置类

@RestController 注解告诉 Spring 使得到的字符串直接返回给调用者

@RequestMapping 注释提供“路由”的信息。它告诉 Spring 任何具有/路径的 HTTP 请求都应映射到该 home 方法

- ## ● SpringBoot 工程热部署

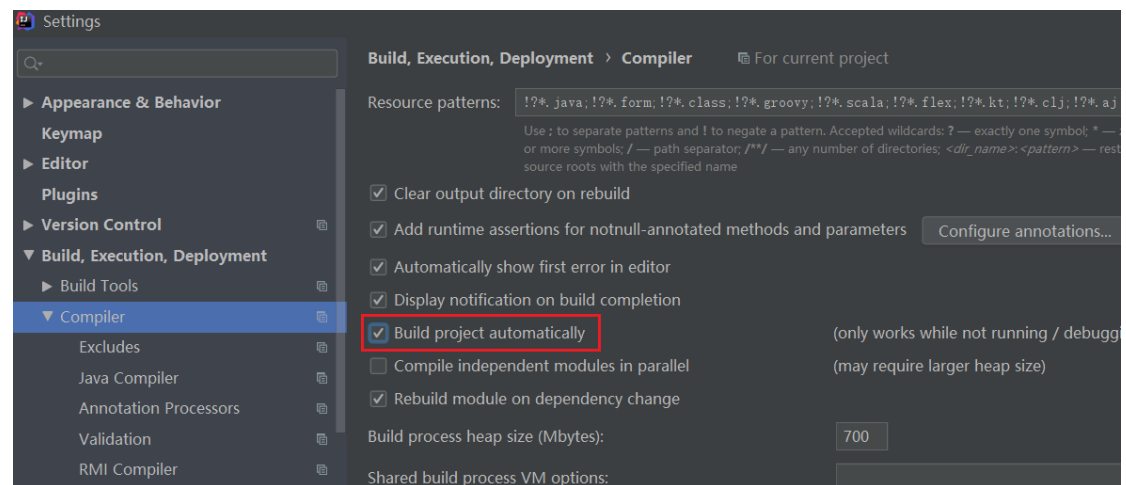
我们在开发中反复修改类、页面等资源，每次修改后都是需要重新启动才生效，这样每次启动都很麻烦，浪费了大量的时间，我们可以在修改代码后不重启就能生效，在 pom.xml 中添加如下配置就可以实现这样的功能，我们称之为热部署。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <optional>true</optional>
</dependency>
```

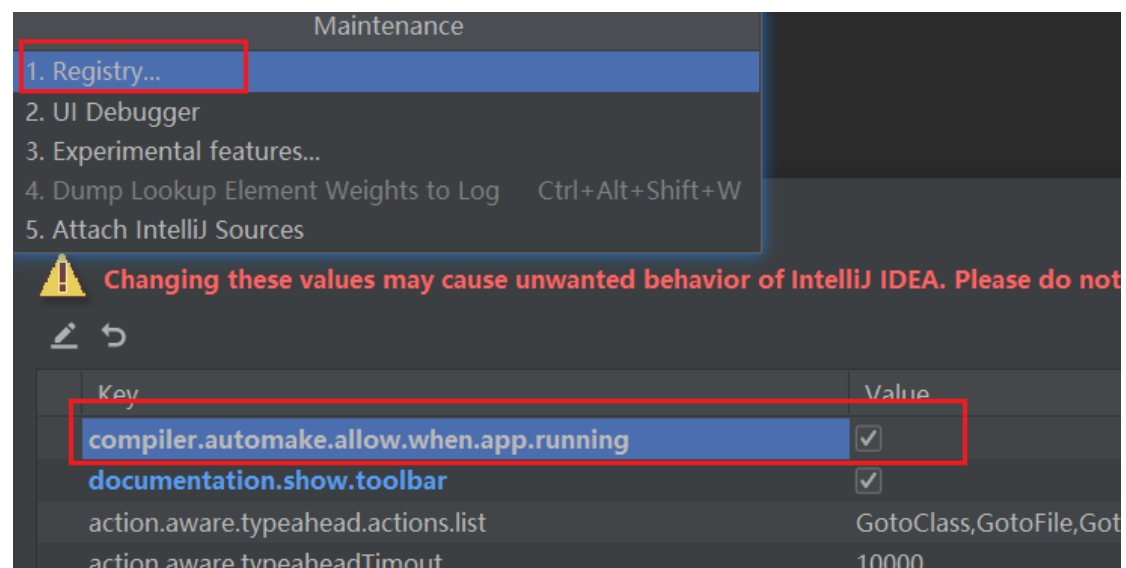
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <optional>true</optional>
</dependency>
```

注意：IDEA 进行 SpringBoot 热部署失败原因

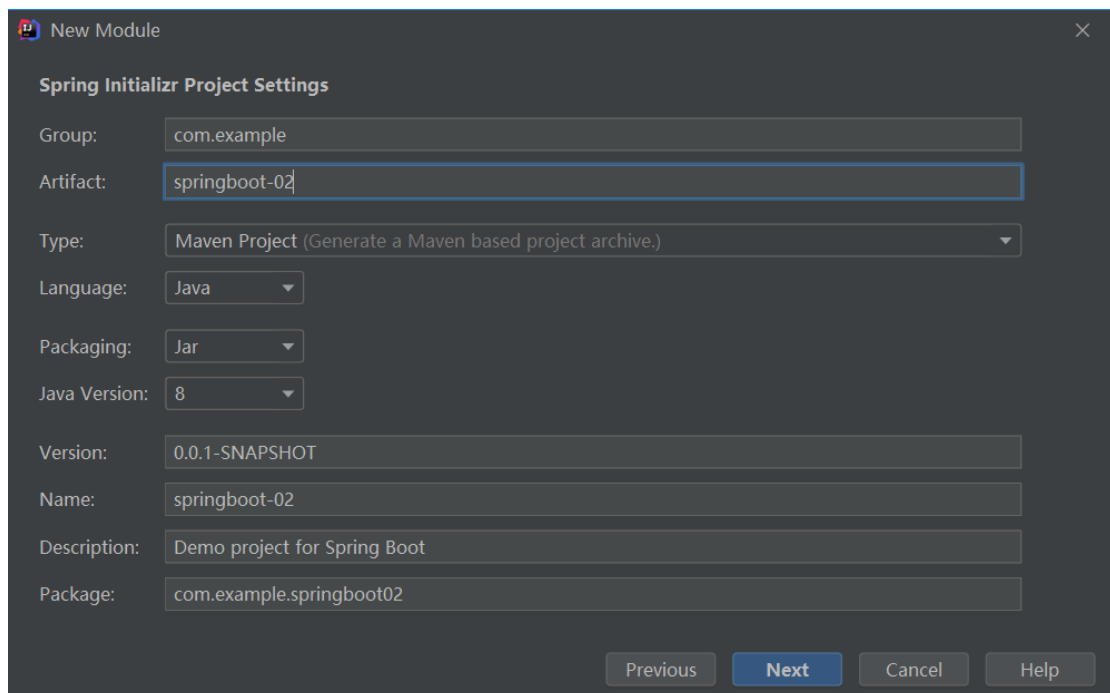
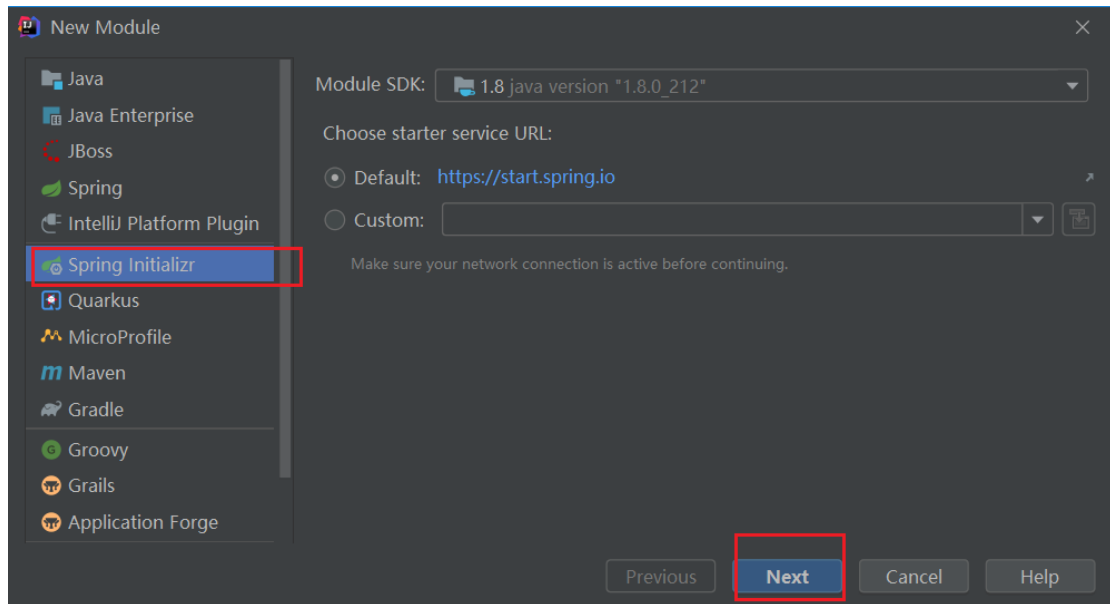
出现这种情况，并不是热部署配置问题，其根本原因是因为 IntelliJ IDEA 默认情况下不会自动编译，需要对 IDEA 进行自动编译的设置，如下：

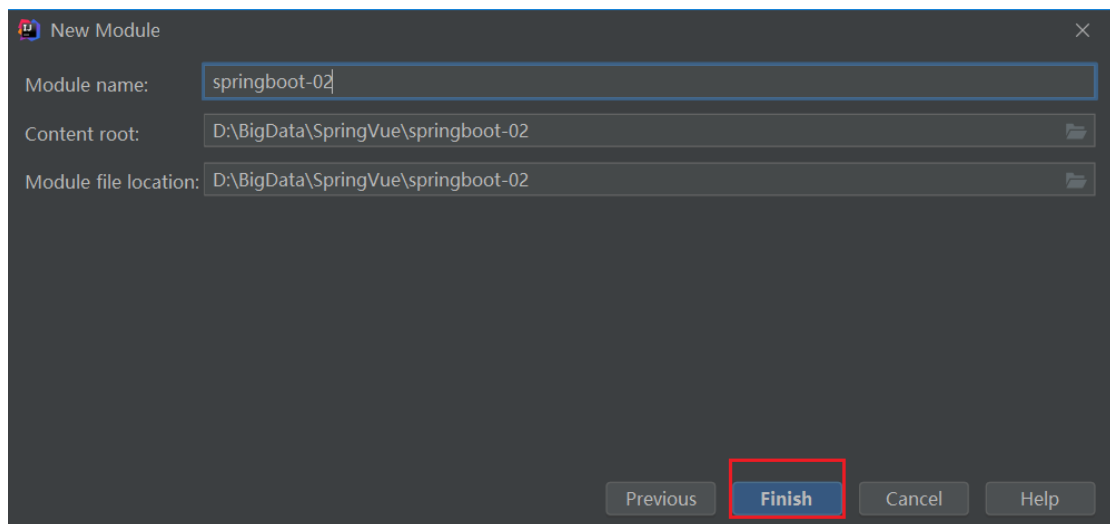
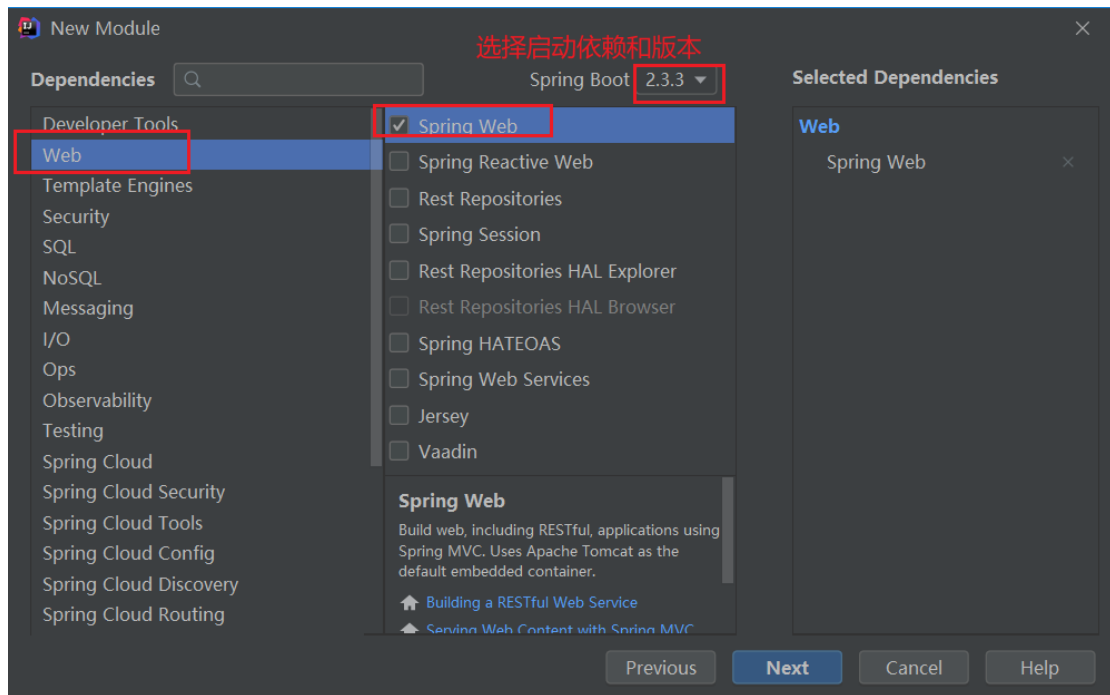


