# Mybatis

参考文档 <https://blog.csdn.net/a745233700/article/details/80977133>

## 什么是mybatis

Mybatis是一个半ORM（对象关系映射）框架，它内部封装了JDBC，加载驱动、创建连接、创建statement等繁杂的过程，开发者开发时只需要关注如何编写SQL语句，可以严格控制sql执行性能，灵活度高。

作为一个半ORM框架，MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。

称mybatis是半自动的orm映射框架，是因为在查询关联对象或关联的集合对象时，需要手动编写sql来完成。

Hibernate全自动orm映射工具，hibernate查询关联的对象或者关联集合对象时，可以根据对象关系模型直接获取

## Mybatis的优缺点

### 优点

（1）基于SQL语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL写在XML里，解除sql与程序代码的耦合，便于统一管理；提供XML标签，支持编写动态SQL语句，并可重用。

（2）与JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；

（3） 很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持）。

（4）能够与Spring很好的集成；

（5）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，支持对象关系组件维护。

### 缺点

（1）SQL语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写SQL语句的功底有一定要求。

（2）SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库

## #{}和${}的区别是什么

${}是字符串替换，#{}是预处理；

Mybatis在处理${}时，就是把${}直接替换成变量的值。而Mybatis在处理#{}时，会对sql语句进行预处理，将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

使用#{} 可以防止sql注入，提高系统的安全性

## Mybatis 是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？

（1）<select> 标签使用 resultType 参数，传递 Java 类，sql 中 select 的字段名保持与 Java 类属性名称一致

（2）使用sql列的别名功能，将列的别名书写为对象属性名。

## Mybatis动态sql有什么用？执行原理？有哪些动态sql

Mybatis动态sql可以在Xml映射文件内，以标签的形式编写动态sql，执行原理是根据表达式的值 完成逻辑判断 并动态拼接sql的功能。

Mybatis提供了9种动态sql标签：trim | where | set | foreach | if | choose | when | otherwise | bind

## Xml映射文件中，除了常见的select、insert、update、delete 标签之外，还有哪些标签

<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态sql的9个标签，其中<sql>为sql片段标签，通过<include>标签引入sql片段，<selectKey>为不支持自增的主键生成策略标签。

## 使用mybatis的mapper接口调用时有哪些要求

 Mapper接口方法名和mapper.xml中定义的每个sql的id相同；

 Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同；

 Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同；

 Mapper.xml文件中的namespace即是mapper接口的类路径。

## 为什么说mybatis是半自动的orm映射工具？它与全自动的有什么区别？

Hibernate 属于全自动的orm映射工具，使用hibernate 查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以是全自动的

Mybatis 在查询关联对象或者关联的集合对象时，需要手动编写sql完成，所以称为半自动的orm映射工具

## 在mapper中如何传递多个参数

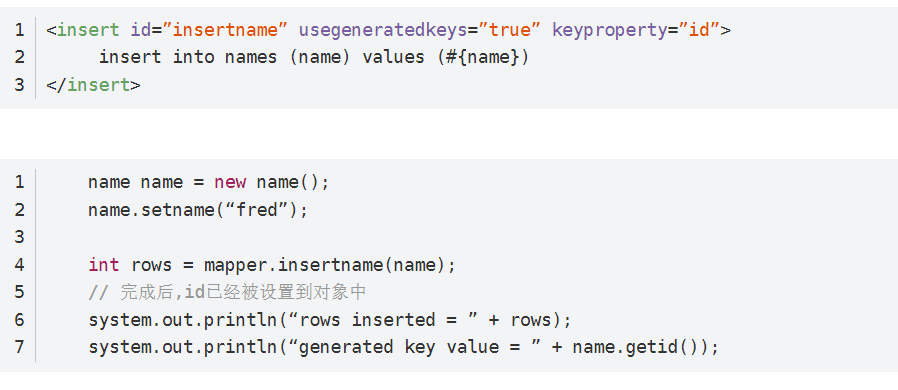
将多个参数封装成map

使用@Param

使用javabean

## 如何获取自动生成的主键值

需要在我们 标签上加 usegeneratedkeys 等于true 再加上 keyproperty 等于 id 就可以了”

xuy

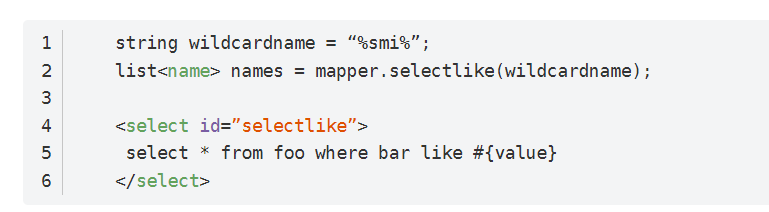
## 当实体类中的属性名和表中的字段名不一样，怎么办？

通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。

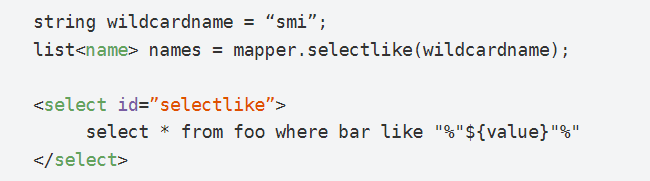
通过<resultMap>来映射字段名和实体类属性名的一一对应的关系

## 模糊查询like语句该怎么写

在Java代码中添加sql通配符。



在sql语句中拼接通配符，会引起sql注入



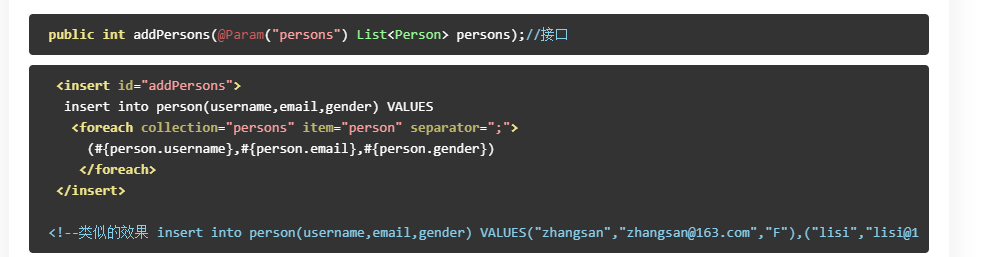
## Mybatis 动态sql有什么用？执行原理？有哪些动态sql?

Mybatis动态sql可以在Xml映射文件内，以标签的形式编写动态sql，执行原理是根据表达式的值 完成逻辑判断 并动态拼接sql的功能。

Mybatis提供了9种动态sql标签：trim | where | set | foreach | if | choose | when | otherwise | bind。

## Mybatis 批量新增

通过for each标签拼接sql



## Mybatis 和hibernate 的区别

MyBatis容易掌握，只要会写SQL语句就行了。而Hibernate门槛较高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用好Hibernate方面需要你的经验和能力都很强才行。

MyBatis可以进行更为精细的SQL优化，而Hibernate要进行SQL优化会比较困难一些。

Hibernate无需写SQL就可以完成单表简单增删改查，而MyBatis单表简单增删改查还需要自己写SQL语句。

Hibernate数据库移植性很好，不同的数据库需要写不同SQL。

# Hibernate 框架

## 为什么要使用hibernate

使用Hibernate框架就不用我们写很多繁琐的SQL语句。**Hibernate实现了ORM，能够将对象映射成数据库表，从而简化我们的开发**

## Hibernate 中怎样实现类之间的关系？（一对多、多对多）

它们通过配置文件中的 many-to-one、one-to-many、many-to-many

## Hibernate 的三种状态之间如何转换

临时状态、持久化状态、游离状态

临时状态

在创建对象在new之后，在save之前的状态

Session中和数据库中都不存在该数据

持久化状态

在save之后，session关闭之前，session中和数据库中都存在对应的数据

游离状态

Session关闭之后，session不存在，数据中有对应的数据

## Hibernate get和load的区别

Get立即查询

get立即加载 查询时立即发送sql语句，当访问实体不存在是返回null

load懒加载

访问对象时发送sql语句，访问实体不存在时返回（对象不存在异常）,如果设置了lazy=“false”,load就不再进行延迟加载

## Jdbc和hibernate的区别

hibernate是JDBC的轻量级封装，hibernate基于jdbc

**jdbc**:

jdbc是纯手工的原生sql语句

执行效率高于hibernate

jdbc可以直接操作数据库的表如:select \* from 表名

**hibernate：**

hibernate是全自动

跨数据库

开发效率高于jdbc

使用的是hql语句最终会转化为sql语句.语法是 from 对象名

## Hibernate 运行原理

1.通过configuration加载了hibernate.cfg.xml文件。

2.加载后创建sessionFactory.(sessionFactory是线程安全的)

3.然后同sessionFactory获得session。（创建会话，相当于jdbc中的connection 是线程不安全的）

4.通过session操作数据库，最后通过transaction来进行事物的控制.

## Hibernate 主键生成策略

1. **increment：先选出数据库当前最大id，在此基础上加1，hibernate会自行处理，适用于int，short，long类型的主键**
2. **sequence：oracle数据库会自行处  理，会增加一个序列配置,我们需要自己配置一个序列标签**

<generator class="sequence">

        <param name="sequence">sequence\_id</param>

</generator>

**3.native：通用多个数据库，数据库本地生成策略，数据库会自动生成一个序列,默认一个序列**

**4.uuid：生成一个32位不重复的字符串，可以跨数据库使用**

## Hibernate 五大核心接口

Configuration(类):

加载配置文件hibernate.cfg.xml文件中的配置信息，从而得到：

hibernate的底层信息：

数据库连接，jdbc驱动，方言，用户名，密码。

hibernate的映射文件(\*.hbm.xml)

sessionFactory（接口）

通过configuration创建sessionFactory。

可以用来获得session。

sessionFactory是线程安全的，

里边加载的数据信息有数据库的配置信息和映射关系.

Session(接口)

线程不安全。

相当于jbdc的connection.

用来操作数据库.

Transaction（接口）

进行事物的控制实现事物的commit（提交)和rollback（回滚）

Query(接口)

 用来执行数据查询的操作

## Jpa 框架

jpa是个规范 不是一个框架

**1.Jpa是什么**

JPA即Java持久化API，JPA的出现主要是为了简化持久层（Dao层）开发以及整合ORM技术，结束Hibernate、TopLink(套铺赁克)、JDO等ORM框架各自为营的局面。

JPA包括以下3方面的技术：

* **ORM映射**：支持XML和注解描述对象和表之间的映射关系
* **API**：操作实体对象来执行CRUD(create read update delete)(增删改查)操作
* **查询语言**：通过面向对象而非面向数据库的查询语言（**JPQL**）查询数据

**Spring Data Jpa的使用**

1. **在项目的pom.xml中添加spring-data-jpa的依赖配置**
2. **在spring核心配置文件中配置spring-data-jpa相关项**
3. **编写实体类并使用JPA注解配置实体类和数据库表的映射关系**
4. **编写符合 Spring Data JPA 规范的 Dao 层接口**

**2.**   **Spring Data JPA是什么**

Spring Data JPA 是 Spring 基于 **ORM 框架**、**JPA 规范**的基础上封装的一套**JPA应用框架**，可使开发者用**极简的代码**即可实现对**数据的访问和操作**。Spring Data JPA不需要过多的关心Dao层的实现，只需关注我们继承的接口，按照一定的规则去编写我们的接口即可。

**3.**   **Spring Data JPA的核心接口**

* **Repository(瑞胖丝特瑞)**: 所有接口的父接口，而且是一个空接口，目的是为了统一所有Repository的类型，让组件扫描的时候能进行识别。
* **CrudRepository**：是Repository的子接口，提供CRUD(增删改查)的功能。
* **Paging(培经)AndSorting(消停)Repository**：是CrudRepository的子接口，添加分页和排序的功能。
* **JpaRepository**：是PagingAndSortingRepository的子接口，增加了一些实用的功能，例如批量操作。
* **JpaSpecification(丝拍谁非K鑫)Executor(A克丝Q特儿)**：用来做负责查询的接口。
* **Specification(丝拍谁非K鑫)**：是Spring Data JPA提供的一个查询规范，要做复杂的查询，只需围绕这个规范来设置查询条件即可。

**4.**   **JPA中常用的注解**

@Entity【嗯忒忒】 标识当前类为实体类

@Table 配置与实体类相对应的数据库表信息(如表名)

@Id 标识当前字段为主键

@Generated(杰呢瑞忒德)Value 配置主键生成策略

@Column(靠乐木) 配置属性对应的数据库字段信息(如字段名)

@ManyToOne【美图玩】 配置一对多关系

@ManyToMany【没图没】 配置多对多关系

@Transient这个注解表示忽略这个属性，那么建表的时候就不会在数据库表中生成一个对应的字段了！

@JoinColumn外键

**5.**   **JPA主键生成策略**

**Identity(爱den忒忒**):主键由数据库自动生成 （主要是自动增长型）

**Sequence(谁困死):**根据底层数据库的序列来生成主键，条件是数据库支持序列

**Table**:使用特定的数据表格来保存主键

**Auto**:由持久化引擎来决定从以上三种策略中选择最合适的

**6.**   **Spring Data Jpa查询方法命名规则**

如果spring data jpa现有的查询方法无法满足你的需求，你只需要在dao层接口上按照spring data jpa的查询方法命名规则去编写方法，spring data jpa会自动解析你的方法名，从而去执行对应的查询，查询方法命名规则(掌握5个)如下:

* **And:**等价于 SQL 中的 and 关键字，比如 findByUsernameAndPassword(String user, Striang pwd)；
* **Or :**等价于 SQL 中的 or 关键字，比如 findByUsernameOrAddress(String user, String addr)；
* **Between :**等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min)；
* **LessThan:**等价于 SQL 中的 "<"，比如 findBySalaryLessThan(int max)；
* **GreaterThan:**等价于 SQL 中的">"，比如 findBySalaryGreaterThan(int min)；
* **IsNull :**等价于 SQL 中的 "is null"，比如 findByUsernameIsNull()；
* **IsNotNull:**等价于 SQL 中的 "is not null"，比如 findByUsernameIsNotNull()；
* **NotNull:**与 IsNotNull 等价；
* **Like :**等价于 SQL 中的 "like"，比如 findByUsernameLike(String user)；
* **NotLike :**等价于 SQL 中的 "not like"，比如 findByUsernameNotLike(String user)；
* **OrderBy:**等价于 SQL 中的 "order by"，比如 findByUsernameOrderBySalaryAsc(String user)；
* **Not:**等价于 SQL 中的 "!="，比如 findByUsernameNot(String user)；
* **In :**等价于 SQL 中的 in"，比如 findByUsernameIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；
* **NotIn:**等价于 SQL 中的 "not in"，比如 findByUsernameNotIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

# springMvc 框架

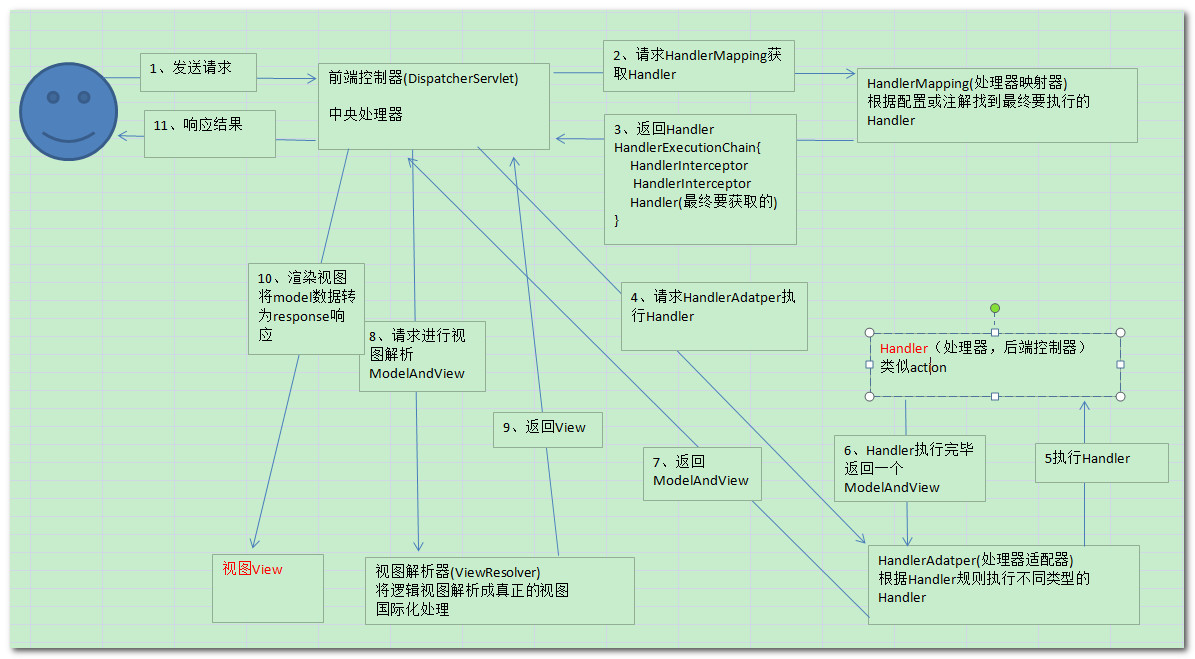
## 什么是springmvc框架

Spring MVC是一个基于Java的实现了MVC设计模式的请求驱动类型的轻量级Web框架，通过把模型-视图-控制器分离，将web层进行职责解耦，把复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

springMvc 处于我们mvc设计模式的 controller层

springMvc 处于我们的controller 控制层，主要是用来接参传参页面跳转的

## springMvc 运行原理



（1）用户发送请求至前端控制器DispatcherServlet；

（2）DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handler；

（3）处理器映射器根据请求url找到具体的处理器Handler，生成处理器对象及处理器拦截器(如果有则生成)，一并返回给DispatcherServlet；

（4）DispatcherServlet 调用 HandlerAdapter处理器适配器，请求执行Handler；

（5）HandlerAdapter 经过适配调用 具体处理器进行处理业务逻辑；

（6）Handler执行完成返回ModelAndView；

（7）HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet；

（8）DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析；

（9）ViewResolver解析后返回具体View；

（10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）

（11）DispatcherServlet响应给用户

## springMvc 接收值有几种

    1.request.getParameter

    2.自定义属性

    3.对象接收

    4.restful

## springMvc返回页面

    1.reqeust

    2.modelAndView

    3.String 返回页面

## SpringMvc带值返回页面

    1.reqeust

    2.modelandview

    3.model

    4.modelMap

## springMvc 的常用注解有哪些

@RequestMapping(" /query") 配置的就是请求路径。可以加到方法上，也可以加到类上

@PathVariable 加在参数上，@RequestMapping(value="/params4/{username}")

@RequestParam 加在参数上使用，可以控制该参数是否必须传递

@ResponseBody 加载方法上，返回的是出数据,同时它依赖jackson的jar包对返回值进行json化的处理

## SpringMvc怎么设定重定向和转发

（1）转发：在返回值前面加"forward:"，譬如"forward:user.do?name=method4"

（2）重定向：在返回值前面加"redirect:"，譬如"redirect:http://www.baidu.com"

## springMvc 转发重定向区别

**重定向**

1、地址栏会发生变化

2、数据不共享

3、执行效率低、好几次请求

**转发**

1、地址栏不会发生变化

2、数据共享

3、执行效率高、一次请求

Springmvc 默认是 转发

# SpringBoot 框架

## 什么是springboot

Springboot 是由Pivotal（呸v套） 团队提供的全新框架。springBoot 是所有基于Spring Framework 5.0 开发的项目起点，springBoot 的设计是为了让我们尽可能快的跑起来spring应用程序，并且尽可能减少配置文件

Spring boot 并不是一个全新的框架，而是将已有的spring组件整合起来。

Spring boot 可以说是遵循约定优于配置这个理念产生的。它的特点非常简单，快速和便捷

Spring boot 简化了基于spring的应用开发，通过少量的代码就能创建一个独立的、产品级别的spring应用。Spring boot为spring平台及第三方库提供开箱即用的设置

**Springboot 特点**

1、为所有spring开发提供一个更快更广泛的入门体验

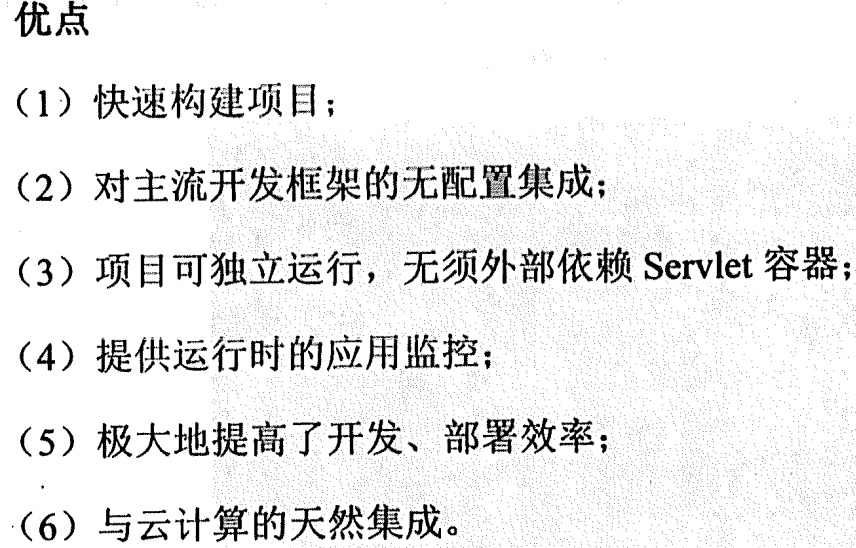
2、零配置。无冗余代码生成和XML强制配置，遵循“约定大于配置”。

3、集成了大量常用的第三方库的配置，spring Boot 应用为这些第三方库提供了几乎可以零配置的开箱即用的能力

4、提供了一系列大型项目常用的非功能性特征，如嵌入式服务器、安全性、度量、运行状况检查、外部化配置

5、spring Boot 不是spring的替代者，spring框架是通过ioc机制来管理bean的。Spring Boot依赖spring框架来管理对象的依赖。Spring Boot并不是spring的精简版，而是为spring做好各种产品级准备

## springBoot优点



## Spring Boot 重要的策略(开箱即用 约定优于配置)

**开箱即用**

Outofbox  是指在开发过程中，通过在maven项目的pom文件中添加相关的依赖包，然后使用对应的注解来替代繁琐的xml配置文件以管理对象的生命周期。

3、这个特点使得开发人员摆脱了

复杂的配置工作以及依赖管理工作，更加注重专注于业务逻辑

**约定优于配置**

是一种由spring Boot本身来配置目标结构，由开发者在结构中添加信息的软件设计范式，这一特点虽降低了部分灵活性，增加了bug定位的复杂性。但是减少了开发人员需要做出决定的数量，同时减少了大量的xml配置，

并且可以将代码编译、测试和打包等工作自动化

## springBoot注解

@SpringBootApplication    声明springboot的启动类

### @SpringBootApplication 注解的执行流程

* @springbootApplication 是一个组合注解 @Configuration @EnableAutoConfiguration @ComponentScan
* （1）@EnableAutoConfiguratio：启用 SpringBoot 的自动配置机制也就是自动扫描其他注解让其生效。
* （2）@ComponentScan：扫描被@Component (@Service,@Controller)注解的 bean，注解默认会扫描该类所在的包下所有的类
* （3）@Configuration：允许在 Spring 上下文中注册额外的 bean 或导入其他配置类。

@RestController      controller类上的注解   加上此注解 代表当前类 加@Controller   当前类所有的方法 默认加上 @ResponseBody

@MapperScan(“包路径”)   扫描dao层包路径 让我们dao接口 不用加注解

@GetMapping            代表当前的方法请求 必须为get请求

@PostMapping           代表当前的方法请求必须为post请求

@DeleteMapping         代表当前方法请求必须为delete请求

@PathVariable            声明方法参数

# Vue框架

## 什么是vue

Vue是一套用于构建用户界面的渐进式框架。 Vue被设计可以自底向上逐层应用。Vue的核心库只关注视图层，易于上手，还便于与第三方库或既有的项目整合。 另一方面，当与[现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)以及各种[支持类库](https://github.com/vuejs/awesome-vue#libraries--plugins)结合使用时 Vue完全能够为复杂的单页应用提供驱动

## Vue路由

**路由的工作原理**

工作原理：

1.解析地址栏 完整的页面地址、路由地址

2.根据路由地址 从路由词典中找到真正的要加载的页面

3.发起ajax请求  请求要加载的页面

4.像指定的容器中 插入加载来的页面

**使用路由模块来实现页面跳转的方式**

方式1：直接修改地址栏

方式2：this.$router.push(‘路由地址’);

方式3：<router-link to="路由地址"></router-link>

**路由模块参数传递**

在页面之间跳转的时候，在有些场景下，比如说list --> detail 需要同时指定参数

1.配置接收方的路由地址 /detail --> /detail/:index

2.发送

    <router-link to="/detail/20" />

    this.$router.push('/detail/20')

3.接收

    this.$route.params.index

## vue钩子函数

|  |  |  |
| --- | --- | --- |
| 钩子函数 | 触发的行为 | 在此阶段可以做的事情 |
| beforeCreadted | vue实例的挂载元素$el和数据对象data都为undefined，还未初始化。 | 加loading事件 |
| created | vue实例的数据对象data有了，$el还没有 | 结束loading、请求数据为mounted渲染做准备 |
| beforeMount | vue实例的$el和data都初始化了，但还是虚拟的dom节点，具体的data.filter还未替换。 | .. |
| mounted | vue实例挂载完成，data.filter成功渲染 | 配合路由钩子使用 |
| beforeUpdate | data更新时触发 |  |
| updated | data更新时触发 | 数据更新时，做一些处理（此处也可以用watch进行观测） |
| beforeDestroy | 组件销毁时触发 |  |
| destroyed | 组件销毁时触发，vue实例解除了事件监听以及和dom的绑定（无响应了），但DOM节点依旧存在 |  |



## Vue 组件

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。

在 Vue 中，父子组件通过 props 传递通信，从父向子单向传递。子组件与父组件通信，通过触发事件通知父组件改变数据。这样就形成了一个基本的父子通信模式。

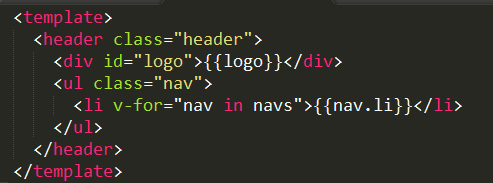
在开发中组件和 HTML、JavaScript 等有非常紧密的关系时，可以根据实际的需要自定义组件，使开发变得更加便利，可大量减少代码编写量。

组件还支持热重载（hotreload）。当我们做了修改时，不会刷新页面，只是对组件本身进行立刻重载，不会影响整个应用当前的状态。CSS 也支持热重载。

**父组件向子组件传递数据**

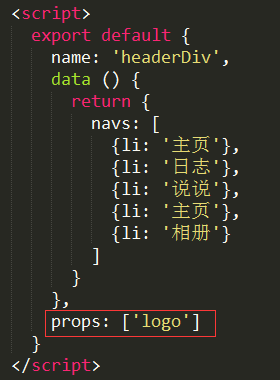
在 Vue 中，可以使用 props 向子组件传递数据。

子组件部分



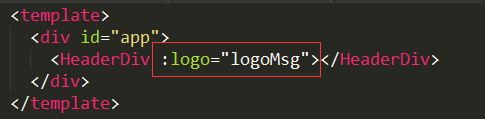
这是 header.vue 的 HTML 部分，logo 是在 data 中定义的变量。

如果需要从父组件获取 logo 的值，就需要使用 **props: ['logo']**



**在 props 中添加了元素之后，就不需要在 data 中再添加变量了**

父组件部分：



在调用组件的时候，使用 v-bind 将 logo 的值绑定为 App.vue 中定义的变量 logoMsg



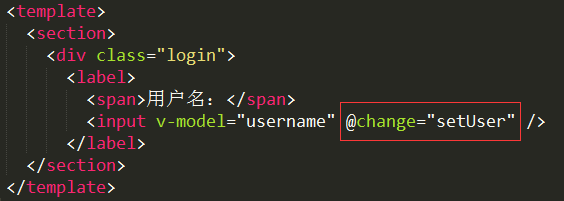
然后就能将App.vue中 logoMsg 的值传给 header.vue 了：

**子组件向父组件传递数据**

 子组件主要通过事件传递数据给父组件

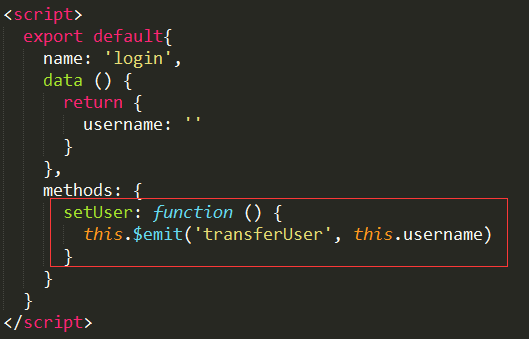
子组件部分：

子组件部分：



这是 login.vue 的 HTML 部分，当<input>的值发生变化的时候，将 username 传递给 App.vue

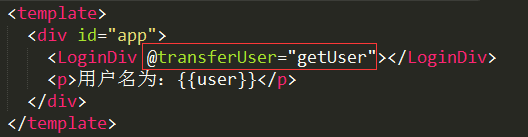
首先声明一个了方法 setUser，用 change 事件来调用 setUser



在 setUser 中，使用了 **$emit**来遍历 transferUser 事件，并返回 this.username

其中 transferUser 是一个自定义的事件，功能类似于一个中转，this.username 将通过这个事件传递给父组件

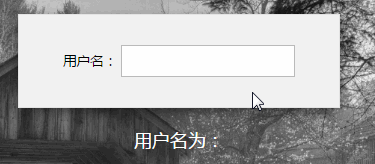
父组件部分：



在父组件 App.vue 中，声明了一个方法 getUser，用 transferUser 事件调用 getUser 方法，获取到从子组件传递过来的参数 username



getUser 方法中的参数 msg 就是从子组件传递过来的参数 username



## Vue指令

v-html 把文本变成可执行的html代码

v-if 条件渲染

v-bind 绑定属性

v-for 循环

**所谓指令，其实本质就是在模板中出现的特殊标记，根据这些标记让框架知道需要对这里的 DOM 元素进行什么操作。**

### v-text

* 解释：更新元素的 textContent，将数据解析为纯文本

<h1 v-text="msg"></h1>  
  
这里v是vue的前缀，text是指令ID，msg是expression。 msg作为ViewModel，当它的值发生改变时，就触发指令text，重新计算标签的textContent(innerText)。

### Mustache{{}}

* 解释：可以局部更新元素的 textContent，将数据解析为纯文本

<div id="app">

<p>苹果的价格为{{ msg }}元</p>

</div>

### v-html

* 解释：更新元素的 innerHTML，将数据解析成html标签

<h1 v-html="msg"></h1>

### v-bind

* 作用：绑定属性
* 语法：v-bind:title="msg"
* 简写：:title="msg"

### v-on

* 作用：绑定事件
* 语法：v-on:click="say" or v-on:click="say('参数', $event)"
* 简写：@click="say"
* 说明：绑定的事件从methods中获取

使用逗号分割绑定多个事件

<div v-on="click:onClick, keyup:onKeyup, keydown:onKeydown"></div>

### 事件修饰符

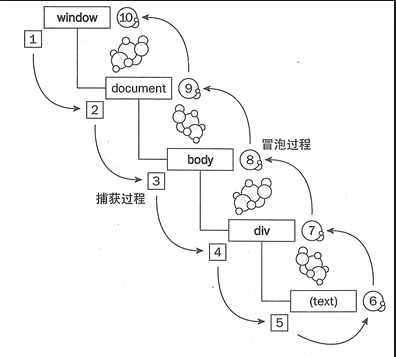
* .stop 阻止冒泡，调用 event.stopPropagation() // 比如说一个div下有一个弹窗和一个表单 点击提交按钮点击了<input type="submit" />提交表单信息 点击弹出层周边空白区域，关闭弹窗 ，当表单显示在弹窗层上方时，为防止弹窗被意外关闭，需要阻止表单提交按钮冒泡行为
* .prevent 阻止默认事件，调用 event.preventDefault() // 比如说点击了链接标签a,在跳转之前，要修改一些URL参数
* .capture 添加事件侦听器时使用事件捕获模式  // 捕获阶段先于冒泡，如果在尽量顶层处理事件，然后阻止传播，可以略微节约性能开销。scroll/resize 这类可能连续触发的事件不冒泡的原因
* .self 只当事件在该元素本身（比如不是子元素）触发时触发回调
* .once 事件只触发一次
* 按键修饰符

<input v-on:keyup.13="submit">   
记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

<input v-on:keyup.enter="submit">

<input @keyup.enter="submit">

按键别名包括：.enter .tab .delete (捕获 “删除” 和 “退格” 键) .esc .space .up .down .left .right.ctrl .shift .meta（windows 键，mac-command 键，)



* .native 在父组件中给子组件绑定一个原生的事件，不加'. native'事件是无法触发的

<div id="app">

　　<my-component @click.native="clickFun"></my-component>

</div>

[复制代码](javascript:void(0);)

Vue.component('my-component', {

　　template: `<a href='#'>click me</a>`

})

new Vue({

　　el: '#app',

　　methods: {

　　　　clickFun: function(){

　　　　　　console.log("message: success")

　　　　}

　　}

})

[复制代码](javascript:void(0);)

### v-model

* 作用：在表单元素上创建双向数据绑定
* 说明：监听用户的输入事件以更新数据

<input v-model="message" placeholder="edit me">

<input type='radio' v-model="radioVal" value='单选按钮' placeholder="edit me">

<!-- checkboxArr的值必须为数组 -->

<input type='checkbox' v-model="checkboxArr" value='复选按钮' placeholder="edit me">

### v-for

* 作用：基于源数据多次渲染元素或模板块

[复制代码](javascript:void(0);)

<div v-for="item in items">

{{ item.text }}

</div>

<!-- 遍历数组 item 为当前项，index 为索引 -->

<p v-for="(item, index) in list">{{item}} -- {{index}}</p>

<!--遍历对象 item 为值，key 为键，index 为索引 -->

<p v-for="(item, key, index) in obj">{{item}} -- {{key}}</p>  
<!-- 遍历常量 item为从1开始的递增值 -->

<p v-for="item in 10">{{item}}</p>

[复制代码](javascript:void(0);)

### v-for的key属性

* 推荐：使用 v-for 的时候提供 key 属性，以获得性能提升。
* 说明：使用v-for更新已渲染的元素列表时,默认用就地复用策略;列表数据修改的时候,它会根据key值去判断某个值是否修改,如果修改,则重新渲染这一项,否则复用之前的元素; 我们在使用的使用经常会使用index(即数组的下标)来作为key,但其实这是不推荐的一种使用方法;

[复制代码](javascript:void(0);)

const list = [

{

id: 1,

name: 'test1',

},

{

id: 4,

name: '我是插队的那条数据',

}

{

id: 2,

name: 'test2',

},

{

id: 3,

name: 'test3',

},

]

之前的数据 之后的数据

key: 0 index: 0 name: test1 key: 0 index: 0 name: test1

key: 1 index: 1 name: test2 key: 1 index: 1 name: 我是插队的那条数据

key: 2 index: 2 name: test3 key: 2 index: 2 name: test2

key: 3 index: 3 name: test3

[复制代码](javascript:void(0);)

// 不推荐

<div v-for="(item, index) in list" :key="index" >{{item.name}}</div>

//推荐

<div v-for="(item, index) in list" :key="item.id" >{{item.name}}</div>

### v-class和v-style

* 说明：这两个都是HTML元素的属性，使用v-bind，只需要通过表达式计算出字符串结果即可
* 表达式的类型：字符串、数组、对象
* 语法：

[复制代码](javascript:void(0);)

<!-- 1 对象书写方式-->

<div v-bind:class="{ active: true }"></div> ===>

<div class="active"></div>

<!-- 2 数组书写方式-->

<div :class="['active', 'text-danger']"></div> ===>

<div class="active text-danger"></div>

<!-- 3 数组对象结合-->

<div v-bind:class="[{ active: true }, errorClass]"></div> ===>

<div class="active text-danger"></div>

<!-- style -->

<!-- 1 对象书写方式 -->

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

<!-- 2 数组书写方式-->

<div v-bind:style="[baseStyles, overridingStyles]"></div>

[复制代码](javascript:void(0);)

### v-if 和 v-show

* v-if：根据表达式的值的真假条件，销毁或重建元素 v-if适合条件不大可能改变的场景
* v-show：根据表达式之真假值，切换元素的 display CSS 属性，dom元素一直在  v-show适合频繁切换

### v-cloak

* 这个指令保持在元素上直到关联实例结束编译。和 CSS 规则如 [v-cloak] { display: none } 一起用时，这个指令可以隐藏未编译的 Mustache 标签直到实例准备完毕。
* 防止刷新页面，网速慢的情况下出现{{ message }}等数据格式

<div v-cloak>

{{ message }}

</div>

### v-pre

* 说明：跳过这个元素和它的子元素的编译过程。可以用来显示原始 Mustache 标签。跳过大量没有指令的节点会加快编译。

<span v-pre>{{ this will not be compiled }}</span>

### v-once

* 说明：只渲染元素和组件一次。随后的重新渲染，元素/组件及其所有的子节点将被视为静态内容并跳过。这可以用于优化更新性能。

<span v-once>This will never change: {{msg}}</span>