项目简介

项目主要范围是开发一套软件，基于Sass模式，为各种客户提供租赁服务。系统包括商业信息的搜集、管理和分析，数据转换管理，可视化的图形、图标和计量表展示等功能，实现生产现场数据的可视化透明化管控，从而监控、协同、调度、分析、预警制造执行的过程，实现生产管控的一体化和可视化。

技术栈

前后端分离，所需技术栈如下所示：

前端	后端
vue	Springboot
element-ui	Mybatis plus
axios	Shiro
	Redis
	Hibernate Validatior
	JWT

后端框架：Springboot

Springboot是目前最流行的后端框架。很多知名互联网公司都在使用Springboot，比如阿里、美团；
Springboot的核心特性可以总结为：
1、快捷，可快速构建一个项目方便对外输出各种形式的服务；
2、灵活，开箱即用；
3、提供了一些大型项目中常见的非功能性特性，如内嵌服务器、安全指标、健康检测、外部化配置等。

ORM框架：Mybatis plus

ORM（Object Relational Mapping）对象关系映射，是一种程序设计技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。从效果上说，它其实是创建了一个可在编程语言里使用的“虚拟对象数据库”。

整合mybatis plus，让项目能完成基本的增删改查操作，步骤很简单：可以去官网看看。

前端框架：vue

开发步骤

前端：

vue 整合 element-ui,axios

后端:

Springboot整合Mybatis plus，增删改查

pom文件：

```
<!--mysql-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<!--Mybatis plus-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.2.0</version>
</dependency>
<!--Mybatis plus 代码生成器-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

配置文件：

```
#mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/vueblog?
useUnicode=true&useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai
spring.datasource.username=root
spring.datasource.password=admin

#配置myabtis plus的mapper的xml文件的扫描路径
mybatis-plus.mapper-locations=classpath*:/mapper/**Mapper.xml
```

开启mapper接口扫描，添加分页插件：