然后 Shift+Ctrl+Alt+/, 选择 Registry

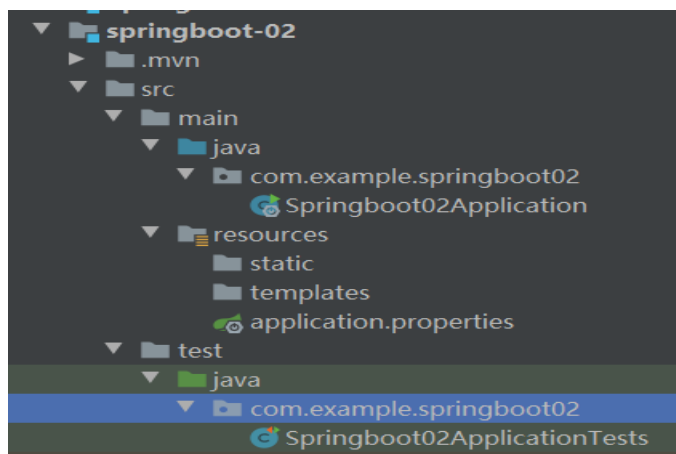


5. 使用 idea 快速创建 SpringBoot 项目





点击 Finish 完成 自动生成的模块目录结构:



通过 idea 快速创建的 SpringBoot 项目的 pom.xml 中已经导入了我们选择的 web 的启动器依赖的坐标

```
pom.xml (springboot-02) x
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/POM/4.0.0 ht
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 ht
    <modelVersion>4.0.0</modelVersion>
    <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.3.3.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>springboot-02</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot-02</name>
    <description>Demo project for Spring Boot</description>

    <properties>
      <java.version>1.8</java.version>
    </properties>

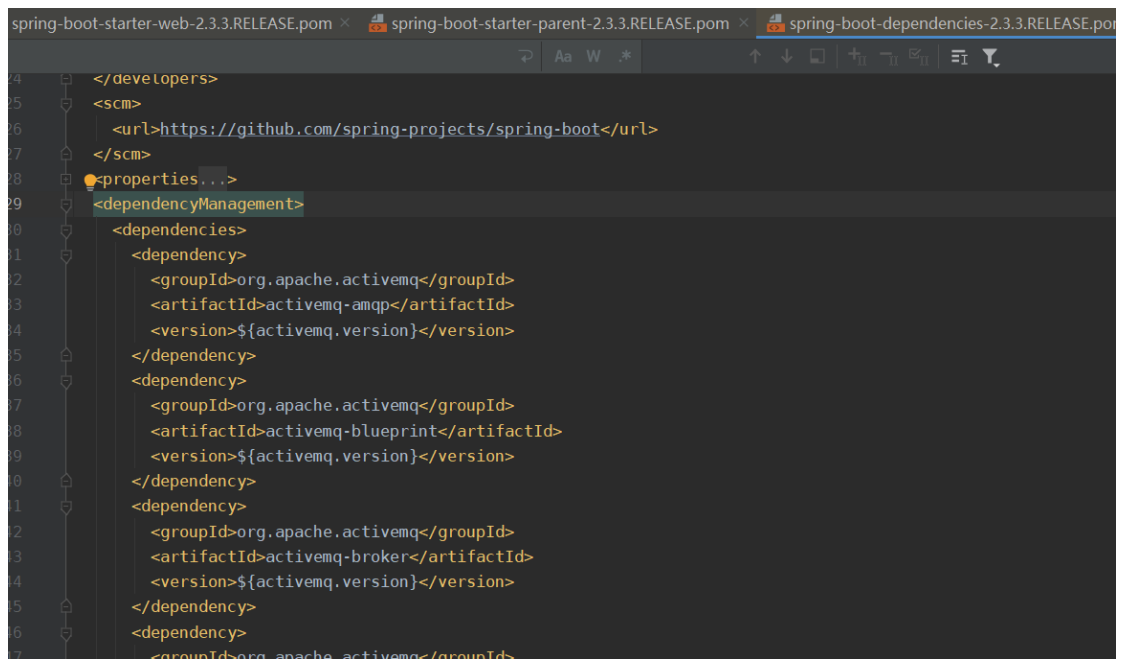
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
```

可以使用快速入门的方式创建 Controller 进行访问，此处不再赘述

6. SpringBoot 原理分析

启动器依赖原理分析

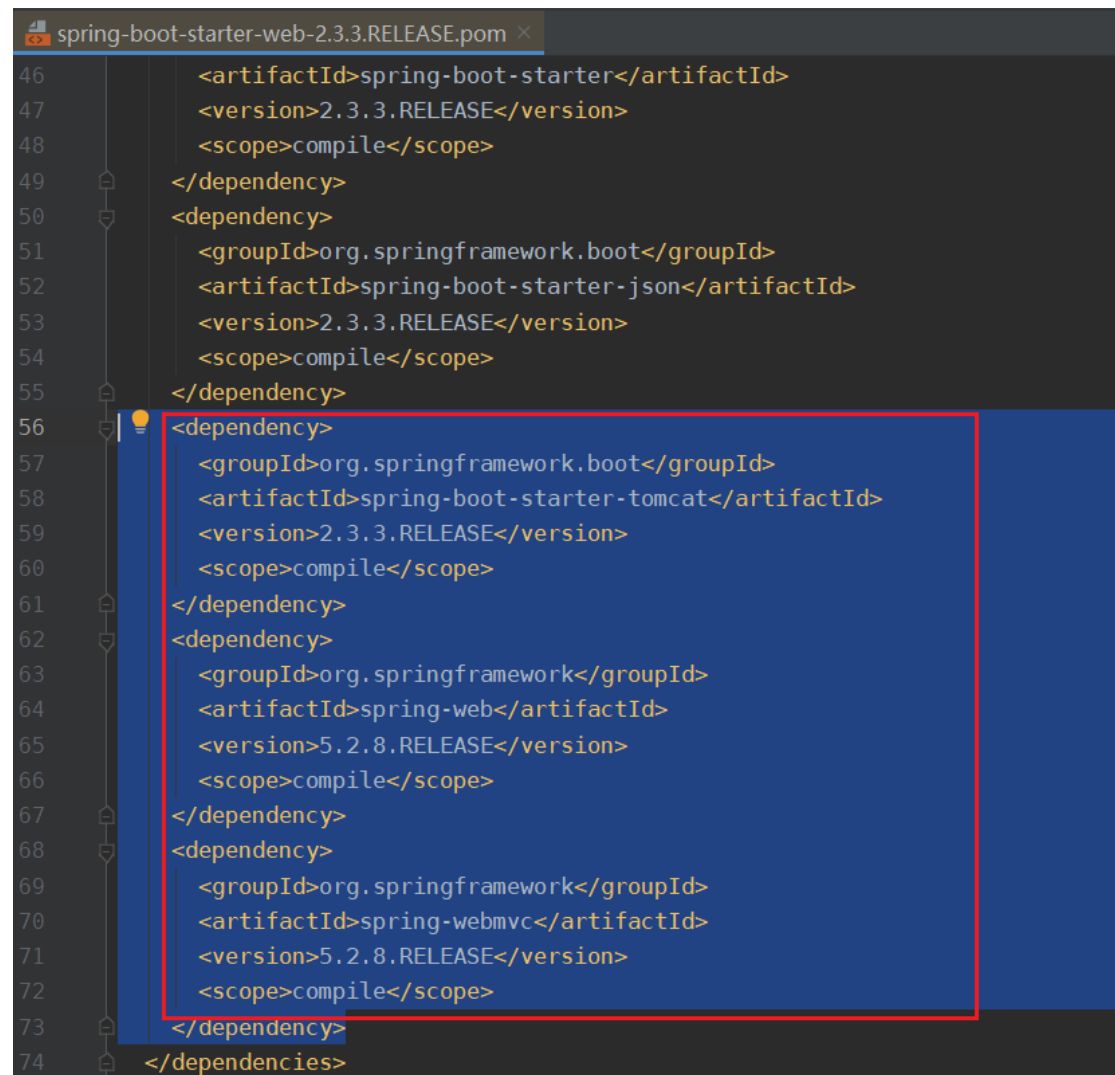
依次点开 POM 配置中的 parent 坐标



```
spring-boot-starter-web-2.3.3.RELEASE.pom x spring-boot-starter-parent-2.3.3.RELEASE.pom x spring-boot-dependencies-2.3.3.RELEASE.pom
4  </developers>
5  <scm>
6    <url>https://github.com/spring-projects/spring-boot</url>
7  </scm>
8  <properties...>
9  <dependencyManagement>
10 <dependencies>
11 <dependency>
12   <groupId>org.apache.activemq</groupId>
13   <artifactId>activemq-amqp</artifactId>
14   <version>${activemq.version}</version>
15 </dependency>
16 <dependency>
17   <groupId>org.apache.activemq</groupId>
18   <artifactId>activemq-blueprint</artifactId>
19   <version>${activemq.version}</version>
20 </dependency>
21 <dependency>
22   <groupId>org.apache.activemq</groupId>
23   <artifactId>activemq-broker</artifactId>
24   <version>${activemq.version}</version>
25 </dependency>
26 <dependency>
27   <groupId>org.apache.activemq</groupId>
```

从上面的 spring-boot-starter-dependencies 的 pom.xml 中我们可以发现，一部分坐标的版本、依赖管理、插件管理已经定义好，所以我们的 SpringBoot 工程继承 spring-boot-starter-parent 后已经具备版本锁定等配置了。所以启动器依赖的作用就是进行依赖的传递。

7. 分析 spring-boot-starter-web



```
46     <artifactId>spring-boot-starter</artifactId>
47     <version>2.3.3.RELEASE</version>
48     <scope>compile</scope>
49 </dependency>
50 <dependency>
51     <groupId>org.springframework.boot</groupId>
52     <artifactId>spring-boot-starter-json</artifactId>
53     <version>2.3.3.RELEASE</version>
54     <scope>compile</scope>
55 </dependency>
56 <dependency>
57     <groupId>org.springframework.boot</groupId>
58     <artifactId>spring-boot-starter-tomcat</artifactId>
59     <version>2.3.3.RELEASE</version>
60     <scope>compile</scope>
61 </dependency>
62 <dependency>
63     <groupId>org.springframework</groupId>
64     <artifactId>spring-web</artifactId>
65     <version>5.2.8.RELEASE</version>
66     <scope>compile</scope>
67 </dependency>
68 <dependency>
69     <groupId>org.springframework</groupId>
70     <artifactId>spring-webmvc</artifactId>
71     <version>5.2.8.RELEASE</version>
72     <scope>compile</scope>
73 </dependency>
74 </dependencies>
```

从上面的 spring-boot-starter-web 的 pom.xml 中我们可以发现，spring-boot-starter-web 就是将 web 开发要使用的 spring-web、spring-webmvc 等坐标进行了“打包”，这样我们的工程只要引入 spring-boot-starter-web 启动器依赖的坐标就可以进行 web 开发了，同样体现了依赖传递的作用。

8. 自动配置原理解析

按住 Ctrl 点击查看启动类上的注解@SpringBootApplication


```

SpringBootApplication.java x
48  * @author Andy Wilkinson
49  * @since 1.2.0
50  */
51  @Target(ElementType.TYPE)
52  @Retention(RetentionPolicy.RUNTIME)
53  @Documented
54  @Inherited
55  @SpringBootConfiguration
56  @EnableAutoConfiguration
57  @ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
58  public @interface SpringBootApplication {
59
60
61  /**
62   * Exclude specific auto-configuration classes such
63   * @return the classes to exclude
64   */
65  @AliasFor(annotation = EnableAutoConfiguration.class)
66  Class<?>[] exclude() default {};
67

```

启用这三个功能，即：

@EnableAutoConfiguration：启用 Spring Boot 的自动配置机制

@ComponentScan：启用@Component 对应用程序所在的软件包的扫描

@SpringBootConfiguration：等同与@Configuration，既标注该类是 Spring 的一个配置类,允许在上下文中注册额外的 bean 或导入其他配置类。

按住 Ctrl 点击查看注解@EnableAutoConfiguration

```

48  */
49  @Target(ElementType.TYPE)
50  @Retention(RetentionPolicy.RUNTIME)
51  @Documented
52  @Inherited
53  @AutoConfigurationPackage
54  @Import({AutoConfigurationImportSelector.class})
55  public @interface EnableAutoConfiguration {
56
57
58  String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enabled.override";
59
60  /**
61   * Exclude specific auto-configuration classes such
62   * @return the classes to exclude
63   */
64

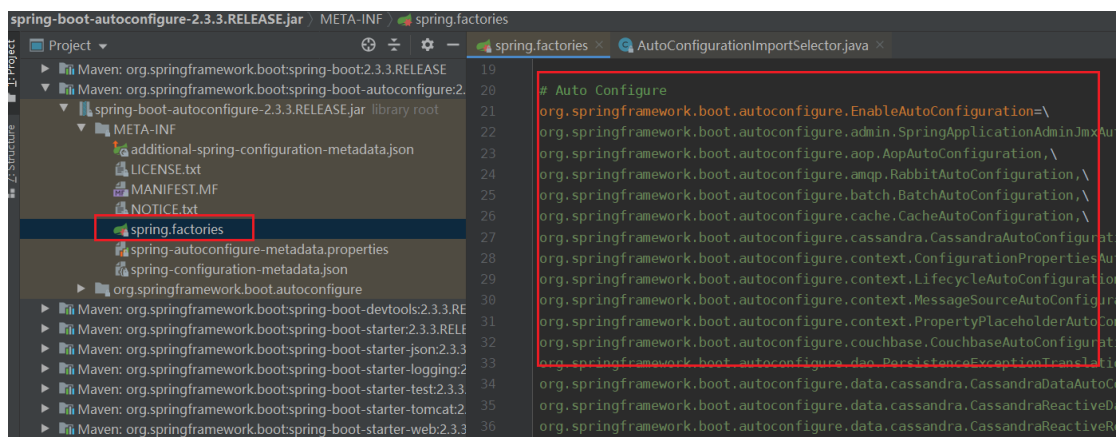
```

其中，@Import(AutoConfigurationImportSelector.class) 导入

AutoConfigurationImportSelector 类

```
AutoConfigurationImportSelector.java
/**
 * Return the auto-configuration class names that should be considered. By default
 * this method will load candidates using {@link SpringFactoriesLoader} with
 * {@link #getSpringFactoriesLoaderFactoryClass()}.
 * @param metadata the source metadata
 * @param attributes the {@link #getAttributes(AnnotationMetadata) annotation
 * attributes}
 * @return a list of candidate configurations
 */
protected List<String> getCandidateConfigurations(AnnotationMetadata metadata, AnnotationAttributes attributes) {
    List<String> configurations = SpringFactoriesLoader.loadFactoryNames(getSpringFactoriesLoaderFactoryClass(),
        getBeanClassLoader());
    Assert.notEmpty(configurations, message: "No auto configuration classes found in META-INF/spring.factories. If you
        + "are using a custom packaging, make sure that file is correct.");
    return configurations;
}
```

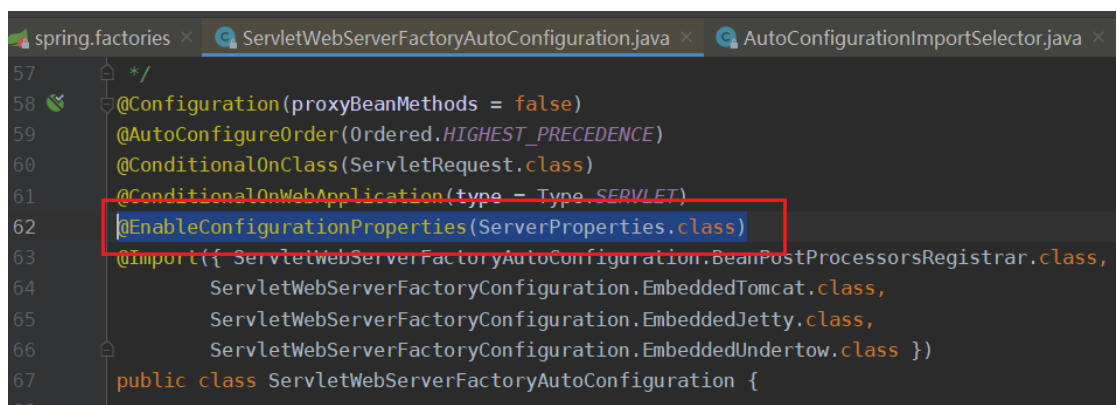
其中，SpringFactoriesLoader.loadFactoryNames 方法的作用就是从 META-INF 目录下 spring.factories 文件中读取指定类对应的类名称列表



```
spring-boot-autoconfigure-2.3.3.RELEASE.jar / META-INF / spring.factories
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,
org.springframework.boot.autoconfigure.context.LifecycleAutoConfiguration,
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,
org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration,
org.springframework.boot.autoconfigure.couchbase.CouchbaseAutoConfiguration,
org.springframework.boot.autoconfigure.dao.PersistenceExceptionTranslationAutoConfiguration,
org.springframework.boot.autoconfigure.data.cassandra.CassandraDataAutoConfiguration,
org.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveDataAutoConfiguration,
org.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveDataInitializationAutoConfiguration,
```

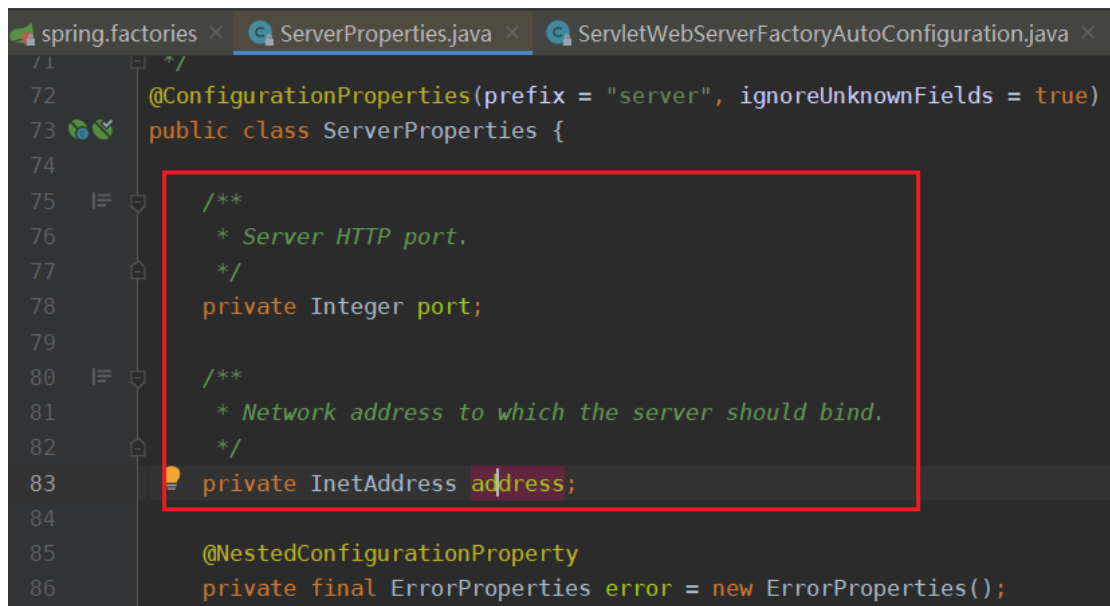
上面配置文件存在大量的以 Configuration 为结尾的类名称，这些类就是存有自动配置信息的类，而 SpringApplication 在获取这些类名后再加载。

我们以 ServletWebServerFactoryAutoConfiguration 为例来分析源码：



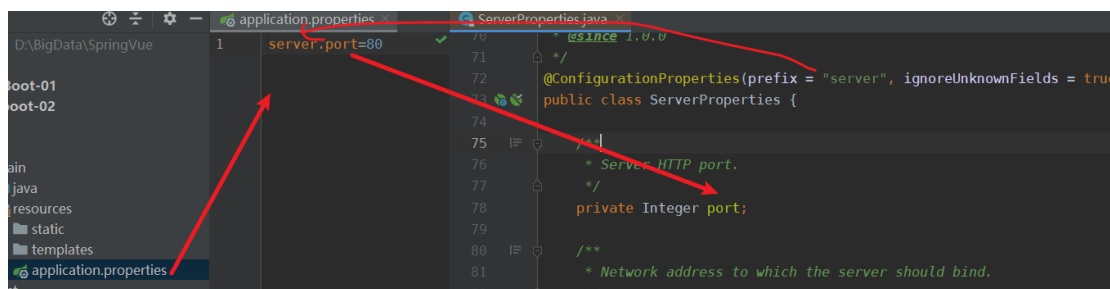
```
spring.factories x ServletWebServerFactoryAutoConfiguration.java x AutoConfigurationImportSelector.java x
57 */
58 @Configuration(proxyBeanMethods = false)
59 @AutoConfigureOrder(Ordered.HIGHEST_PRECEDENCE)
60 @ConditionalOnClass(ServletRequest.class)
61 @ConditionalOnWebApplication(type = Type.SERVLET)
62 @EnableConfigurationProperties(ServerProperties.class)
63 @Import({ ServletWebServerFactoryAutoConfiguration.BeanPostProcessorsRegistrar.class,
64     ServletWebServerFactoryConfiguration.EmbeddedTomcat.class,
65     ServletWebServerFactoryConfiguration.EmbeddedJetty.class,
66     ServletWebServerFactoryConfiguration.EmbeddedUndertow.class })
67 public class ServletWebServerFactoryAutoConfiguration {
68 }
```

进入 ServerProperties.class 源码如下



```
71  /*  
72  @ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)  
73  public class ServerProperties {  
74  
75      /**  
76       * Server HTTP port.  
77       */  
78      private Integer port;  
79  
80      /**  
81       * Network address to which the server should bind.  
82       */  
83      private InetAddress address;  
84  
85      @NestedConfigurationProperty  
86      private final ErrorProperties error = new ErrorProperties();
```

其中，prefix = "server" 表示 SpringBoot 配置文件中的前缀，SpringBoot 会将配置文件中以 server 开始的属性映射到该类的字段中。映射关系如下：



第六章 SpringBoot 的配置文件

1. SpringBoot 配置文件类型和作用

SpringBoot 是基于约定的，所以很多配置都有默认值，但如果想使用自己的配置替换默认配置的话，就可以使用 application.properties 或者 application.yml（application.yaml）进行配置。

SpringBoot 默认会从 Resources 目录下加载 application.properties 或 application.yml（application.yaml）文件

其中，application.properties 文件是键值对类型的文件，之前一直在使用，所以此处不在对 properties 文件的格式进行阐述。除了 properties 文件外，SpringBoot 还可以使用 yaml 文件进行配置，下面对 yaml 文件进行讲解。

2. application.yml 配置文件

YML 文件格式是 YAML (YAML Aint Markup Language)编写的文件格式，YAML 是一种直观的能够被电脑识别的数据序列化格式，并且容易被人类阅读，容易和脚本语言交互的，可以被支持 YAML 库的不同的编程语言程序导入，比如：C/C++，Ruby，Python，Java，Perl，C#，PHP 等。YML 文件是以数据为核心的，比传统的 xml 方式更加简洁。YML 文件的扩展名可以使用.yml 或者.yaml。

3. yml 配置文件的语法

● 配置普通数据

语法： `key: value`

示例代码：

```
Name: Tom
```

注意：value 之前有一个空格

● 配置对象数据

语法：

```
key:
  key1: value1
  key2: value2
```

或者：

```
key: {key1: value1,key2: value2}
```

示例代码：

```
person:
  name: haohao
  age: 31
  addr: beijing
```

或者

```
person: {name: haohao,age: 31,addr: beijing}
```

注意：key1 前面的空格个数不限定，在 yml 语法中，相同缩进代表同一个级别

● 配置 Map 数据

同上面的对象写法

● 配置数组 (List、Set) 数据

语法：

```
key:
  - value1
  - value2
```

或者：

```
key: [value1,value2]
```

示例代码：

city:

- beijing
- tianjin
- shanghai
- chongqing

或者

city: [beijing,tianjin,shanghai,chongqing]

集合中的元素是对象形式

student:

- name: zhangsan
age: 18
score: 100
- name: lisi
age: 28
score: 88
- name: wangwu
age: 38
score: 90

注意：value 与之间的 - 之间存在一个空格

4. SpringBoot 配置信息的查询

上面提及过，SpringBoot 的配置文件，主要的目的就是配置信息进行修改的，但在配置时的 key 从哪里去查询呢？我们可以查阅 SpringBoot 的官方文档

文档 URL: <https://docs.spring.io/spring-boot/docs/2.3.3.RELEASE/reference/htmlsingle/#common-application-properties>

我们可以通过配置 application.properties 或者 application.yml 来修改 SpringBoot 的默认配置

例如：

application.properties 文件

```
server.port=8888
server.servlet.context-path=demo
```

application.yml 文件

```
server:
  port: 8888
  servlet:
    context-path: /demo
```

5. 配置文件与配置类的属性映射方式

使用注解@Value 映射

我们可以通过@Value 注解将配置文件中的值映射到一个 Spring 管理的 Bean 的字段上
例如：

application.yml 配置如下：

```
person:
  name: zhangsan
  age: 18
```

示例代码

```
@RestController
public class MyController {

    @Value("${person.name}")
    private String name;
    @Value("${person.age}")
    private Integer age;

    @RequestMapping("/")
    String home(){
        return name+" "+age;
    }
}
```

使用注解@ConfigurationProperties 映射

通过注解@ConfigurationProperties(prefix="配置文件中的 key 的前缀")可以将配置文件中的配置自动与实体进行映射

此方式注意：需要添加依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>
```

使用@ConfigurationProperties 方式可以进行配置文件与实体字段的自动映射，但需要字段必须提供 set 方法才可以，而使用@Value 注解修饰的字段不需要提供 set 方法

```

@RestController
@ConfigurationProperties(prefix = "person")
public class MyController {

    private String name;
    private String age;

    public void setName(String name) {
        this.name = name;
    }

    public void setAge(String age) {
        this.age = age;
    }

    @RequestMapping("/")
    String home() { return name+" "+age; }

}

```

第七章 SpringBoot 与整合其他技术

1. SpringBoot 整合 Mybatis

添加 Mybatis 的启动器依赖

```

<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>

```

添加 Mysql 驱动

```

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>

```

添加数据库连接信息

#DB Configuration:

```
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test?characterEncoding=utf8&useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=123456
```

在 application.properties 中添加 mybatis 的信息

```
#spring 集成 Mybatis 环境
mybatis.type-aliases-package=com.example.springboot02.domain
#加载 Mybatis 映射文件
mybatis.mapper-locations=classpath:mapper/*Mapper.xml
```

编写 Mapper

```
@Mapper
public interface UserMapper {

    int deleteByPrimaryKey(Integer id);

    int insert(User record);

    User selectByPrimaryKey(Integer id);

    List<User> selectAll();

    int updateByPrimaryKey(User record);

}
```

配置 Mapper 映射文件

此处我们使用插件自动生成 Mapper 和配置文件

众所周知，MyBatis 的核心有两大组件：SqlSessionFactory 和 Mapper 接口。前者表示数据库链接，后者表示 SQL 映射。当我们基于 Spring 使用 MyBatis 的时候，也要保证在

Spring 环境中能存在着两大组件。

MyBatis-Spring-Boot-Starter 将会完成以下功能:

- 1、自动发现存在的 DataSource
- 2、利用 SqlSessionFactoryBean 创建并注册 SqlSessionFactory
- 3、创建并注册 SqlSessionTemplate
- 4、自动扫描 Mappers，并注册到 Spring 上下文环境方便程序的注入使用

默认情况下，MyBatis-Spring-Boot-Starter 会查找以 **@Mapper** 注解标记的映射器。

你需要给每个 MyBatis 映射器标识上 `@Mapper` 注解，但是这样非常的麻烦，这时可以使用 `@MapperScan` 注解来扫描包。

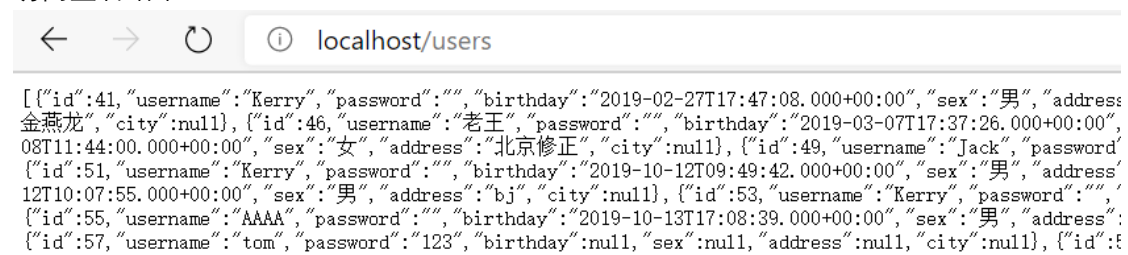
```
@SpringBootApplication
@ComponentScan("com.example.springboot02.dao")
public class Springboot02Application {

    public static void main(String[] args) {
        SpringApplication.run(Springboot02Application.class, args);
    }
}
```

编写 Controller 测试

```
@RestController
public class UserController {
    @Autowired
    UserMapper userMapper;
    @RequestMapping("/users")
    List<User> getUsers(){
        List<User> users = userMapper.selectAll();
        return users;
    }
}
```

访问查看结果:

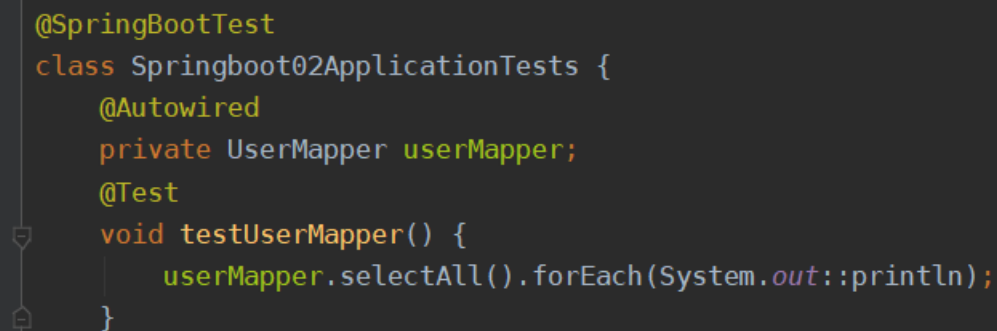


2. SpringBoot 整合 Junit

添加 Junit 的启动器依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

编写测试类:

A screenshot of a code editor with a dark background. It shows a Java class named 'Springboot02ApplicationTests' annotated with '@SpringBootTest'. The class has a private field 'userMapper' annotated with '@Autowired' and a test method 'testUserMapper()' annotated with '@Test'. The test method calls 'userMapper.selectAll().forEach(System.out::println);'.

```
@SpringBootTest
class Springboot02ApplicationTests {
    @Autowired
    private UserMapper userMapper;
    @Test
    void testUserMapper() {
        userMapper.selectAll().forEach(System.out::println);
    }
}
```

3. SpringBoot 整合 Redis

添加 redis 的启动器依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

application.properties 配置 redis 的连接信息

```
#Redis
spring.redis.host=127.0.0.1
spring.redis.port=6379
```

编写测试类

```

@SpringBootTest
class Springboot02ApplicationTests {
    @Autowired
    private UserMapper userMapper;
    @Test
    void testUserMapper() { userMapper.selectAll().forEach(System.out::println); }

    @Autowired
    private RedisTemplate<String, String> redisTemplate;

    @Test
    public void testRedis() throws JsonProcessingException {
        //从redis缓存中获得指定的数据
        String userListData = redisTemplate.boundValueOps( key: "user.findAll").get();
        //如果redis中没有数据的话
        if(null==userListData){
            //查询数据库获得数据
            List<User> all = userMapper.selectAll();
            //转换成json格式字符串
            ObjectMapper om = new ObjectMapper();
            userListData = om.writeValueAsString(all);
            //将数据存储到redis中，下次在查询直接从redis中获得数据，不用在查询数据库
            redisTemplate.boundValueOps( key: "user.findAll").set(userListData);
            System.out.println("=====从数据库获得数据=====");
        }else{
            System.out.println("=====从redis缓存中获得数据=====");
        }
        System.out.println(userListData);
    }
}

```

4. Spring 整合 Swagger

添加依赖

```

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
    <version>3.0.0</version>
</dependency>

```

启动程序上添加注解

@EnableOpenApi

```

@SpringBootApplication
@ComponentScan("com.example")
@MapperScan("com.example.springboot02.dao")
@EnableOpenApi
public class Springboot02Application {
    public static void main(String[] args) { SpringApplication.run(Springboot02App
}

```

访问地址: <http://localhost:8080/swagger-ui/index.html>

第八章 项目简介服务端接口开发

实现优学在线网站的注册登录,首页,列表页,播放页

1. 需求一 注册页面

手机:

创建密码:

确认密码:

验证码:

88367

短信验证:

[免费获取短信](#)

注册

已有账号, 请[登录](#)

用户专享服务:

极致的视频学习体验

丰富的直播课程免费学

开课提醒, 不错过每一次精彩

讲、学、练、考, 多种高效学习方式快速提升成绩

接口分析

- 1) 验证码获取接口
- 2) 验证码验证接口
- 3) 用户注册接口

手机:

✓

创建密码:

✓

确认密码:

✓

验证码:

88367

短信验证:

33秒后重新获得

请输入验证码!

注册

已有账号, 请[登录](#)

注册成功后自动登录

2. 需求二 登录页面

已有账号登录页面



接口分析

1) 用户手机密码登录接口

3. 需求三 首页



接口分析:

- 1) Banner 轮播广告接口
- 2) 课程分类接口

4. 需求四 列表页



接口分析

1) 列表查询接口

5. 需求五 详情页



优学IT职业在线教育

首页

精品课程

在线直播

IT培训班

IT培训机构

在线就业班

APP下载

首页 > 全部课程 > 程序开发 > 大数据开发

【进阶】走进大数据之Hive企业级实战 数据管理与统计...

课程安排 有效期 至2030-12-28 | 6课时

大数据开发

Hive企业级实战 数据管理与统计分析

分享 收藏 上课提醒 退订课程

课程介绍	课程安排 (6)	课程评价	常见问题	联系客服
全部	视频(6)			
	第1讲 · 大数据概述及Hive基本概念1	45:42		
	第2讲 · 大数据概述及Hive基本概念2	41:55		

接口分析

- 1) 课程详情接口
- 2) 课程课时安排接口
- 3) 购买接口

6. 需求六 播放页

优学IT职业在线教育

首页

精品课程

在线直播

IT培训班

IT培训机构

在线就业班

APP下载

课件1: 大数据概述及Hive基本概念1

返回课程主页

优就业www.ujiuye.com

00:00:09 00:45:41

线路 收藏 手机看 问题反馈 白天模式 倍速

目录

01 【视频】大数据概述及Hive基本概念1 [45:42]

02 【视频】大数据概述及Hive基本概念2 [41:55]

03 【视频】Hive的基本数据类型 [59:54]

04 【视频】Hive之DDL数据操作 [01:08:03]

05 【视频】DQL数据操作1 [43:01]

06 【视频】DQL数据操作2 [34:24]

第九章 Vue 框架

1. Vue.js 是什么

Vue (读音 /vju:/, 类似于 view) 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是, Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层, 不仅易于上手, 还便于与第三方库或既有项目整合。另一方面, 当与现代化的工具链以及各种支持类库结合使用时, Vue 也完全能够为复杂的单页应用提供驱动。

快速入门

通过如下方式引入 Vue 或者下载放到本地引入:

```
<!-- 开发环境版本, 包含了有帮助的命令行警告 -->
```

```
<script
```

```
src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>或者:
```

```
<!-- 生产环境版本, 优化了尺寸和速度 -->
```

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>My first Vue app</title>
```

```
  <script src="../vue.js"></script>
```

```
</head>
```

```
<body>
```

```
<div id="app">
```

```
  {{ message }}
```

```
</div>
```

```
<script>
```

```
  var app = new Vue({
```

```
    el: '#app',
```

```
    data: {
```

```
      message: 'Hello Vue!'
```

```
    }
```

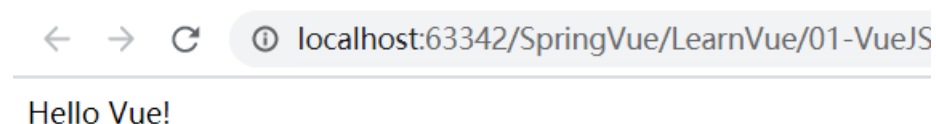
```
  })
```

```
</script>
```

```
</body>
```

```
</html>
```


浏览器访问该页面:



2. Vue 实例

每个 Vue 应用都是通过用 Vue 函数创建一个新的 Vue 实例开始的:

```
var vm = new Vue({  
  // 选项  
})
```

一个 Vue 应用由一个通过 new Vue 创建的根 Vue 实例，以及可选的嵌套的、可复用的组件树组成。

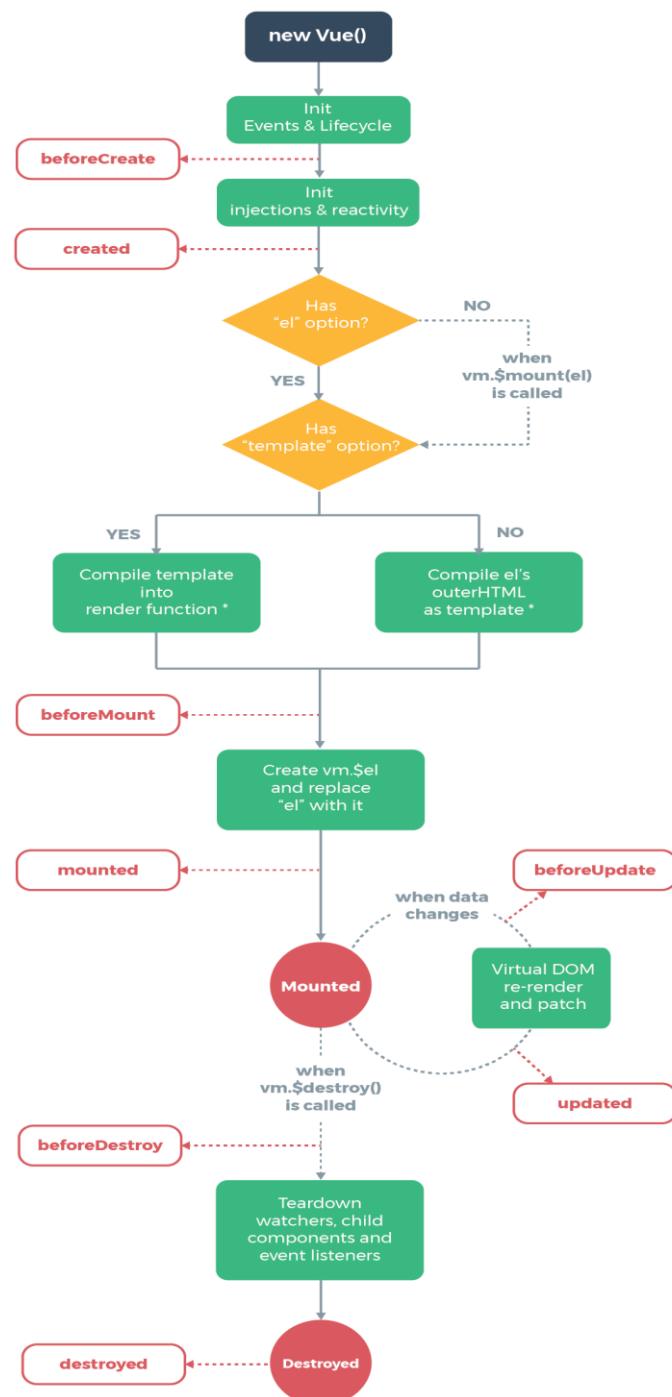
```
<body>  
<div id="app">  
  <div @click='handleClick'>{{message}}</div>  
  <my-item/>  
</div>  
<script>  
  //Vue 组件都是 Vue 实例  
  Vue.component("my-item",{  
    template:"<div>hello component </div>"  
  });  
  
  //创建vue实例 根实例  
  const vm = new Vue({  
    el: '#app', //接管app dom  
    data: { //数据 值发生改变时，视图将会产生“响应”，即匹配更新为新的值  
      message: 'Hello Vue!'  
    },  
    methods:{  
      handleClick:function () {  
        console.log("hello world");  
        this.message = 'hello world!';  
      }  
    }  
  });  
  
  // Vue 实例还暴露了一些有用的实例 property 与方法。它们都有前缀 $，以便与用户定义的 property 区分开来。  
  vm.$watch('message',function () {  
    console.log("message 值改变");  
  })  
  
  console.log(vm.$data);  
  console.log(vm.$el);  
</script>  
</body>
```

实例树形结构

▼ <Root>
 <MyItem>

3. 生命周期

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

4. 模板语法-插值

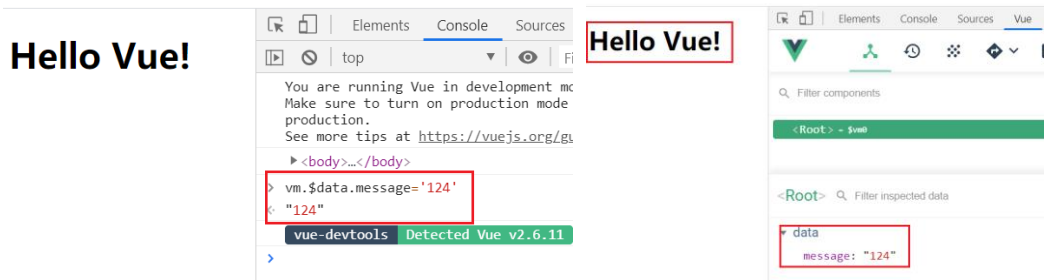
文本:

数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值:

```
{{ message }}
```

通过使用 `v-once` 指令, 你也能执行一次性地插值, 当数据改变时, 插值处的内容不会更新。

```
<h1 v-once> {{ message }}</h1>
```



如图,修改数据对象 `message` 属性的值,视图中绑定的值不会改变.

原始 HTML:

双大括号会将数据解释为普通文本, 而非 HTML 代码。为了输出真正的 HTML, 你需要使用 `v-html`



5. 模板语法-指令

指令 (Directives) 是带有 `v-` 前缀的特殊 attribute。指令的职责是, 当表达式的值改变时, 将其产生的连带影响, 响应式地作用于 DOM。

指令	用途
<code>v-pre</code>	跳过元素编译
<code>v-once</code>	只渲染一次
<code>v-text</code>	更新元素内容
<code>v-html</code>	渲染 HTML 源码
<code>v-show</code>	切换 display 属性
<code>v-if</code> <code>v-else-if</code> <code>v-else</code>	条件渲染
<code>v-for</code>	列表渲染

v-on	绑定事件监听
v-bind	动态绑定属性
v-model	表单控件双向绑定
v-cloak	斗篷,实例编译结束后移除
v-slot	提供具名插槽或需要接收 prop 的插槽

模板语法-指令 示例:

```

<!DOCTYPE html>
<html>
<head>
  <title>模板语法 指令</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
  <style>
    [v-cloak]{
      color: red;
    }
  </style>
</head>
<body>
<div id="app">
  <div>Hello World! {{message}}</div>
<!-- 跳过这个元素和它的子元素的编译过程-->
  <div v-pre>{{message}}</div>
<!-- 只渲染一次-->
  <div v-once>Hello World! {{message}}</div>
<!-- 解析 HTML-->
  <div v-html="rawHtml"></div>
<!-- 替换 Hello World!-->
  <div v-text="message">Hello World!</div>
<!-- 显示隐藏-->
  <div v-show="display">Show Me</div>
<!-- 条件渲染-->
  <div v-if="type === 'A'">A </div>
  <div v-else-if="type === 'B'">B</div>
  <div v-else-if="type === 'C'">C</div>
  <div v-else>Not A/B/C</div>
<!-- 列表渲染 基于源数据多次渲染元素或模板块
      数据类型 Array | Object | number | string | Iterable
-->
  <ul>
    <li v-for="(user,index) in users" v-bind:key="user.id" >
      {{user.id}}-{{user.name}}-{{index}}
    </li>
  </ul>

```

```

    </ul>
<!-- 绑定事件监听器-->
    <button v-on:click="btnClick">显示/隐藏</button>
<!-- 动态地绑定属性 -->
    <div>
        
    </div>
<!-- 表单双向绑定 限制于 input textarea select components-->
    <div>
        <input v-model="message" placeholder="编辑内容">
    </div>
    <div>
        <textarea v-model="message" placeholder="编辑内容
"></textarea>
    </div>
    <div>
        <select v-model="message">
            <option>苹果</option>
            <option>香蕉</option>
            <option>葡萄</option>
        </select>
    </div>

<!-- v-cloak 斗篷 实例编译结束时移除-->
    <div v-cloak >实例编译结束改变颜色</div>

<!-- v-slot 插槽 -->
    <component-a>
        <template v-slot:right >
            <input v-model="message" >
        </template>
        <template v-slot:left>
            <button @click="btnClick" >确定</button>
        </template>
    </component-a>
</div>
<script>
    // setTimeout(()=>{
        let componentA = {
            data:{
                name:function () {
                    return "组件 A"
                }
            }
        }
    })

```

```

    },
    template: '<div><slot name="left"></slot><slot
name="right"></slot></div>'
  }

  const vm = new Vue({
    el: '#app',
    components: {
      componentA: componentA
    },
    data: {
      message: 'Hello Vue!',
      rawHtml: '<div>Hi Vue</div>',
      type: 'A',
      display: true,
      users:
        [
          {name: '小白', id: '001'},
          {name: '小黑', id: '002'}
        ]
    },
    img: {
      url: 'https://cn.bing.com/th?id=OHR.UrquhartCastle_EN-
CN3630399683_1920x1080.jpg&rf=LaDigue_1920x1080.jpg&pid=hp',
      height: "100px",
      width: "200px"
    }
  },
  methods: {
    btnClick: function () {
      this.display=!this.display;

      this.type="ABC".charAt(parseInt(Math.random()*3,10));
    }
  },
})

// },3000)
</script>
</body>
</html>

```

6. 计算属性和侦听

计算属性:对于任何复杂逻辑, 你都应当使用计算属性,计算属性写在 `computed` 对象内.语法:

```
computed: { //计算属性是基于它们的响应式依赖进行缓存的
  fullName: function() {
    return this.firstName + this.lastName
  }
}
```

如上 `fullName` 计算属性的值由 `firstName` 和 `lastName` 的值决定.

侦听属性:观察和响应 Vue 实例上的数据变动的方式 `watch`

```
watch: { //数据变化时执行异步或开销较大的操作时.
  //要侦听的属性名称: 函数体
}
```

计算属性和侦听示例

```
<!DOCTYPE html>
<html>
<head>
  <title>计算属性和侦听</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
</head>
<body>
<div id="app">
  <div>姓名: {{fullName}}</div>
  <div>姓名: {{getFullName()}}</div>
  <div>姓名: {{watchFullName}}</div>
  <div>时间戳: {{nowTime}}</div>
  <div>时间戳: {{getNowTime()}}</div>
  <div>{{counter}}</div>
  <button @click="btnClick">计数</button>
</div>

<script>
  const vm = new Vue({
    el: '#app',
    data: {
      counter: 0,
      firstName: '王',
      lastName: '小宝',
      watchFullName: '王小宝'
    },
    computed: { //计算属性是基于它们的响应式依赖进行缓存的
```

```

        fullName:function() {
            return this.firstName + this.lastName
        },
        nowTime:function () {
            return Date.now()
        }
    },
    methods:{//每当触发重新渲染时, 调用方法将总会再次执行函数
        getFullName:function () {

            return this.firstName+this.lastName
        },
        getNowTime(){
            return Date.now()
        },
        btnClick:function () {
            this.counter++
        }
    },
    watch:{ //数据变化时执行异步或开销较大的操作时.如下是对 watch 的滥用.

        firstName: function () {
            this.watchFullName = this.firstName+this.lastName
        },
        lastName: function () {
            this.watchFullName = this.firstName+this.lastName
        },

    }
})
</script>
</body>
</html>

```

7. Class 与 Style 动态绑定

操作元素的 class 列表和内联样式是数据绑定的一个常见需求。因为它们都是 attribute，所以我们可以用 `v-bind` 处理它们：只需要通过表达式计算出字符串结果即可。不过，字符串拼接麻烦且易错。因此，在将 `v-bind` 用于 class 和 style 时，Vue.js 做了专门的增强。表达式结果的类型除了字符串之外，还可以是对象或数组。

Class 和 Style 动态绑定示例

```
<!DOCTYPE html>
<html>
<head>
  <title>Class 和 Style 动态绑定</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
  <style>
    .active {
      color: red;
    }
    .largeFont{
      font-size: x-large;
    }
  </style>
</head>
<body>
<div id="app">
  <!-- 通过 isActive 属性的值动态绑定 active 样式-->
  <div v-bind:class="{active:isActive}">Hello Vue!</div>
  <!-- 可以写成下面形式-->
  <div v-bind:class="classObject">Hello Vue!</div>
  <!-- 可以使用计算属性-->
  <div v-bind:class="classAct">Hello Vue!</div>
  <!-- 可以使用数组绑定多个样式-->
  <div v-bind:class="[classAct , bigFont]">Hello Vue!</div>
  <!-- 绑定内联样式-->
  <div v-bind:style="{ color: activeColor, fontSize: fontSize +
'px' }">Hello Vue!</div>

</div>
<script>
  const vm = new Vue({
    el: '#app',
    data: {
      activeColor:'red',
      fontSize:30,
      bigFont:'largeFont',
      isActive:true,
      classObject:
        {
          active:true
        }
    },
```

```

        computed:{
            classAct:function () {
                return{
                    active:this.isActive
                }
            }
        }
    })
</script>
</body>
</html>

```

8. 事件绑定

可以用 `v-on` 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。

事件绑定示例

```

<!DOCTYPE html>
<html>
<head>
    <title>事件绑定</title>
    <meta charset="UTF-8">
    <script src="../js/vue.js"></script>

</head>
<body>
<div id="app">

<!--      绑定一个点击事件 v-on 可以简写@ -->
    <button v-on:click="btnClick">事件监听</button>
    <br/>
<!--      传入参数-->
    <button @click="sayHello('hello vue!')">SayHello</button>
    <br/>
    <pre>
        事件修饰符
        .stop 阻止单击事件冒泡
        .prevent 阻止默认行为
        .capture 事件捕获
        .self 事件源是自己时触发
        .once 事件触发一次
        .passive 阻止默认行为
    </pre>

```

```

<!--      prevent    阻止默认行为-->
    <form action="http://baidu.com">
        <button type="submit" @click.prevent="onSubmit">阻止默认
提交</button>
    </form>
    <br/>

<!--      stop      阻止单击事件冒泡-->
    <div v-on:click="sayHello('我是DIV')" >stop 示例 我是DIV
        <button v-on:click.stop="sayHello('我是
Button')">SayHello</button>
    </div>

<!--      capture   事件捕获-->
    <div v-on:click.capture="sayHello('我是DIV')" >capture 示例
我是DIV
        <button v-on:click="sayHello('我是
Button')">SayHello</button>
    </div>

<!--      self      事件源是自己时触发-->
    <div v-on:click.self="sayHello('我是DIV')" >self 示例 我是
DIV
        <button v-on:click="sayHello('我是
Button')">SayHello</button>
    </div>

<!--      once      事件触发一次-->
    <button @click.once="sayHello('仅仅触发一次')">触发一次
</button>

<!--      passive   指示监听器永远不会调用 preventDefault()
    passive 就是专门用来跟 preventDefault 作对的，使他不起作用，同时，
passive 不能和 prevent 同时使用，prevent 会失效，而且会警告！！
    passive 主要用在移动端的 scroll 事件，来提高浏览器响应速度，提升用户
体验。

    因为 passive=true 等于提前告诉了浏览器，touchstart 和 touchmove 不
会阻止默认事件，
    手刚开始触摸，浏览器就可以立刻给与响应；否则，手触摸屏幕了，但要等待
touchstart 和 touchmove 的结果，
    多了这一步，响应时间就长了，用户体验也就差了。
-->

```

```

<pre>    按键修饰符
        监听键盘事件时检查详细的按键
        语法：v-on:keyup.按键码或者别名

```

默认别名:

- .enter
- .tab
- .delete (捕获“删除”和“退格”键)
- .esc
- .space
- .up
- .down
- .left
- .right
- .middle
- .ctrl
- .alt
- .shift

.meta 注意: 在 Mac 系统键盘上, meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)

```
</pre>
```

```
<br/>
```

```
<input v-on:keyup.enter="changeMessage" placeholder="输入内容回车" />{{message}}
```

```
</div>
```

```
<script>
```

```
const vm = new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello !'  
  },  
  methods: {
```

```
    btnClick: function (event) {  
      alert(this.message);  
      console.log(event);  
    },
```

```
    sayHello: function (msg) {  
      alert(msg)  
    }  
  },
```

```
  onSubmit: function (event) {  
    //阻止默认提交行为 可以使用 修饰符 prevent 代替下面语句  
    // event.preventDefault()  
    alert("表单提交")  
  },
```

```
  changeMessage: function (event) {  
    this.message=event.target.value;
```

```

    }
  }
})
</script>
</body>
</html>

```

9. 表单输入绑定

v-model 指令在表单 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。

表单控件绑定示例

```

<!DOCTYPE html>
<html>
<head>
  <title>表单控件绑定</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
</head>
<body>
<div id="app">
  <div>
    <label for="txtLabel">文本</label>
    <input id="txtLabel" v-model="message">{{message}}
  </div>
  <div>
    <label for="areaLabel">文本域</label>
    <textarea id="areaLabel" v-model="message"></textarea>
  </div>
  <div>
    <label for="singleChk">单复选框:</label>
    <input id="singleChk" type="checkbox" v-model="checked">{{checked}}
    <div>多复选框:
      <input id="zhangshan" value="张三" type="checkbox" v-model="checkedNames"> <label for="zhangshan">张三</label>
      <input id="lisi" value="李四" type="checkbox" v-model="checkedNames"><label for="lisi">李四</label>
      <input id="wangwu" value="王五" type="checkbox" v-model="checkedNames"><label for="wangwu">王五</label>
      您的朋友们:{{checkedNames}}
    </div>
  </div>

```

```

</div>
<div>
  单选按钮:
  <input id="fish" type="radio" value="鱼" v-
model="picked"><label for="fish">鱼</label>
  <input id="bearpaw" type="radio" value="熊掌" v-
model="picked"><label for="bearpaw">熊掌</label>
  您的选择:{{picked}}
</div>
<div>
  单选择框
  <select v-model="selectedCity">
    <option disabled value="">请选择</option>
    <option v-for="city in cities" v-
bind:value="city">{{city}}</option>
  </select>
  您最喜欢的城市:{{selectedCity}}
</div>
<div>
  多选择框<br/>
  <select v-model="selectedCities" multiple>
    <option disabled value="">请选择</option>
    <option v-for="city in cities" v-
bind:value="city">{{city}}</option>
  </select>
  您喜欢的城市:{{selectedCities}}
</div>

<div>
  修饰符
  <br/>
  .lazy change 事件_之后_进行同步
  <input v-model.lazy="message">{{message}}
  <br/>
  .number 自动将用户的输入值转为数值类型
  <input v-model.number="age" type="number"> {{age}}
  <br/>
  .trim 过滤用户输入的首尾空白字符
  <input v-model.trim="message">{{message}}
</div>
</div>
<script>
  const vm = new Vue({

```

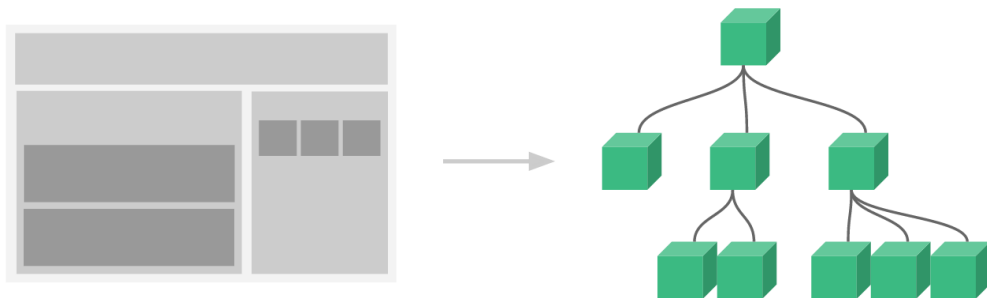
```

    el: '#app',
    data: {
      message: 'Hello !',
      checked: true,
      checkedNames: [],
      picked: '',
      cities: ['北京', '上海', '广州'],
      selectedCity: '',
      selectedCities: [],
      age: 0
    }
  })
</script>
</body>
</html>

```

10. 组件基础

组件是可复用的 Vue 实例，组件在 Vue 根实例中作为自定义元素来使用。通常一个应用会以一棵嵌套的组件树的形式来组织：



组件有两种注册类型：全局注册和局部注册。

组件示例

```

<!DOCTYPE html>
<html>
<head>
  <title>组件</title>
  <meta charset="UTF-8">
  <script src="../../js/vue.js"></script>
</head>
<body>

```

```

<div id="app1">
  <my-counter></my-counter>
  <counter-a></counter-a>
</div>
<div id="app2">
  <my-counter></my-counter>
<!-- <counter-a></counter-a>-->
  <counter-b></counter-b>
</div>
<script>
  const MyCounter={
    data:function () {
      return {
        counter:0
      }
    },
    template:'<div> 计数器: {{counter}} <button
@click="addOpt">+</button><button @click="subOpt">-</button>
</div>',
    methods:{
      addOpt:function () {
        this.counter++
      },
      subOpt:function () {
        this.counter--
      }
    }
  }

  //全局注册 所有 Vue 实例都可以用
  Vue.component("MyCounter",MyCounter )

  const vm = new Vue({
    el: '#app1',
    data: {
      message: 'Hello Vue!'
    },
    //局部注册 只有本实例可用
    components: {
      'counter-a':MyCounter
    }
  })

```



```

    const vm2 = new Vue({
      el:"#app2",
      components: {
        'counter-b':MyCounter
      }
    })
  </script>
</body>
</html>

```

11. 组件传值

通过 Prop 向子组件传递数据, Prop 是你可以在组件上注册的一些自定义 attribute。当一个值传递给一个 prop attribute 的时候, 它就变成了那个组件实例的一个 property。

组件传值示例

```

<!DOCTYPE html>
<html>
<head>
  <title>组件传值</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
</head>
<body>
<div id="app1">
  <!-- 通过属性向组件传值 -->
    <counter-a :title="title"></counter-a>
</div>

<script>
  const MyCounter={
    data:function () {
      return {
        counter:0
      }
    },
    //定义组件属性
    props:[
      'title'
    ],
    template:`
      <div> {{title}}: {{counter}}

```

```

        <button @click="add0pt">+</button>
        <button @click="sub0pt">-</button>
    </div>
    `
  },
  methods:{
    add0pt:function () {
      this.counter++
    },
    sub0pt:function () {
      this.counter--
    }
  }
}

const vm = new Vue({
  el: '#app1',
  data: {
    title: '我的计算器'
  },
  //局部注册 只有本实例可用
  components: {
    'counter-a':MyCounter
  }
})
</script>
</body>
</html>

```

12. 监听子组件事件

子组件可以通过调用内建的 \$emit 方法并传入事件名称来触发一个事件.

组件事件监听

```

<!DOCTYPE html>
<html>
<head>
  <title>组件事件监听</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
</head>
<body>
<div id="app1">

```

```

<!-- 监听子组件的 change-num 事件-->
    <counter-a :title="title" v-on:change-
num="changeNum" ></counter-a>
    <counter-a :title="title" v-on:change-
num="changeNum" ></counter-a>
    结果 : {{totalNum}}
</div>

<script>
    const MyCounter={
        data:function () {
            return {
                counter:0
            }
        },
        //定义组件属性
        props:[
            'title'
        ],
        //子组件通过 $emit 触发事件
        template:`
            <div> {{title}}: {{counter}}
                <button @click="addOpt">+</button>
                <button @click="subOpt">-</button>

            </div>
        `
    },
    methods:{
        addOpt:function () {
            this.counter++
            this.$emit('change-num',1)
        },
        subOpt:function () {
            this.counter--
            this.$emit('change-num',-1)
        }
    }
}

const vm = new Vue({
    el: '#app1',
    data: {
        totalNum:0,
        title: '我的计算器'
    }
})

```

```

    },
    //局部注册 只有本实例可用
    components: {
        'counter-a':MyCounter
    },
    methods: {
        changeNum:function (value) {
            this.totalNum+=value
        }
    }
  })
</script>
</body>
</html>

```

13. 非父子组件传值

通过事件总线,或者使用 Vuex 管理状态.

非父子组件传值总线示例

```

<!DOCTYPE html>
<html>
<head>
  <title>非父子组件传值</title>
  <meta charset="UTF-8">
  <script src="../js/vue.js"></script>
</head>
<body>
<div id="app1">
  <cpn-a content="Hello Vue"></cpn-a>
  <cpn-a content="Hello World"></cpn-a>
</div>

<script>
  //绑定一个总线属性
  Vue.prototype.bus= new Vue()

  const MyComponet={
    data:function () {
      return {
        selfContent:this.content
      }
    },

```

```

//定义组件属性
props:[
  'content'
],
template:`
  <div>
    <div @click="handleClick" > {{selfContent}} </div>

  </div>
`,
methods:{
  handleClick:function () { // 总线上触发 change 事件
    this.bus.$emit('change',this.selfContent)
  }
},
mounted:function () { // 在挂载回调中 总线上监听 change 事件
  let this_ = this
  this.bus.$on('change',function (msg) {
    this_.selfContent = msg
  })
}
}
const vm = new Vue({
  el: '#app1',
  components: {
    'cpn-a':MyComponet
  }
})
</script>
</body>
</html>

```

14. 动态组件

使用 is attribute 来切换不同的组件

```
<component v-bind:is="currentTabComponent"></component>
```

动态组件示例

```

<!DOCTYPE html>
<html>
<head>
  <title>动态组件</title>
  <meta charset="UTF-8">
  <script src="../../js/vue.js"></script>
</head>
<body>
<div id="app1">
  <comment :is="type" :content="getContent"></comment>
  <button @click="handleClick">toggle</button>
</div>

<script>

  const myComponent={
    data:function () {
      return {
        selfContent:this.content
      }
    },
    //定义组件属性
    props:[
      'content'
    ],
    template:`
      <div>
        <div > {{selfContent}} </div>
      </div>
    `
  }

  const vm = new Vue({
    el: '#app1',
    data:{
      type:'cpn-a'
    },
    computed:{
      getContent(){
        if(this.type==='cpn-a')
          return "我是 A 组件"
        else
          return '我是 B 组件'
        }
    }
  }),

```

```
    components: {
      'cpn-a': myComponent,
      'cpn-b': myComponent
    },
    methods: {
      handleClick() {
        this.type = this.type === 'cpn-a' ? 'cpn-b' : 'cpn-a'
      }
    }
  })
</script>
</body>
</html>
```

15. 过度和动画

详见官方文档 <https://cn.vuejs.org/v2/guide/transitions.html>

第十章 VUE CLI 脚手架

使用 Vue.js 开发大型应用时，我们需要考虑代码目录结构、项目结构和部署、热加载、代码单元测试等事情。如果每个项目都要手动完成这些工作，那无疑效率比较低效，所以通常我们会使用一些脚手架工具来帮助完成这些事情。

Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统，提供：

- 通过 `@vue/cli` 实现的交互式的项目脚手架。
- 通过 `@vue/cli` + `@vue/cli-service-global` 实现的零配置原型开发。
- 一个运行时依赖 (`@vue/cli-service`)，该依赖：
 - 可升级；
 - 基于 webpack 构建，并带有合理的默认配置；
 - 可以通过项目内的配置文件进行配置；
 - 可以通过插件进行扩展。
- 一个丰富的官方插件集合，集成了前端生态中最好的工具。
- 一套完全图形化的创建和管理 Vue.js 项目的用户界面。

Vue CLI 致力于将 Vue 生态中的工具基础标准化。它确保了各种构建工具能够基于智能的默认配置即可平稳衔接，这样你可以专注在撰写应用上，而不必花好几天去纠结配置的问题。

官方网站

<https://cli.vuejs.org/zh/>

使用前提

安装 NodeJS:

■ 安装 NodeJS

可以直接在官方网站中下载安装.

网址: <http://nodejs.cn/download/>

■ 检测安装的版本

默认情况下自动安装 Node 和 NPM

Node 环境要求 8.9 以上或者更高版本

■ 什么是 NPM 呢?

NPM 的全称是 Node Package Manager

是一个 NodeJS 包管理和分发工具, 已经成为了非官方的发布 Node 模块 (包) 的标准。

后续我们会经常使用 NPM 来安装一些开发过程中依赖包.

注意:

由于国内直接使用 npm 的官方镜像是非常慢的, 这里推荐使用淘宝 NPM 镜像。

你可以使用淘宝定制的 cnpm (gzip 压缩支持) 命令行工具代替默认的 npm:

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

这样就可以使用 cnpm 命令来安装模块了:

```
cnpm install [name]
```

或者

```
npm config set registry https://registry.npm.taobao.org
```

安装 Webpack

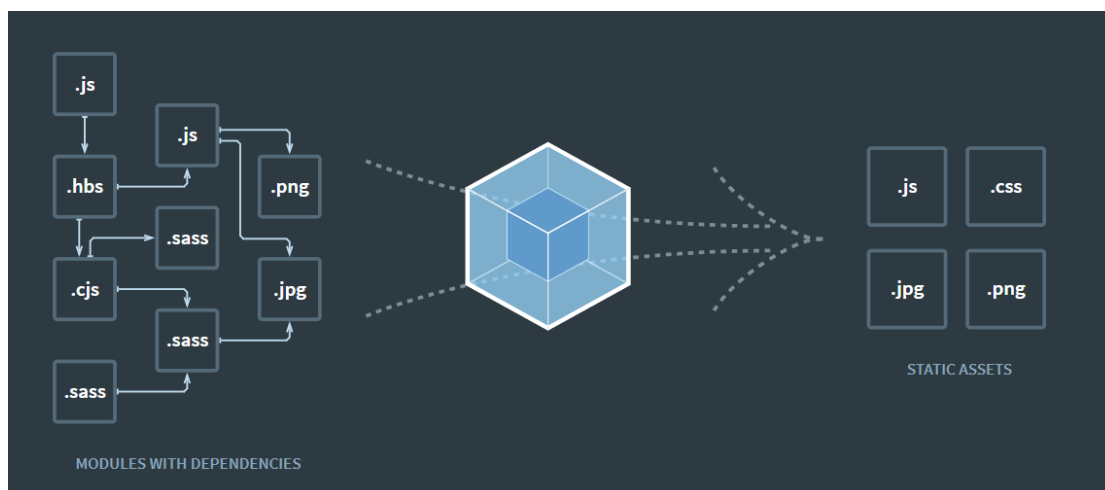
■ Vue.js 官方脚手架工具使用了 webpack 模板

对所有的资源会压缩等优化操作

它在开发过程中提供了一套完整的功能, 能够使得我们开发过程中变得高效。

■ Webpack 的全局安装

```
npm install webpack -g
```



安装 Vue 脚手架

```
npm install -g @vue/cli
```

注意：上面安装的是 Vue CLI3 的版本，如果需要按照 Vue CLI2 的方式初始化项目时不可以的。

拉取 2.x 模板 (旧版本)

Vue CLI 3 和旧版使用了相同的 `vue` 命令，所以 Vue CLI 2 (`vue-cli`) 被覆盖了。如果你仍然需要使用旧版本的 `vue init` 功能，你可以全局安装一个桥接工具：

```
npm install -g @vue/cli-init
# `vue init` 的运行效果将会跟 `vue-cli@2.x` 相同
vue init webpack my-project
```

```
npm install -g @vue/cli-init
```

运行以下命令来创建一个新项目：

```
vue create <项目名称>
```

```
D:\BigData\SpringVue\LearnVue>vue create vuedemo

Vue CLI v4.5.4
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
```

手动选择功能选项

```
? Please pick a preset: Manually select features
? Check the features needed for your project:
> (*) Choose Vue version
  (*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  (*) Router
  ( ) Vuex
  ( ) CSS Pre-processors
  ( ) Linter / Formatter
  ( ) Unit Testing
  ( ) E2E Testing
```

空格键选择,回车下一步

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue
? Choose a version of Vue.js that you want to start the
> 2.x
  3.x (Preview)
```

选择版本号 2.x

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue
? Choose a version of Vue.js that you want to start the p
? Use history mode for router? Y (Requires proper server se
? Where do you prefer placing config for Babel, ESLint, e
> In dedicated config files 专门的文件存储配置项目
  In package.json
```

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue
? Choose a version of Vue.js that you want to start the p
? Use history mode for router? (Requires proper server se
? Where do you prefer placing config for Babel, ESLint, e
? Save this as a preset for future projects? (y/N) n
```

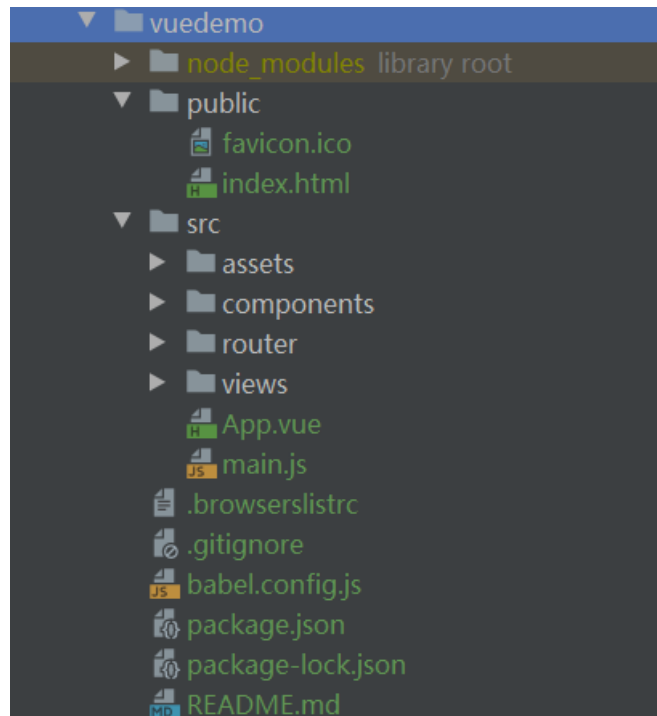
是否保存为预置

```
Initializing git repository...
Installing CLI plugins. This might take a while...

[.....] \ fetchMetadata: sill pacote range ma
```

开始安装

项目创建成功后:目录结构



终端进入项目目录

```
D:\BigData\SpringVue\LearnVue>cd vuedemo
```

安装 Axios

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中。

```
npm install --save axios vue-axios
```

安装完后我们就可用在项目中使用 axios 来处理网络接口请求了。

Axios 示例代码:

```
import Vue from 'vue'
import axios from 'axios'
import VueAxios from 'vue-axios'
```

```
Vue.use(VueAxios, axios)
```

```
Vue.axios.get(api).then((response) => {
  console.log(response.data)
})
```

```
this.axios.get(api).then((response) => {
  console.log(response.data)
})
```

```
this.$http.get(api).then((response) => {  
  console.log(response.data)  
})
```

安装 Vuex

```
npm install vuex --save
```

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。Vuex 也集成到 Vue 的官方调试工具 devtools extension，提供了诸如零配置的 time-travel 调试、状态快照导入导出等高级调试功能。

从一个简单的 Vue 计数应用开始

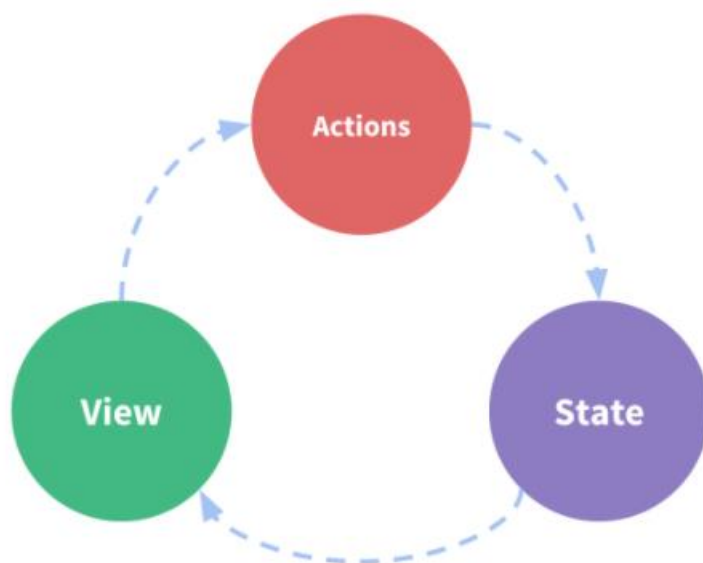
```
new Vue({  
  // state  
  data () {  
    return {  
      count: 0  
    }  
  },  
  // view  
  template: `  
    <div>{{ count }}</div>  
  `,  
  // actions  
  methods: {  
    increment () {  
      this.count++  
    }  
  }  
})
```

这个简单的状态自管理应用包含以下几个部分：

state，驱动应用的数据源；

view，以声明方式将 state 映射到视图；

actions，响应在 view 上的用户输入导致的状态变化。



“单向数据流”理念的简单示意图

但是，当我们的应用遇到多个组件共享状态时，单向数据流的简洁性很容易被破坏：

多个视图依赖于同一状态。

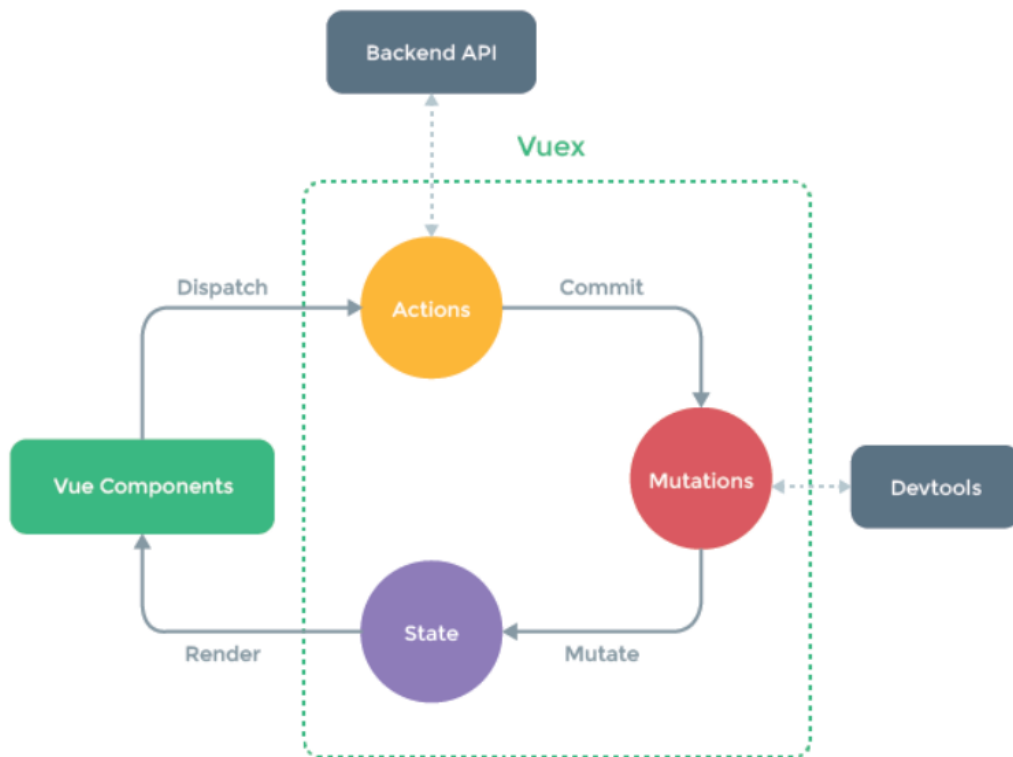
来自不同视图的行为需要变更同一状态。

对于问题一，传参的方法对于多层嵌套的组件将会非常繁琐，并且对于兄弟组件间的状态传递无能为力。对于问题二，我们经常会采用父子组件直接引用或者通过事件来变更和同步状态的多份拷贝。以上的这些模式非常脆弱，通常会导致无法维护的代码。

因此，我们为什么不把组件的共享状态抽取出来，以一个全局单例模式管理呢？在这种模式下，我们的组件树构成了一个巨大的“视图”，不管在树的哪个位置，任何组件都能获取状态或者触发行为！

通过定义和隔离状态管理中的各种概念并通过强制规则维持视图和状态间的独立性，我们的代码将会变得更结构化且易维护。

这就是 Vuex 背后的基本思想，借鉴了 Flux、Redux 和 The Elm Architecture。与其他模式不同的是，Vuex 是专门为 Vue.js 设计的状态管理库，以利用 Vue.js 的细粒度数据响应机制来进行高效的状态更新。



Vuex 使用实例:

```
index.js
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3 Vue.use(Vuex)
4 // 创建一个 store
5 export default new Vuex.Store({ options: {
6   // 单状态实例
7   state: {
8     counter: 0
9   },
10   // 更新状态的方法
11   // mutation内定义的必须是同步函数
12   // 使用store.commit 方法触发
13   mutations: {
14     increment (state) {
15       state.counter++
16     }
17   },
18   // 异步函数定义在actions
19   // 通过 store.dispatch 方法触发
20   // 例如 store.dispatch('increment')
21   actions: {
22     increment ({ commit }) {
23       // 模拟异步
24       setTimeout( handler: () => {
25         commit('increment')
26       }, 1000)
27     }
28   },
29   modules: {
30   }
31 })

main.js
1 import Vue from 'vue'
2 import App from './App.vue'
3 import router from './router'
4 import store from './store'
5
6 Vue.config.productionTip = false
7
8 new Vue({
9   router,
10   // 把 store 对象注入
11   store,
12   render: h => h(App)
13 }).$mount( elementOrSelector: '#app')
```

```
App.vue
1 <template>
2   <div id="app">
3     当前值:{{counter}}
4     <button @click="increment">+</button>
5     <button @click="incrementAsync">+</button>
6   </div>
7 </template>
8 <script>
9   export default {
10     name: 'APP',
11     computed: {
12       counter: function () {
13         // 获取状态值
14         return this.$store.state.counter;
15       }
16     },
17     methods: {
18       increment() {
19         // 触发更新状态
20         this.$store.commit('increment')
21         console.log(this.$store.state.count)
22       },
23       incrementAsync(){
24         // 异步分发事件
25         this.$store.dispatch('increment')
26       }
27     }
28   }
29 </script>
```

什么情况下我应该使用 Vuex?

Vuex 可以帮助我们管理共享状态，并附带了更多的概念和框架。这需要对短期和长期效益进行权衡。

如果您不打算开发大型单页应用，使用 Vuex 可能是繁琐冗余的。确实是如此——如果您的应用够简单，您最好不要使用 Vuex。一个简单的 store 模式就足够您所需了。但是，如果您需要构建一个中大型单页应用，您很可能会考虑如何更好地在组件外部管理状态，Vuex 将会成为自然而然的选择。

Vuex 规定了一些需要遵守的规则：

应用层级的状态应该集中到单个 store 对象中。

提交 mutation 是更改状态的唯一方法，并且这个过程是同步的。

异步逻辑都应该封装到 action 里面。

只要你遵守以上规则，如何组织代码随你便。如果你的 store 文件太大，只需将 action、mutation 和 getter 分割到单独的文件。

对于大型应用，我们会希望把 Vuex 相关代码分割到模块中。下面是项目结构示例：

```
├─ index.html
├─ main.js
├─ api
│  └─ ... # 抽取API请求
├─ components
│  ├─ App.vue
│  └─ ...
└─ store
   ├─ index.js      # 我们组装模块并导出 store 的地方
   ├─ actions.js    # 根级别的 action
   ├─ mutations.js  # 根级别的 mutation
   └─ modules
      ├─ cart.js     # 购物车模块
      └─ products.js # 产品模块
```

第十章 项目前端实现

参见课上代码实现

附录一 :开发工具

JetBrains 旗下的产品:

IntelliJ IDEA 偏重于 Java 开发，旗舰产品，它可以通过（捆绑的或可下载的）插件的方式提供 WebStorm 和 PhpStorm 所有的功能。支持 Scala 和 Groovy 开发，也可以通过插件支持 Ruby 和 Python 语言。

PhpStorm 侧重于 PHP 开发

WebStorm 侧重于 JS 开发

RubyMine 侧重于 Ruby 和 Rails 开发

PyCharm 侧重于 Python 和 Django 开发

WebStorm 与 IntelliJ IDEA 相比，功能少，集中于 JS 开发这一块，更加轻量级，新项目配置起来更简单。

官网: <https://www.jetbrains.com/>

本项目设计前后台开发使用产品 IntelliJ IDEA 下载页面:

<https://www.jetbrains.com/idea/download/#section=windows>

Visual Studio Code:

Visual Studio Code 重新定义和优化了代码编辑，以便生成和调试新式 Web 应用程序和云应用程序。专注于代码的编辑，使用多个光标进行快速的以键盘为中心的高级编辑。代码导航，Regex 查找、大纲显示和窥视定义。使用适用于 Node.js、TypeScript 和 JavaScript 的集成调试工具诊断应用程序存在的问题。设置代码中的断点、中断异常、监视变量、单步执行代码或向上导航至调用堆栈，以及附加到本地运行进程。

官网: <https://www.visualstudio.com/products/code-vs>

Mysql 下载

64 位 5.7

<https://mirrors.huaweicloud.com/mysql/Downloads/MySQL-5.7/mysql-5.7.29-winx64.msi>

32 位 5.7

<https://mirrors.huaweicloud.com/mysql/Downloads/MySQL-5.7/mysql-5.7.29-win32.msi>

Mysql 客户端工具

下载地址

https://www.heidisql.com/installers/HeidiSQL_11.0.0.5919_Setup.exe

Git 代码分布式版本控制工具

下载地址:

<https://git-scm.com/download/win>

JDK

本项目版本要求 JDK8 +

官方下载页面

<https://www.oracle.com/java/technologies/javase-downloads.html>

官方华为镜像

<https://repo.huaweicloud.com/java/jdk/>

开源 JDK 镜像

<https://mirrors.tuna.tsinghua.edu.cn/AdoptOpenJDK/>

JDK8 下载地址

64 位

<https://repo.huaweicloud.com/java/jdk/8u202-b08/jdk-8u202-windows-x64.exe>

32 位

<https://repo.huaweicloud.com/java/jdk/8u202-b08/jdk-8u202-windows-i586.exe>

Maven

Maven 是一个项目管理工具，可以对 Java 项目进行构建、依赖管理。

下载地址

<https://mirrors.huaweicloud.com/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip>

Nodejs

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。Node.js 的包管理器 npm，是全球最大的开源库生态系统。

下载地址

64 位

<https://mirrors.huaweicloud.com/nodejs/latest/node-v14.8.0-x64.msi>

32 位

<https://mirrors.huaweicloud.com/nodejs/latest/node-v14.8.0-x86.msi>