```
@Configuration
@EnableTransactionManagement
@MapperScan("com.markerhub.*.mapper")
public class MybatisPlusConfig {
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor paginationInterceptor = new
PaginationInterceptor();
        return paginationInterceptor;
    }
}
```

利用mybatis plus 代码生成：详细代码 CodeGenerator

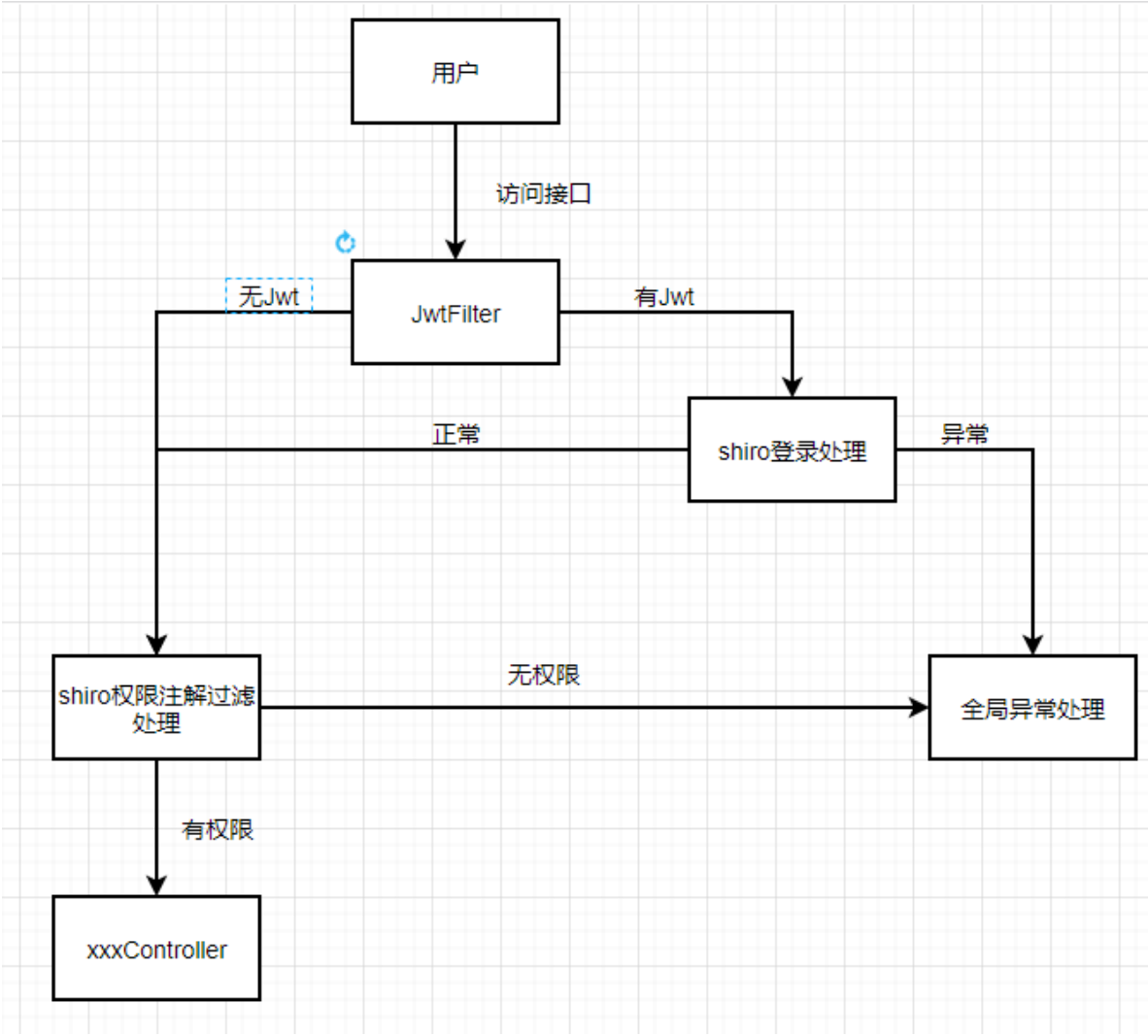
> 运行CodeGenerator的main方法，输入表名，即可生成对应controller service mapper entity。

Springboot整合Shiro + JWT，身份校验，会话共享

考虑到后面可能需要做集群、负载均衡等，所以就需要会话共享，而shiro的缓存和会话信息，我们一般考虑使用redis来存储这些数据，所以，我们不仅仅需要整合shiro，同时也需要整合redis。在开源的项目中，我们找到了一个shiro-redis-spring-boot-starter可以快速整合shiro-redis，配置简单，具体教程可以看官方文档。

而因为我们需要做的是前后端分离项目的骨架，所以一般我们会采用token或者jwt作为跨域身份验证解决方案。所以整合shiro的过程中，我们需要引入jwt的身份验证过程。

流程图：



pom文件:

```
<!-- shiro-redis -->
<dependency>
    <groupId>org.crazycake</groupId>
    <artifactId>shiro-redis-spring-boot-starter</artifactId>
    <version>3.2.1</version>
</dependency>
<!-- jwt -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
<!-- hutool工具类 -->
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.3.3</version>
</dependency>
```

配置文件:

```
shiro-redis:
  enabled: true
  redis-manager:
    host: 127.0.0.1:6379
cps:
  jwt:
    # 加密秘钥
    secret: f4e2e52034348f86b67cde581c0f9eb5
    # token有效时长，7天，单位秒
    expire: 604800
    header: token
```

配置文件ShiroConfig：

1.引入RedisSessionDAO和RedisCacheManager，为了解决shiro的权限数据和会话信息能保存到redis中，实现会话共享。

2.重写了SessionManager和DefaultWebSecurityManager，同时在DefaultWebSecurityManager中为了关闭shiro自带的session方式，我们需要设置为false，这样用户就不再能通过session方式登录shiro。后面将采用jwt凭证登录。

3.在ShiroFilterChainDefinition中，我们不再通过编码形式拦截Controller访问路径，而是所有的路由都需要经过JwtFilter这个过滤器，然后判断请求头中是否含有jwt的信息，有就登录，没有就跳过。跳过之后，有Controller中的shiro注解进行再次拦截，比如@RequiresAuthentication，这样控制权限访问。

Springboot异常处理，统一结果封装

pom文件：

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

异常处理：

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.validation.BindingResult;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;

/**
 * @Author: GYX
 * @Description: 全局异常处理
```

```
 * * Slf4j注解 lombok日志注解 相当于private final Logger logger =
LoggerFactory.getLogger(XXX.class);
 * * RestControllerAdvice注解 是@ControllerAdvice和@ResponseBody的合并 通过对异常的拦
截实现的统一异常返回处理
 * * ResponseStatus注解 指定返回的状态码
 * * ExceptionHandler注解 拦截指定类型的异常
 * @Date: 2021/8/21 14:47
 * @Version: 1.0
 */
@Slf4j
@RestControllerAdvice
public class GlobalExcepitonHandler {
    // 捕捉shiro的异常 401
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    @ExceptionHandler(ShiroException.class)
    public Result handle401(ShiroException e) {
        return Result.fail(401, e.getMessage(), null);
    }
    //Assert的异常处理 400客户端错误
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(value = IllegalArgumentException.class)
    public Result handler(IllegalArgumentException e) {
        return Result.fail(e.getMessage());
    }
    //@Validated 校验错误异常处理 400客户端错误
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(value = MethodArgumentNotValidException.class)
    public Result handler(MethodArgumentNotValidException e) {
        BindingResult bindingResult = e.getBindingResult();
        ObjectError objectError =
bindingResult.getAllErrors().stream().findFirst().get();
        return Result.fail(objectError.getDefaultMessage());
    }
    //运行时异常处理 500服务器错误
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler(value = RuntimeException.class)
    public Result handler(RuntimeException e) {
        log.error("运行时异常:-------------->", e);
        return Result.fail(HttpStatus.INTERNAL_SERVER_ERROR, e.getMessage());
    }
}
```

统一结果封装：

约定前后端交互格式，方便前端在拦截请求后做出统一的操作，可定义前置拦截器，如无权限跳转；后置拦截器，如异常弹窗。

```
package cc.orientech.cps.common;

import lombok.Data;
import org.springframework.http.HttpStatus;
import org.springframework.util.Assert;

import java.io.Serializable;

/**
 * @Author: GYX
```

```
 * @Description: 统一结果封装
 * @Date: 2021/8/21 14:47
 * @Version: 1.0
 */
@Data
public class Result implements Serializable {
    private int code;
    private String msg;
    private Object data;

    public static Result ok(Object data) {
        return ok("操作成功", data);
    }
    public static Result ok(String msg, Object data) {
        Result m = new Result();
        m.setCode(HttpStatus.OK.value());
        m.setData(data);
        m.setMsg(msg);
        return m;
    }
    public static Result fail(String msg) {
        return fail(msg, null);
    }
    public static Result fail(HttpStatus httpStatus, String msg) {
        Assert.notNull(httpStatus,"httpStatus不能为空");
        return fail(httpStatus.value(), msg, null);
    }
    public static Result fail(String msg, Object data) {
        return fail(HttpStatus.BAD_REQUEST.value(), msg, data);
    }
    public static Result fail(int code, String msg, Object data) {
        Result m = new Result();
        m.setCode(code);
        m.setData(data);
        m.setMsg(msg);
        return m;
    }
}
```

Springboot整合Hibernate Validatior，实体校验

pom文件:

配置文件:

解决跨域问题

pom文件:

配置文件: