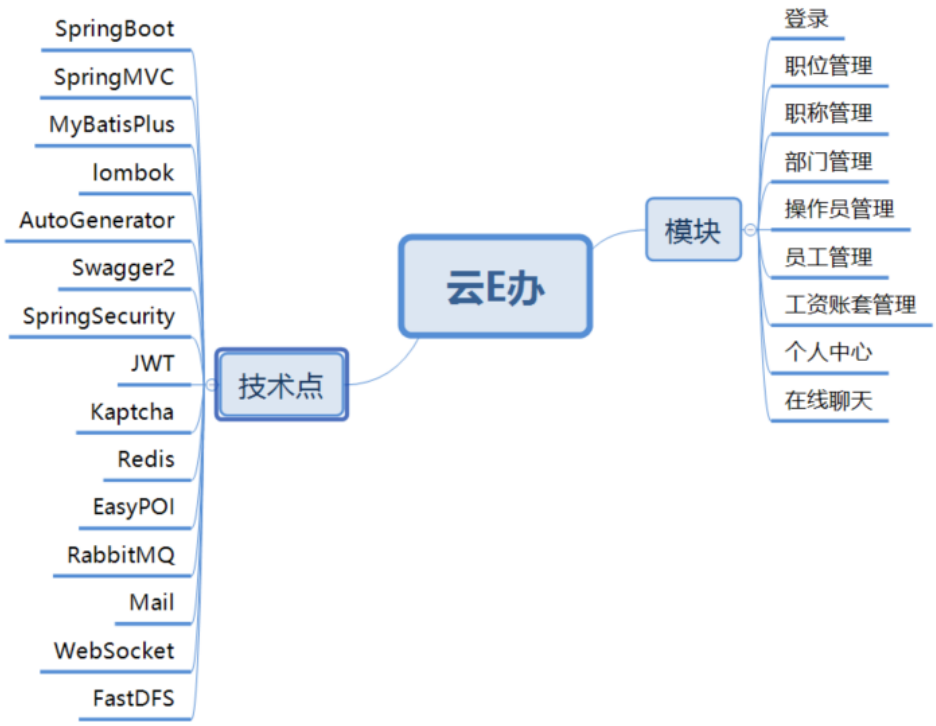
云E办(后端)

1. 项目介绍

本项目目的是实现中小型企业的在线办公系统，云E办在线办公系统是一个用来管理日常的办公事务的一个系统，他能够管的内容有：日常的各种流程审批，新闻，通知，公告，文件信息，财务，人事，费用，资产，行政，项目，移动办公等等。它的作用就是通过软件的方式，方便管理，更加简单，更加扁平。更加高效，更加规范，能够提高整体的管理运营水平。

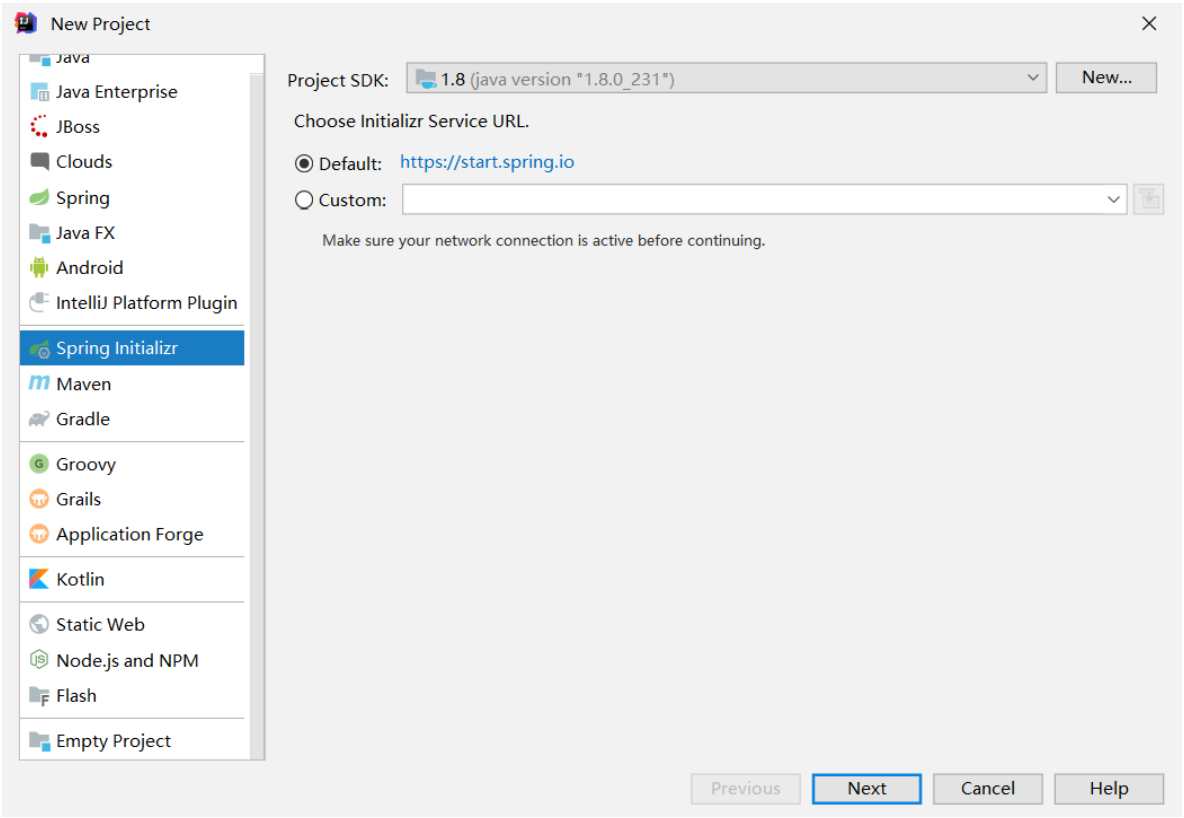
本项目在技术方面采用最主流的前后端分离开发模式，使用业界最流行、社区非常活跃的开源框架Spring Boot来构建后端，旨在实现云E办在线办公系统。包括职位管理、职称管理、部门管理、员工管理、工资管理、在线聊天等模块。项目中还会使用业界主流的第三方组件扩展大家的知识面和技能池。

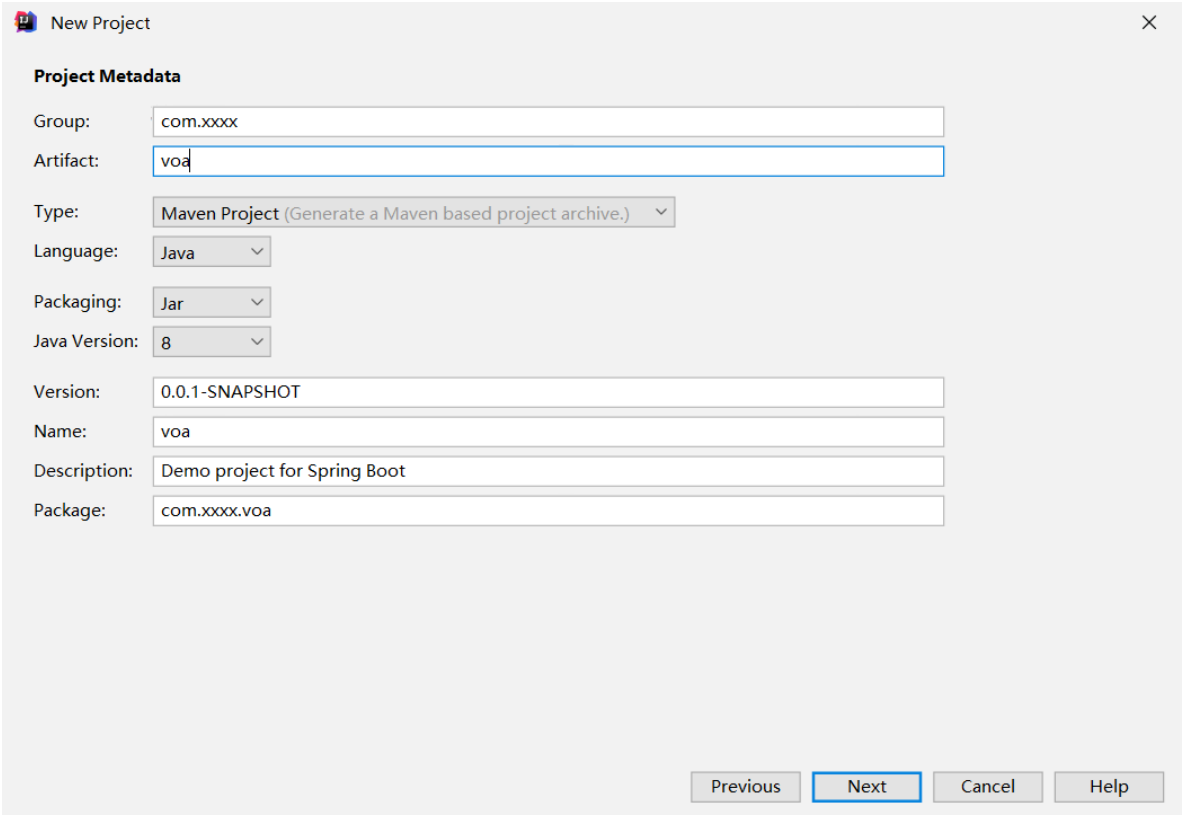
本项目主要模块及技术点如图

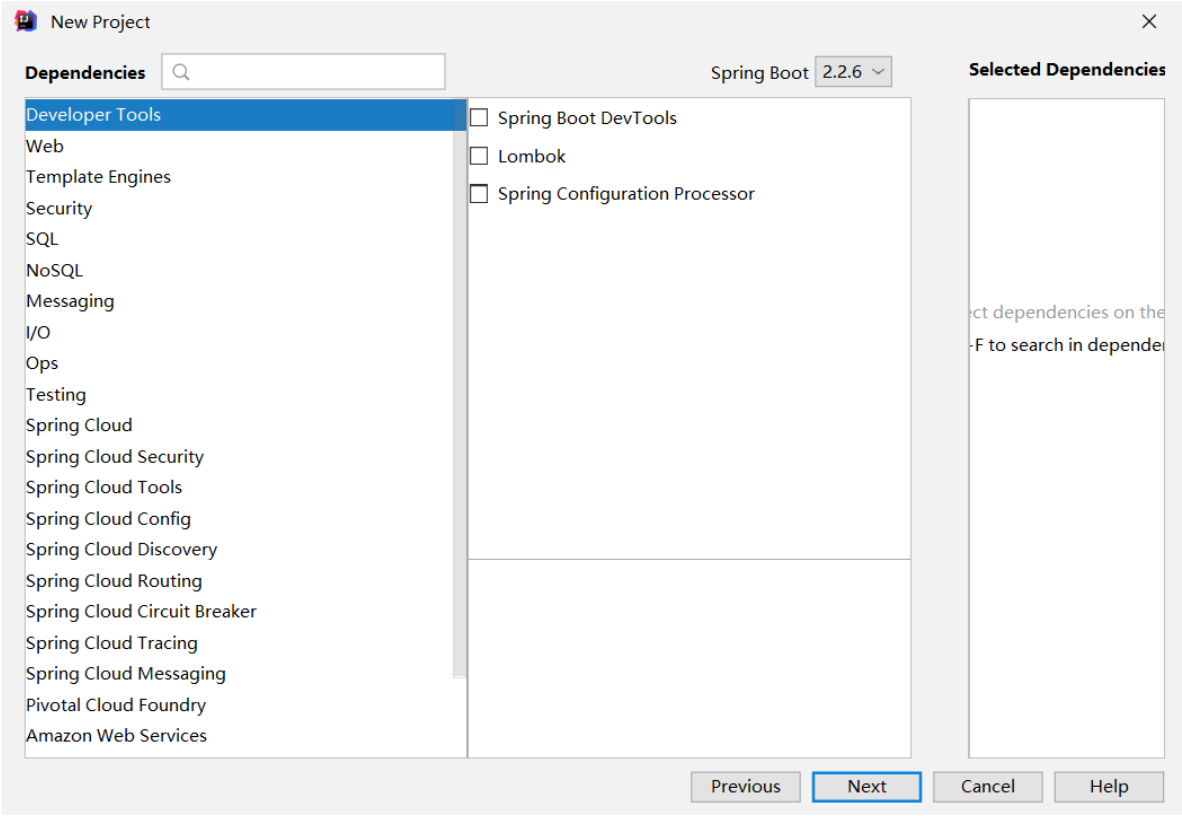


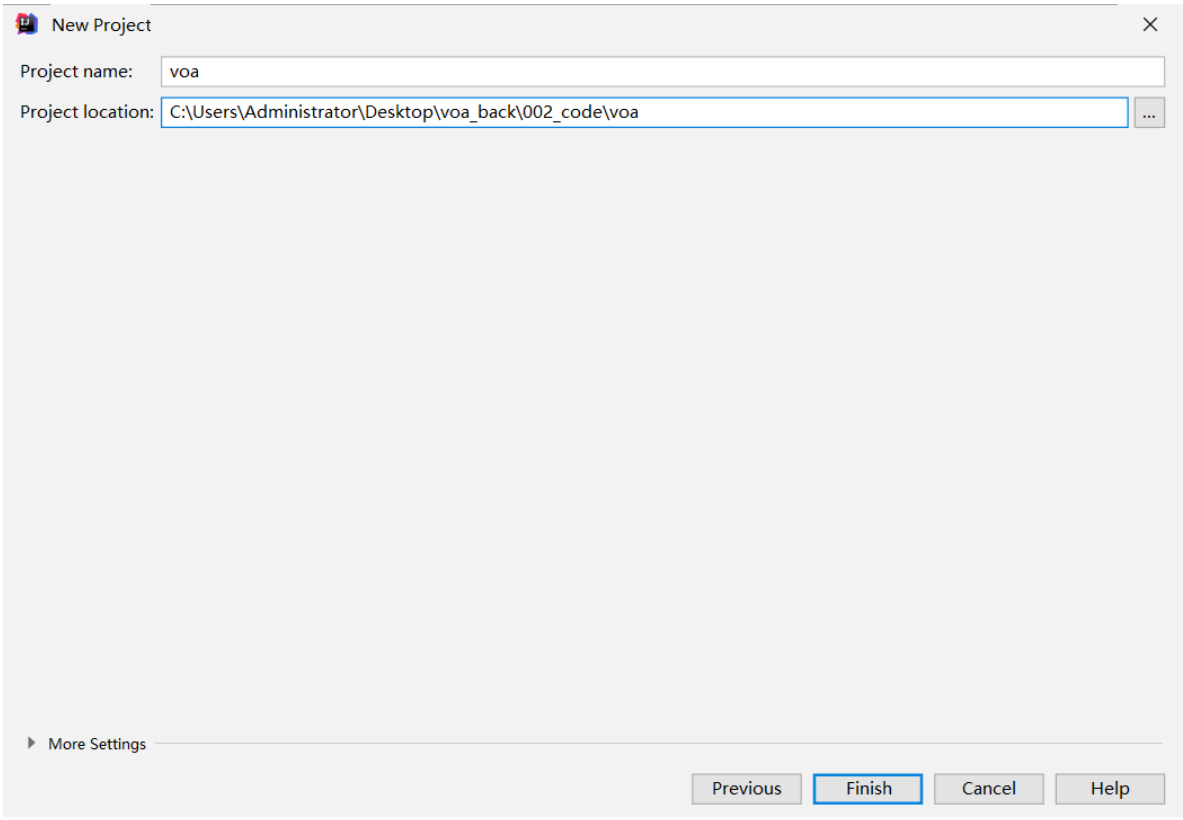
2. 搭建项目

2.1. 创建父项目


New Project
Java Enterprise
JBoss
Clouds
Spring
Java FX
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle
Groovy
Grails
Application Forge
Kotlin
Static Web
Node.js and NPM
Flash
Empty Project
Project SDK:
1.8 (java version "1.8.0_231")
New...
Choose Initializr Service URL.
Default:
https://start.spring.io
Custom:
Make sure your network connection is active before continuing.
Previous
Next
Cancel
Help


New Project
Project Metadata
Group:
com.xxxx
Artifact:
voa
Type:
Maven Project (Generate a Maven based project archive.)
Language:
Java
Packaging:
Jar
Java Version:
8
Version:
0.0.1-SNAPSHOT
Name:
voa
Description:
Demo project for Spring Boot
Package:
com.xxxx.voa
Previous
Next
Cancel
Help





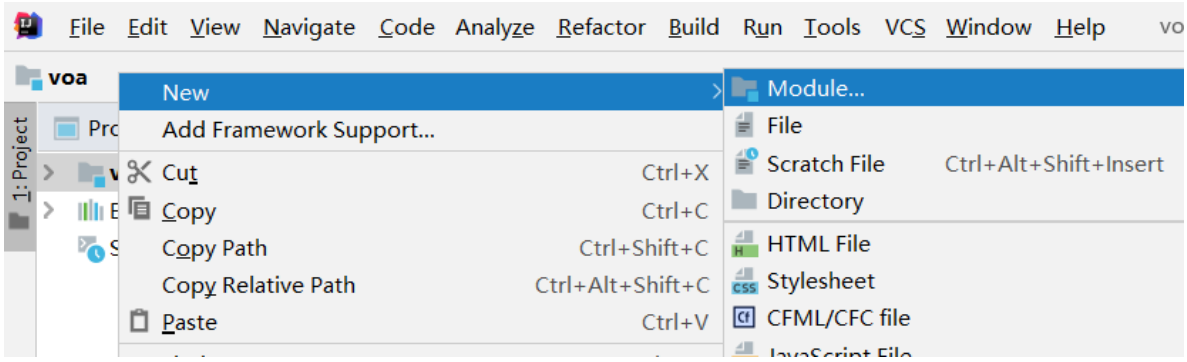
添加依赖

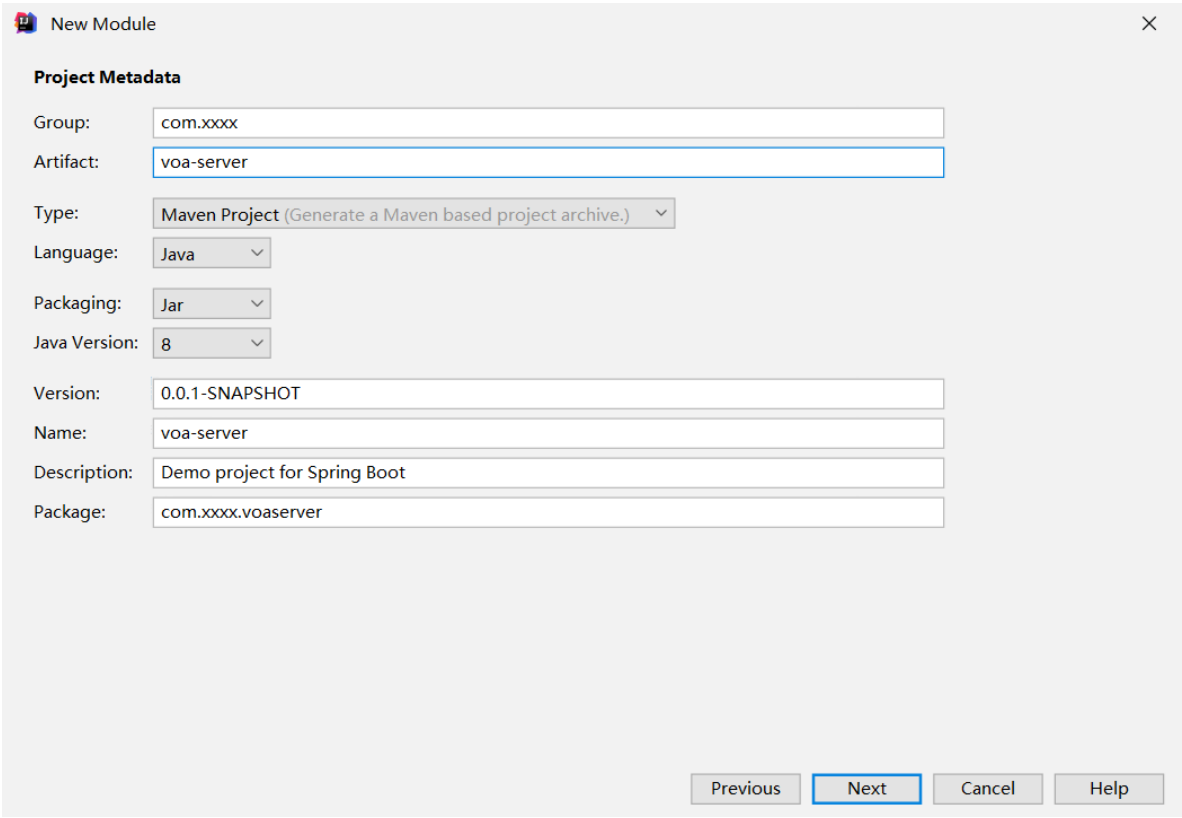
yeb的pom.xml

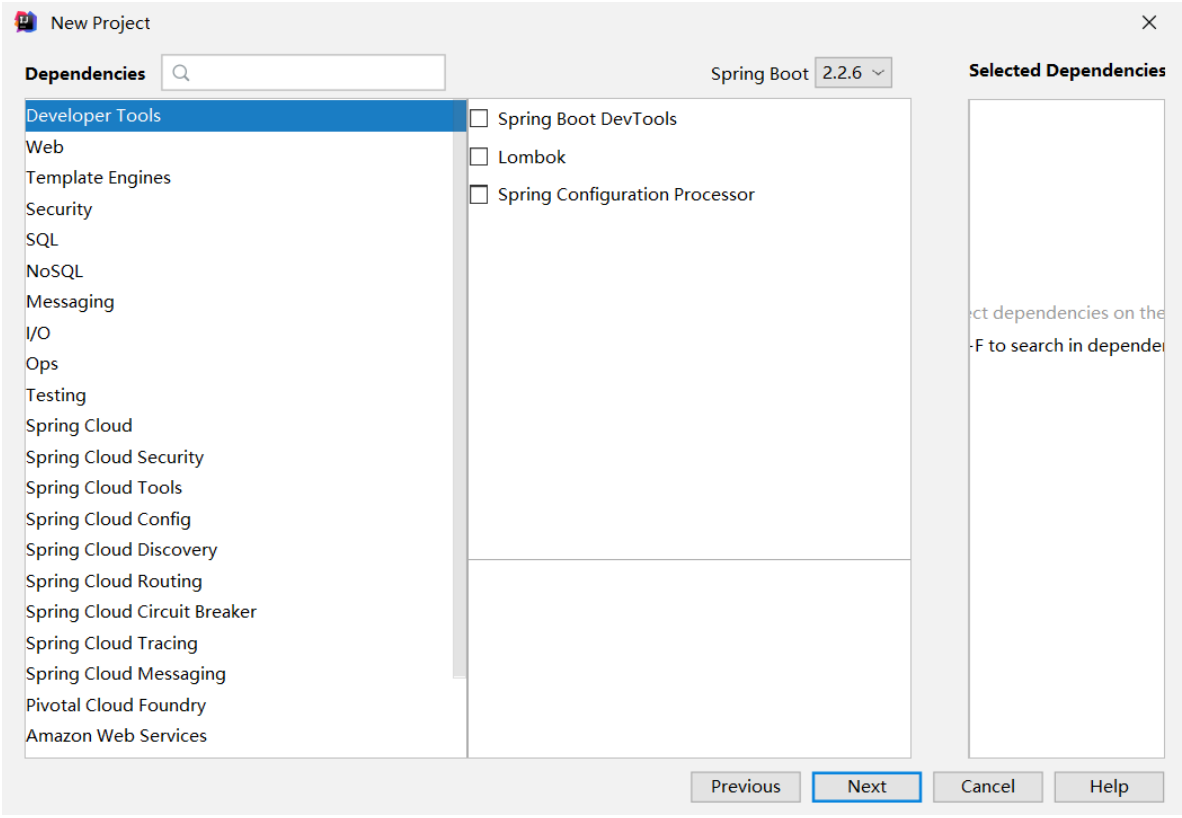
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

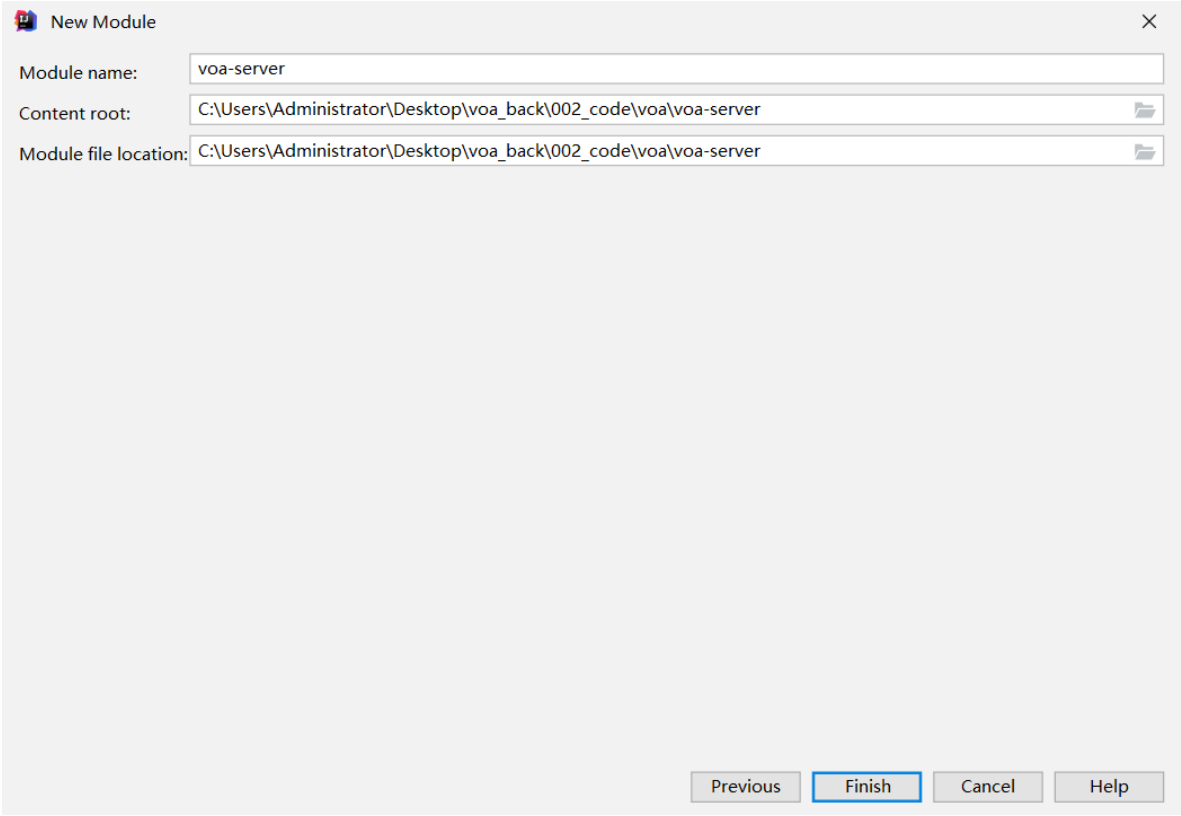
```
    <modules>
        <module>yeb-server</module>
        <module>yeb-generator</module>
    </modules>


    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.0.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>com.xxxx</groupId>
    <artifactId>yeb</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>yeb</name>
    <description>Demo project for Spring Boot</description>

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

2.2. 创建yeb-server子项目




New Module
Project Metadata
Group: com.xxxx
Artifact: voa-server
Type: Maven Project (Generate a Maven based project archive.)
Language: Java
Packaging: Jar
Java Version: 8
Version: 0.0.1-SNAPSHOT
Name: voa-server
Description: Demo project for Spring Boot
Package: com.xxxx.voaserver
Previous
Next
Cancel
Help


New Project
Dependencies
Spring Boot 2.2.6
Selected Dependencies
Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Testing
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Tracing
Spring Cloud Messaging
Pivotal Cloud Foundry
Amazon Web Services
Spring Boot DevTools
Lombok
Spring Configuration Processor
Previous
Next
Cancel
Help



2.3. 添加依赖

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.xxxx</groupId>
        <artifactId>yeb</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <groupId>com.xxxx</groupId>
    <artifactId>yeb-server</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>yeb-server</name>

    <dependencies>
        <!--web 依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--lombok 依赖-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
```

```
        <!--mysql 依赖-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!--mybatis-plus 依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.3.1.tmp</version>
        </dependency>
        <!-- swagger2 依赖 -->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.7.0</version>
        </dependency>
        <!-- Swagger第三方ui依赖 -->
        <dependency>
            <groupId>com.github.xiaoymin</groupId>
            <artifactId>swagger-bootstrap-ui</artifactId>
            <version>1.9.6</version>
        </dependency>
    </dependencies>
</project>
```

2.4. 修改配置文件

application.yml

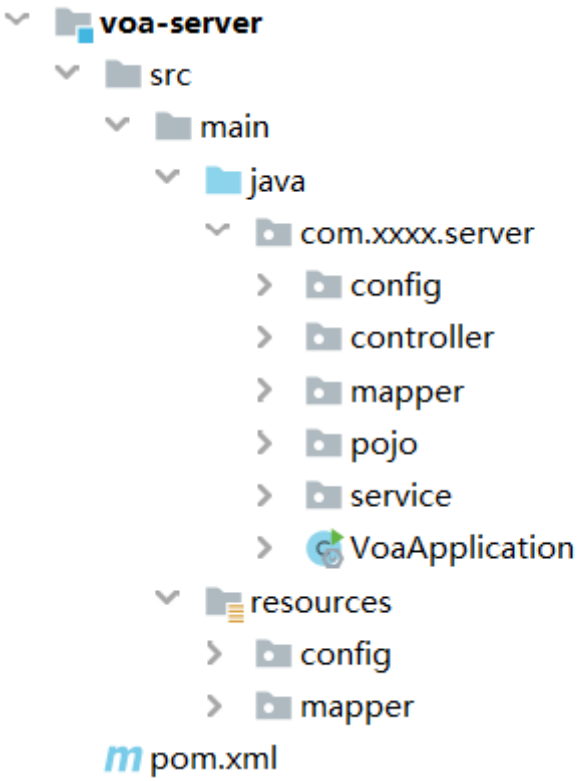
```
server:
  # 端口
  port: 8081

spring:
  # 数据源配置
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/yeb?useUnicode=true&characterEncoding=UTF-
8&serverTimezone=Asia/Shanghai
    username: root
    password: root
    hikari:
      # 连接池名
      pool-name: DateHikariCP
      # 最小空闲连接数
      minimum-idle: 5
      # 空闲连接存活最大时间，默认600000（10分钟）
      idle-timeout: 180000
      # 最大连接数，默认10
      maximum-pool-size: 10
      # 从连接池返回的连接的自动提交
      auto-commit: true
      # 连接最大存活时间，0表示永久存活，默认1800000（30分钟）
      max-lifetime: 1800000
      # 连接超时时间，默认30000（30秒）
```

```
      connection-timeout: 30000
      # 测试连接是否可用的查询语句
      connection-test-query: SELECT 1

# Mybatis-plus配置
mybatis-plus:
  #配置Mapper映射文件
  mapper-locations: classpath*:/mapper/*Mapper.xml
  # 配置MyBatis数据返回类型别名（默认别名是类名）
  type-aliases-package: com.xxxx.server.pojo
  configuration:
    # 自动驼峰命名
    map-underscore-to-camel-case: false

## Mybatis SQL 打印(方法接口所在的包，不是Mapper.xml所在的包)
logging:
  level:
    com.xxxx.server.mapper: debug
```

2.5. 启动类

```
package com.xxxx.server;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * 启动类
 * @author zhoubin
 * @since 1.0.0
 */
@SpringBootApplication
@MapperScan("com.xxxx.server.mapper")
public class VoaApplication {

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

}
```

2.6. 完整目录



3. AutoGenerator的使用

3.1. AutoGenerator是什么?

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Pojo、Mapper、Mapper XML、Service、Controller 等各个模块的代码

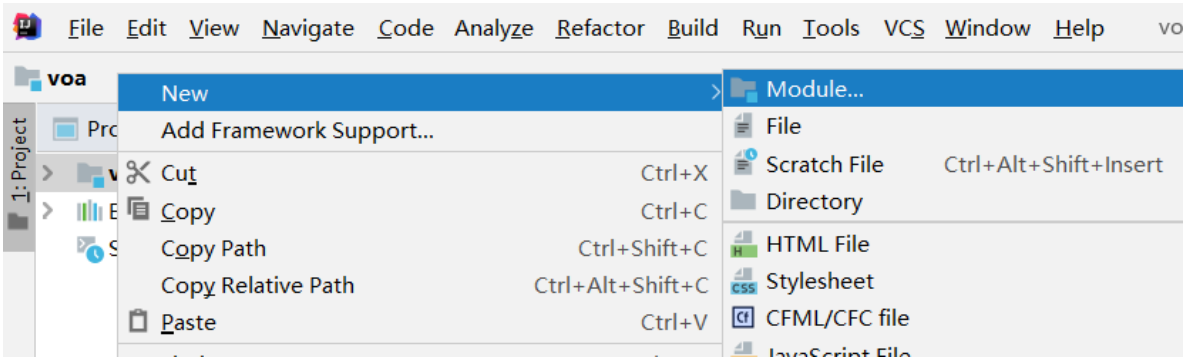
3.2. AutoGenerator能干什么?

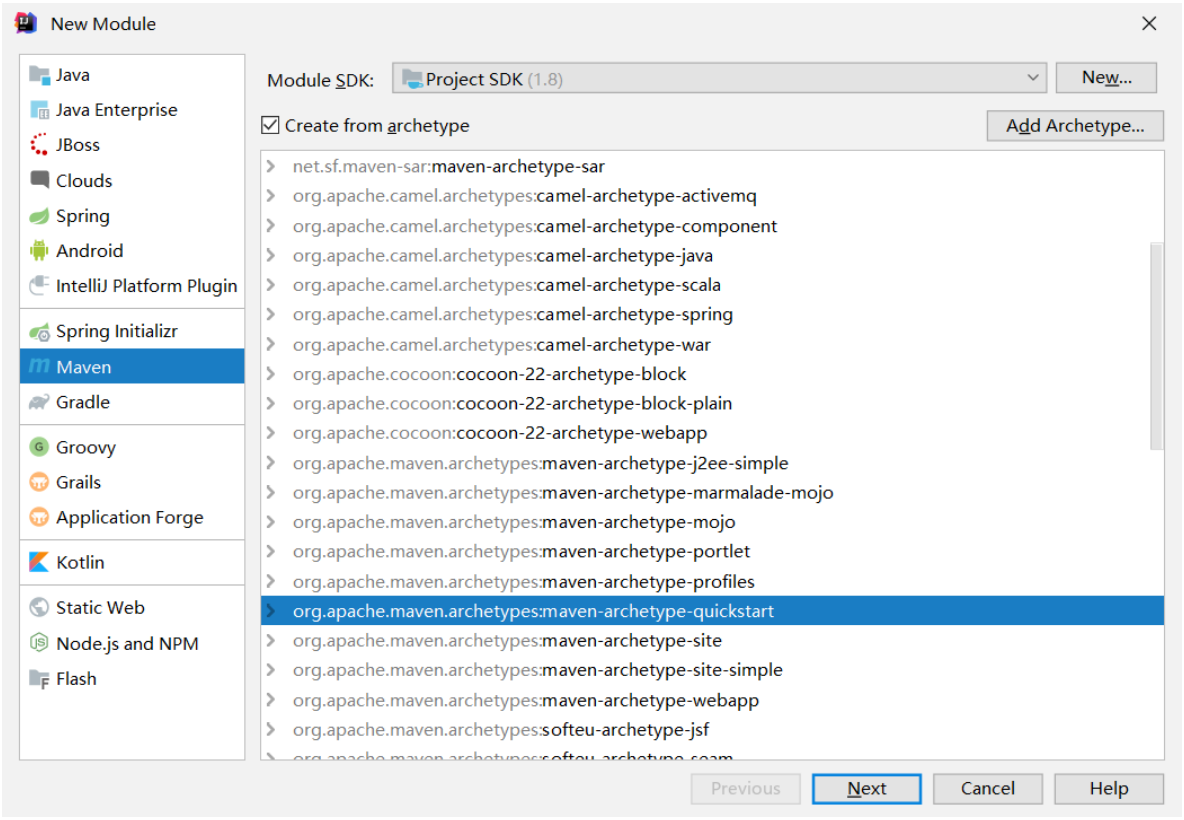
对于单表而言，几乎是一个全能的工具，极大的提升了开发效率。更多的关注业务逻辑的实现。

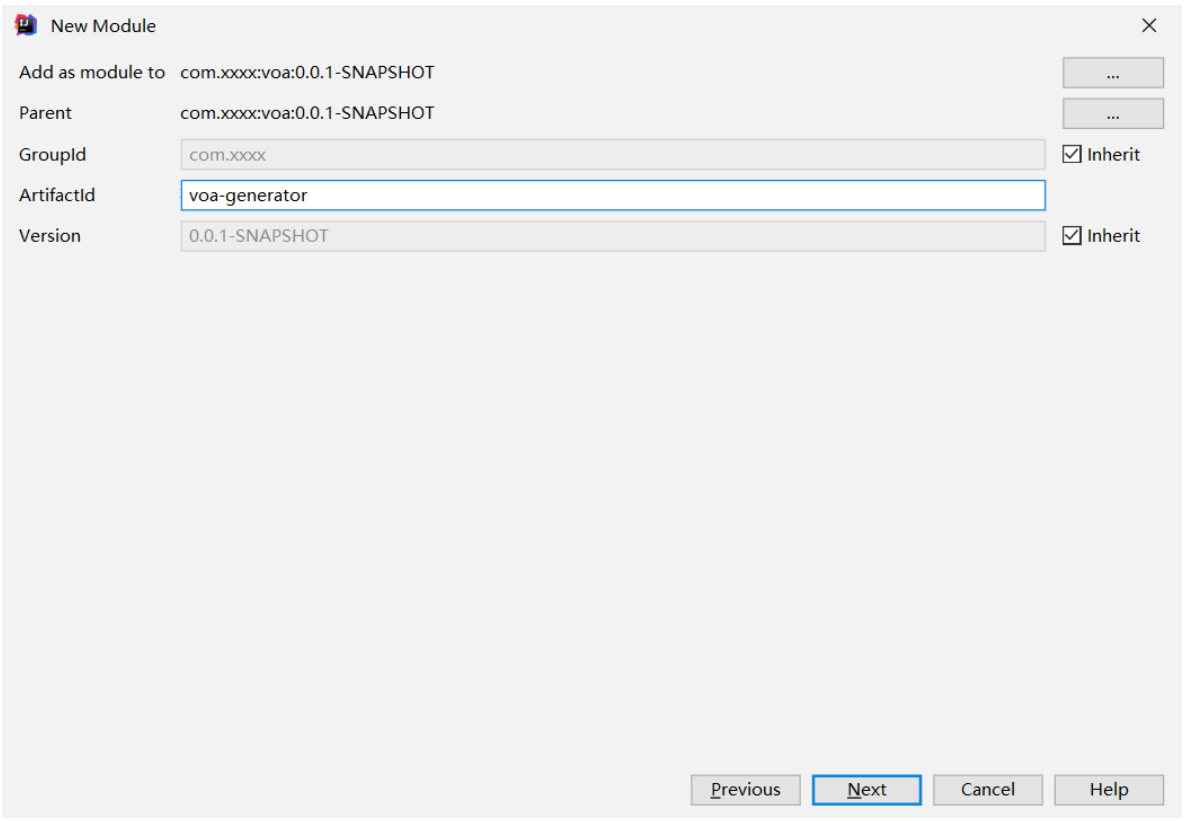
3.3. 怎么使用?

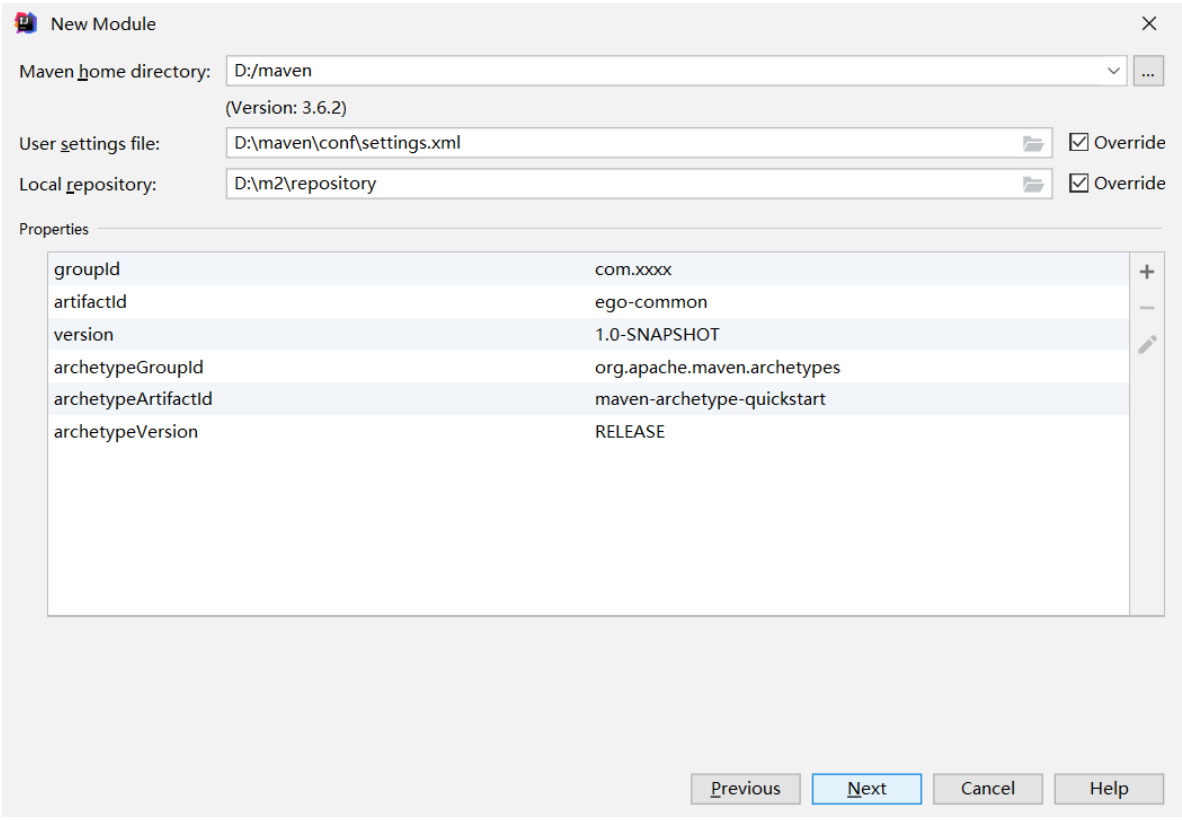
3.3.1. 创建一个AutoGenerator项目

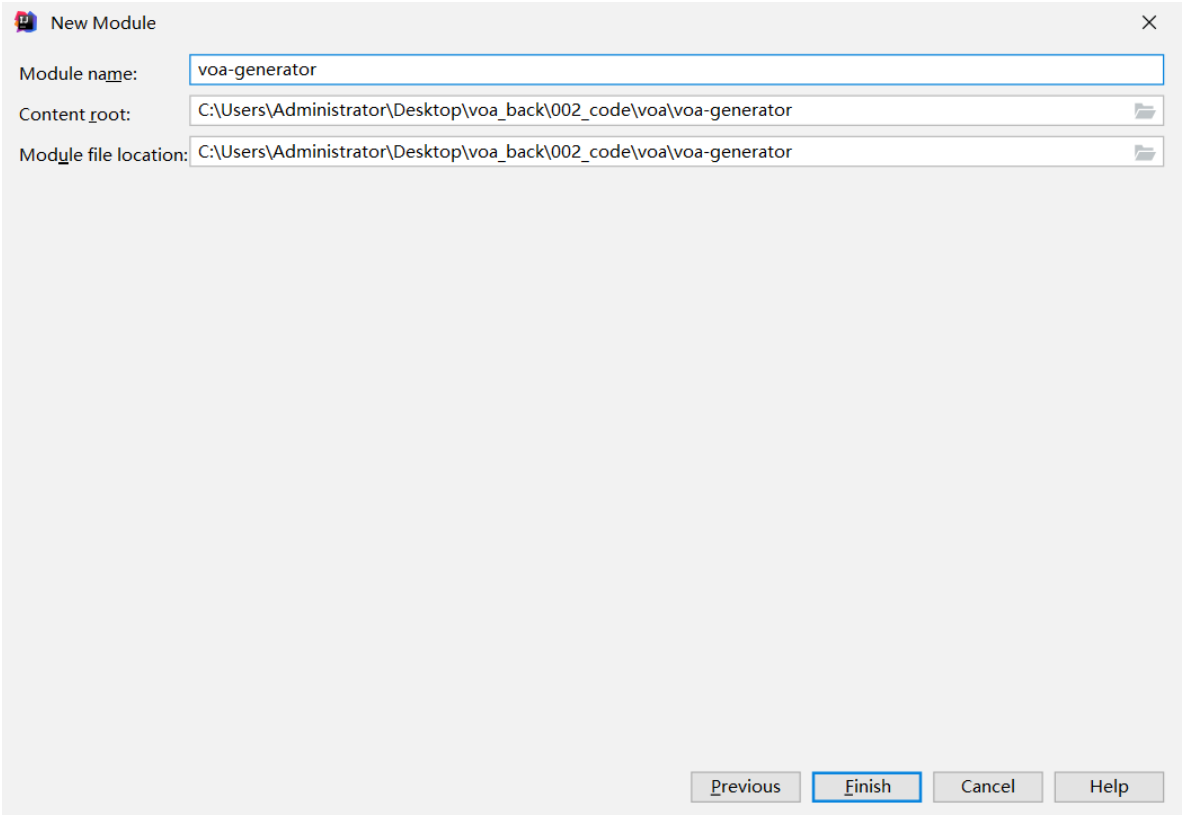
AutoGenerator本身和我们项目没有关联，所以可以单独新建为一个Project，这边也做成Maven聚合项目里的一个子项目




New Module
Java
Java Enterprise
JBoss
Clouds
Spring
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle
Groovy
Grails
Application Forge
Kotlin
Static Web
Node.js and NPM
Flash
Module SDK:
Project SDK (1.8)
New...
Create from archetype
Add Archetype...
net.sf.maven-sar:maven-archetype-sar
org.apache.camel.archetypes:camel-archetype-activemq
org.apache.camel.archetypes:camel-archetype-component
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
org.apache.camel.archetypes:camel-archetype-spring
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
org.apache.maven.archetypes:softeu-archetype-jsf
Previous
Next
Cancel
Help


New Module
Add as module to
com.xxxx:voa:0.0.1-SNAPSHOT
...
Parent
com.xxxx:voa:0.0.1-SNAPSHOT
...
GroupId
com.xxxx
Inherit
ArtifactId
voa-generator
Version
0.0.1-SNAPSHOT
Inherit
Previous
Next
Cancel
Help





3.3.2. 添加依赖

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <parent>
        <groupId>com.xxxx</groupId>
        <artifactId>yeb</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <groupId>com.xxxx</groupId>
    <artifactId>yeb-generator</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!--web 依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--mybatis-plus 依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.3.1.tmp</version>
        </dependency>
        <!--mybatis-plus 代码生成器依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.3.1.tmp</version>
        </dependency>
        <!--freemarker 依赖-->
        <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
        </dependency>
        <!--mysql 依赖-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

</project>
```

3.3.3. CodeGenerator工具类

```
package com.xxxx.generator;

import com.baomidou.mybatisplus.core.exceptions.MybatisPlusException;
import com.baomidou.mybatisplus.core.toolkit.StringPool;
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
```

```java
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.InjectionConfig;
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
import com.baomidou.mybatisplus.generator.config.FileOutConfig;
import com.baomidou.mybatisplus.generator.config.GlobalConfig;
import com.baomidou.mybatisplus.generator.config.PackageConfig;
import com.baomidou.mybatisplus.generator.config.StrategyConfig;
import com.baomidou.mybatisplus.generator.config.TemplateConfig;
import com.baomidou.mybatisplus.generator.config.po.TableInfo;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * 执行 main 方法控制台输入模块表名回车自动生成对应项目目录中
 *
 * @author zhoubin
 * @since 1.0.0
 */

public class CodeGenerator {

    /**
     * <p>
     * 读取控制台内容
     * </p>
     */
    public static String scanner(String tip) {
        Scanner scanner = new Scanner(System.in);
        StringBuilder help = new StringBuilder();
        help.append("请输入" + tip + "：");
        System.out.println(help.toString());
        if (scanner.hasNext()) {
            String ipt = scanner.next();
            if (StringUtils.isNotEmpty(ipt)) {
                return ipt;
            }
        }
        tadminow new MybatisPlusException("请输入正确的" + tip + "！");
    }

    public static void main(String[] args) {
        // 代码生成器
        AutoGenerator mpg = new AutoGenerator();

        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir");
        gc.setOutputDir(projectPath + "/yeb-generator/src/main/java");
        //作者
        gc.setAuthor("zhoubin");
        //打开输出目录
        gc.setOpen(false);
        //xml开启 BaseResultMap
        gc.setBaseResultMap(true);
```

```
        //xml 开启BaseColumnList
        gc.setBaseColumnList(true);
        // 实体属性 Swagger2 注解
        gc.setSwagger2(true);
        mpg.setGlobalConfig(gc);

        // 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://localhost:3306/yeb?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia" +
                "/Shanghai");
        dsc.setDriverName("com.mysql.cj.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("root");
        mpg.setDataSource(dsc);

        // 包配置
        PackageConfig pc = new PackageConfig();
        pc.setParent("com.xxxx")
                .setEntity("pojo")
                .setMapper("mapper")
                .setService("service")
                .setServiceImpl("service.impl")
                .setController("controller");
        mpg.setPackageInfo(pc);

        // 自定义配置
        InjectionConfig cfg = new InjectionConfig() {
            @Override
            public void initMap() {
                // to do nothing
            }
        };

        // 如果模板引擎是 freemarker
        String templatePath = "/templates/mapper.xml.ftl";
        // 如果模板引擎是 velocity
        // String templatePath = "/templates/mapper.xml.vm";

        // 自定义输出配置
        List<FileOutConfig> focList = new ArrayList<>();
        // 自定义配置会被优先输出
        focList.add(new FileOutConfig(templatePath) {
            @Override
            public String outputFile(TableInfo tableInfo) {
                // 自定义输出文件名 ， 如果你 Entity 设置了前后缀、此处注意 xml 的名称会跟着发生
变化！！
                return projectPath + "/yeb-generator/src/main/resources/mapper/" +
tableInfo.getEntityName() + "Mapper"
                        + StringPool.DOT_XML;
            }
        });
        cfg.setFileOutConfigList(focList);
        mpg.setCfg(cfg);

        // 配置模板
        TemplateConfig templateConfig = new TemplateConfig();
```

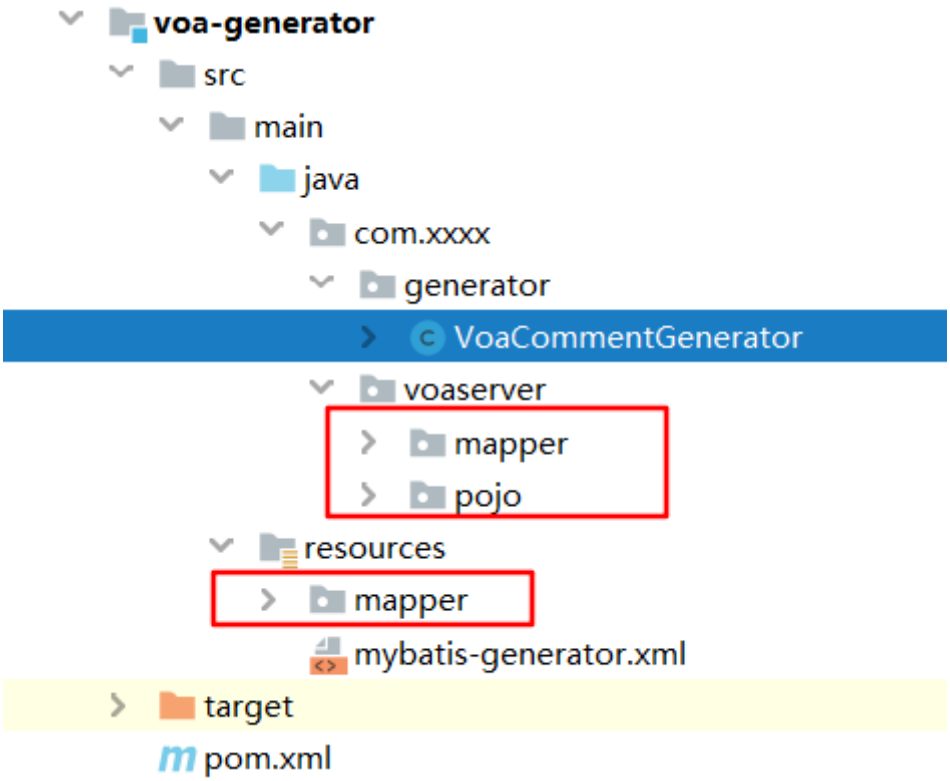
```
        templateConfig.setXml(null);
        mpg.setTemplate(templateConfig);

        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        //数据库表映射到实体的命名策略
        strategy.setNaming(NamingStrategy.underline_to_camel);
        //数据库表字段映射到实体的命名策略
        strategy.setColumnNaming(NamingStrategy.no_change);
        //lombok模型
        strategy.setEntityLombokModel(true);
        //生成 @RestController 控制器
        strategy.setRestControllerStyle(true);
        strategy.setInclude(scanner("表名，多个英文逗号分割").split(","));
        strategy.setControllerMappingHyphenStyle(true);
        //表前缀
        strategy.setTablePrefix("t_");
        mpg.setStrategy(strategy);
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        mpg.execute();
    }

}
```

3.3.4. 执行

执行main方法，在控制台直接输出表名，多个表名用`,`隔开

```
D:\java\jdk\bin\java.exe ...
请输入表名，多个英文逗号分割：
t_admin,t_salary
```

3.3.5. 结果



4. 登录功能

我们这边使用`Spring Security`框架实现登录功能,关于`Spring Security`知识点请参考之前文档

4.1. 添加依赖

pom.xml

```
<!--security 依赖-->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!--JWT 依赖-->
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt</artifactId>
   <version>0.9.0</version>
</dependency>
```

4.2. 修改配置

application.yml

```yaml
jwt:
  # JWT存储的请求头
  tokenHeader: Authorization
  # JWT 加解密使用的密钥
  secret: yeb-secret
  # JWT的超期限时间（60*60*24）
  expiration: 604800
  # JWT 负载中拿到开头
  tokenHead: Bearer
```

4.3. 添加Jwt Token的工具类

JwtTokenUtil.java

```java
package com.xxxx.yeb.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

/**
 * Jwt Token工具类
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Component
public class JwtTokenUtil {

    private static final String CLAIM_KEY_USERNAME = "sub";
    private static final String CLAIM_KEY_CREATED = "created";
    @Value("${jwt.secret}")
    private String secret;
    @Value("${jwt.expiration}")
    private Long expiration;

    /**
     * 根据负载生成JWT Token
     *
     * @param claims
     * @return
     */
    private String generateToken(Map<String, Object> claims) {
        return Jwts.builder()
                .setClaims(claims)
                .setExpiration(generateExpirationDate())
                .signWith(SignatureAlgorithm.ES512, secret)
                .compact();
    }
```

```
/**
 * 从token中获取JWT中的负载
 *
 * @param token
 * @return
 */
private Claims getClaimsFromToken(String token) {
   Claims claims = null;
   try {
      claims = Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody();
   } catch (Exception e) {
      e.printStackTrace();
   }
   return claims;
}

/**
 * 生成token过期时间
 *
 * @return
 */
private Date generateExpirationDate() {
   return new Date(System.currentTimeMillis() + expiration * 1000);
}

/**
 * 从token中获取过期时间
 *
 * @param token
 * @return
 */
private Date getExpiredDateFromToken(String token) {
   Claims claims = getClaimsFromToken(token);
   return claims.getExpiration();
}

/**
 * 判断token是否失效
 *
 * @param token
 * @return
 */
private boolean isTokenExpired(String token) {
   Date expiredDate = getExpiredDateFromToken(token);
   return expiredDate.before(new Date());
}

/**
 * 从token中获取登录用户名
 *
 * @param token
 * @return
 */
public String getUserNameFormToken(String token) {
```

```
        String username;
        try {
            Claims claims = getClaimsFromToken(token);
            username = claims.getSubject();
        } catch (Exception e) {
            username = null;
        }
        return username;
    }

    /**
     * 验证token是否有效
     *
     * @param token
     * @param userDetails
     * @return
     */
    public boolean validateToken(String token, UserDetails userDetails) {
        String username = getUserNameFormToken(token);
        return username.equals(userDetails.getUsername()) && !isTokenExpired(token);
    }

    /**
     * 根据用户信息生成token
     * @param userDetails
     * @return
     */
    public String generateToken(UserDetails userDetails){
        Map<String,Object> claims = new HashMap<>();
        claims.put(CLAIM_KEY_USERNAME,userDetails.getUsername());
        claims.put(CLAIM_KEY_CREATED,new Date());
        return generateToken(claims);
    }

    /**
     * 判断token是否可以被刷新
     * @param token
     * @return
     */
    public boolean canRefresh(String token){
        return !isTokenExpired(token);
    }

    /**
     * 刷新token
     * @param token
     * @return
     */
    public String refreshToken(String token){
        Claims claims = getClaimsFromToken(token);
        claims.put(CLAIM_KEY_CREATED,new Date());
        return generateToken(claims);
    }


}
```

4.4. Admin实现UserDetails类

```
package com.xxxx.pojo;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.io.Serializable;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

/**
 * <p>
 *
 * </p>
 *
 * @author zhoubin
 */
@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
@TableName("t_admin")
@ApiModel(value = "Admin对象", description = "")
public class Admin implements Serializable, UserDetails {

    @ApiModelProperty(value = "id")
    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;

    @ApiModelProperty(value = "姓名")
    private String name;

    @ApiModelProperty(value = "手机号码")
    private String phone;

    @ApiModelProperty(value = "住宅电话")
    private String telephone;

    @ApiModelProperty(value = "联系地址")
    private String address;

    @ApiModelProperty(value = "是否启用")
    private Boolean enabled;

    @ApiModelProperty(value = "用户名")
    private String username;
```

```
    @ApiModelProperty(value = "密码")
    private String password;

    @ApiModelProperty(value = "用户头像")
    private String userFace;

    @ApiModelProperty(value = "备注")
    private String remark;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }
}
```

4.5. 添加公共返回对象

RespBean.java

```
package com.xxxx.server.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 通用返回结果对象
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class RespBean {
```

```
    private long code;
    private String message;
    private Object obj;

    /**
     * 成功返回结果
     *
     * @param message
     */
    public static RespBean success(String message) {
        return new RespBean(200, message, null);
    }

    /**
     * 成功返回结果
     *
     * @param obj
     * @param message
     */
    public static RespBean success(String message, Object obj) {
        return new RespBean(200, message, obj);
    }

    /**
     * 失败返回结果
     *
     * @param message
     */
    public static RespBean error(String message) {
        return new RespBean(500, message, null);
    }

    /**
     * 失败返回结果
     * @param message
     * @param obj
     * @return
     */
    public static RespBean error(String message,Object obj) {
        return new RespBean(500, message, obj);
    }
}
```

4.6. 添加登录相应接口

4.6.1. AdminLoginParam

```
package com.xxxx.server.pojo;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;

@Data
@EqualsAndHashCode(callSuper = false)
```

```
@Accessors(chain = true)
@ApiModel(value = "AdminLogin对象", description = "")
public class AdminLoginParam {
    @ApiModelProperty(value = "用户名", required = true)
    private String username;
    @ApiModelProperty(value = "密码", required = true)
    private String password;
}
```

4.6.2. LoginController

```
package com.xxxx.server.controller;

import com.xxxx.server.pojo.Admin;
import com.xxxx.server.pojo.AdminLoginParam;
import com.xxxx.server.pojo.RespBean;
import com.xxxx.server.service.IAdminService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import java.security.Principal;

/**
 * 登录控制器
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Api(tags = "LoginController")
@RestController
public class LoginController {

    @Autowired
    private IAdminService adminService;


    @ApiOperation(value = "登录之后返回token")
    @PostMapping("/login")
    public RespBean login(@RequestBody AdminLoginParam adminLoginParam,
HttpServletRequest request) {
        return adminService.login(adminLoginParam.getUsername(),
adminLoginParam.getPassword(), request);
    }


    @ApiOperation(value = "获取当前用户信息")
    @GetMapping("/admin/info")
    public Admin getAdminInfo(Principal principal) {
        if (null == principal) {
            return null;
```

```
        }
        String username = principal.getName();
        Admin admin = adminService.getAdminByUserName(username);
        admin.setPassword(null);
        return admin;
    }


    @ApiOperation(value = "退出登录")
    @PostMapping("/logout")
    public RespBean logout() {
        return RespBean.success("注销成功！");
    }

}
```

4.6.3. IAdminService

```
package com.xxxx.server.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.xxxx.server.pojo.Admin;
import com.xxxx.server.pojo.Role;

import java.util.List;

/**
 * <p>
 *  服务类
 * </p>
 *
 * @author zhoubin
 */
public interface IAdminService extends IService<Admin> {

    /**
    * 登录返回token
    * @param username
    * @param password
    * @return
    */
   RespBean login(String username,String password);

   /**
    * 根据用户名获取用户
    * @param username
    */
   Admin getAdminByUserName(String username);
}
```

4.6.4. AdminServiceImpl

```
package com.xxxx.server.service.impl;

import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
```

```
import com.xxxx.server.config.security.JwtTokenUtil;
import com.xxxx.server.mapper.AdminMapper;
import com.xxxx.server.mapper.RoleMapper;
import com.xxxx.server.pojo.Admin;
import com.xxxx.server.pojo.RespBean;
import com.xxxx.server.pojo.Role;
import com.xxxx.server.service.IAdminService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import javax.servlet.http.HttpServletRequest;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author zhoubin
 */
@Service
public class AdminServiceImpl extends ServiceImpl<AdminMapper, Admin> implements
IAdminService {
    @Autowired
    private AdminMapper adminMapper;
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private PasswordEncoder passwordEncoder;
    @Autowired
    private JwtTokenUtil jwtTokenUtil;
    @Value("${jwt.tokenHead}")
    private String tokenHead;

    /**
     * 登录返回token
     *
     * @param username
     * @param password
     * @return
     */
    @Override
    public RespBean login(String username, String password) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        if (null == userDetails || !passwordEncoder.matches(password,
userDetails.getPassword())) {
            return RespBean.error("用户名或密码不正确!");
        }
        if (!userDetails.isEnabled()){
```

```
            return RespBean.error("账号被禁用，请联系管理员!");
        }
        UsernamePasswordAuthenticationToken authentication =
                new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String token = jwtTokenUtil.generateToken(userDetails);
        Map<String, String> tokenMap = new HashMap<>();
        tokenMap.put("token", token);
        tokenMap.put("tokenHead", tokenHead);
        return RespBean.success("登录成功", tokenMap);
    }

    /**
     * 根据用户名获取用户
     *
     * @param username
     * @return
     */
    @Override
    public Admin getAdminByUserName(String username) {
        return adminMapper.selectOne(new QueryWrapper<Admin>().eq("username",
username));
    }
}
```

4.7. 配置SpringSecurity

SecurityConfig.java

```
package com.xxxx.server.config.security;

import com.xxxx.server.config.security.component.JwtAuthenticationTokenFilter;
import com.xxxx.server.config.security.component.RestAuthenticationEntryPoint;
import com.xxxx.server.config.security.component.RestfulAccessDeniedHandler;
import com.xxxx.server.pojo.Admin;
import com.xxxx.server.service.IAdminService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationM
anagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurer
Adapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
```

```
/**
 * Security配置类
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private IAdminService adminService;
    @Autowired
    private RestAuthenticationEntryPoint restAuthenticationEntryPoint;
    @Autowired
    private RestfulAccessDeniedHandler restfulAccessDeniedHandler;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) tadminows Exception {

auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) tadminows Exception {
        //使用JWT，不需要csrf
        http.csrf()
                .disable()
                //基于token，不需要session
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                //允许登录访问
                .antMatchers("/login","/logout")
                .permitAll()
                //除上面外，所有请求都要求认证
                .anyRequest()
                .authenticated()
                .and()
                //禁用缓存
                .headers()
                .cacheControl();
        //添加jwt 登录授权过滤器
        http.addFilterBefore(jwtAuthenticationTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
        //添加自定义未授权和未登录结果返回
        http.exceptionHandling()
                .accessDeniedHandler(restfulAccessDeniedHandler)
                .authenticationEntryPoint(restAuthenticationEntryPoint);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    @Bean
```

```
    public UserDetailsService userDetailsService() {
        //获取登录用户信息
        return username -> {
            Admin admin = adminService.getAdminByUserName(username);
            if (null != admin) {
                return admin;
            }
            return null;
        };
    }

    @Bean
    public JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter() {
        return new JwtAuthenticationTokenFilter();
    }
}
```

4.8. 添加自定义未授权及未登录的结果返回

RestAuthorizationEntryPoint.java

```
package com.xxxx.server.config.security.component;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.xxxx.server.pojo.RespBean;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 当未登录或者token失效时访问接口时，自定义的返回结果
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Component
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {

   @Override
   public void commence(HttpServletRequest request, HttpServletResponse response,
AuthenticationException authException) tadminows IOException, ServletException {
      response.setCharacterEncoding("UTF-8");
      response.setContentType("application/json");
      PrintWriter out = response.getWriter();
      RespBean bean = RespBean.error("权限不足，请联系管理员！");
      bean.setCode(401);
      out.write(new ObjectMapper().writeValueAsString(bean));
      out.flush();
      out.close();
   }
}
```

RestfulAccessDeniedHandler.java

```
package com.xxxx.server.config.security.component;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.xxxx.server.pojo.RespBean;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 当访问接口没有权限时，自定义返回结果类
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Component
public class RestfulAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
AccessDeniedException e) tadminows IOException, ServletException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        RespBean bean = RespBean.error("权限不足，请联系管理员！");
        bean.setCode(403);
        out.write(new ObjectMapper().writeValueAsString(bean));
        out.flush();
        out.close();
    }
}
```

4.9. 添加Jwt登录授权过滤器

JwtAuthenticationTokenFilter.java

```
package com.xxxx.server.config.security.component;

import com.xxxx.server.config.security.JwtTokenUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * Jwt登录授权过滤器
 *
 * @author zhoubin
 * @since 1.0.0
 */
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private JwtTokenUtil jwtTokenUtil;
    @Value("${jwt.tokenHeader}")
    private String tokenHeader;
    @Value("${jwt.tokenHead}")
    private String tokenHead;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain chain) tadminows ServletException, IOException {
        String authHeader = request.getHeader(this.tokenHeader);
        //存在token
        if (null != authHeader && authHeader.startsWith(this.tokenHead)) {
            String authToken = authHeader.substring(this.tokenHead.length());
            String username = jwtTokenUtil.getUserNameFormToken(authToken);
            //token中存在用户名但未登录
            if (null!=username&&
null==SecurityContextHolder.getContext().getAuthentication()){
                //登录
                UserDetails userDetails =
this.userDetailsService.loadUserByUsername(username);
                //验证token是否有效，重新设置用户对象
                if (jwtTokenUtil.validateToken(authToken,userDetails)){
                    UsernamePasswordAuthenticationToken authentication =
                            new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                    authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);
                }
            }
        }
        chain.doFilter(request,response);
    }
}
```

5. 接口文档

接口文档以及测试使用`swagger`实现，关于`swagger`知识点请参考之前文档

5.1. 添加依赖

```
<!-- swagger2 依赖 -->
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger2</artifactId>
   <version>2.7.0</version>
</dependency>
<!-- Swagger第三方ui依赖 -->
<dependency>
   <groupId>com.github.xiaoymin</groupId>
   <artifactId>swagger-bootstrap-ui</artifactId>
   <version>1.9.6</version>
</dependency>
```

5.2. 配置Swagger

Swagger2Config.java

```
package com.xxxx.server.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.ApiKey;
import springfox.documentation.service.AuthorizationScope;
import springfox.documentation.service.Contact;
import springfox.documentation.service.SecurityReference;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;
import java.util.List;

/**
 * Swagger2配置
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Configuration
@EnableSwagger2
public class Swagger2Config {
    @Bean
    public Docket createRestApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())
                .select()
                //为当前包下的controller生成api文档

.apis(RequestHandlerSelectors.basePackage("com.xxxx.server.controller"))
                .paths(PathSelectors.any())
```

```
                .build()
    }

    private ApiInfo apiInfo() {
        //设置文档信息
        return new ApiInfoBuilder()
                .title("云E办接口文档")
                .description("云E办接口文档")
                .contact(new Contact("xxxx", "http:localhost:8081/doc.html",
"xxxx@xxxx.com"))
                .version("1.0")
                .build();
    }
}
```

5.3. 准备测试接口

HelloController.java

```
package com.xxxx.server.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * 测试Controller
 *
 * @author zhoubin
 * @since 1.0.0
 */
@RestController
public class HelloController {

   @GetMapping("/hello")
   public String hello(){
      return "hello";
   }

}
```

5.4. 测试

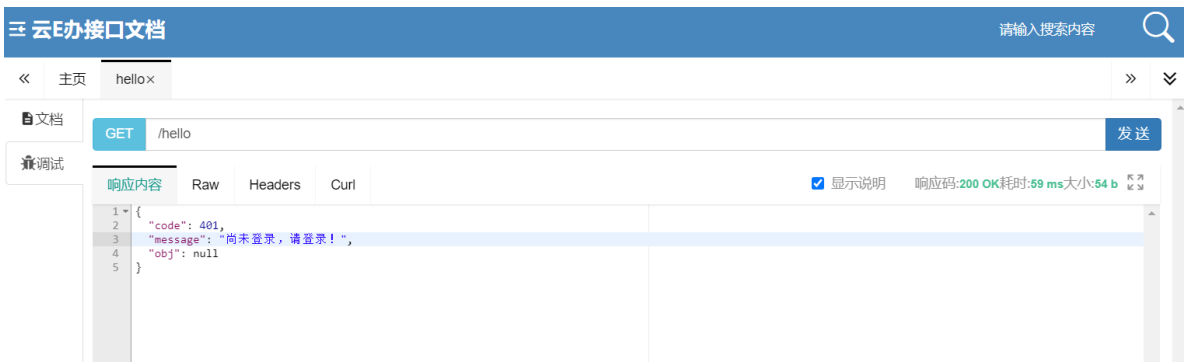
访问http:localhost:8081/doc.html

发现无法访问，这是因为Swagger的地址被SpringSecurity拦截。我们修改下SpringSecurity的配置，放行Swagger

SecurityConfig.java

```
@Override
public void configure(WebSecurity web) tadminows Exception {
   //放行静态资源
   web.ignoring().antMatchers(
         "/login",
         "/logout",
         "/css/**",
         "/js/**",
         "/index.html",
         "/favicon.ico",
         "/doc.html",
         "/webjars/**",
         "/swagger-resources/**",
         "/v2/api-docs/**");
}
```

重启项目再次访问 `/hello` 接口



发现接口访问失败，提示**暂未登录或token已经过期**。这是因为 `/hello` 接口需要登录成功之后才能访问

5.5. 添加Authorize

Swagger2Config.java

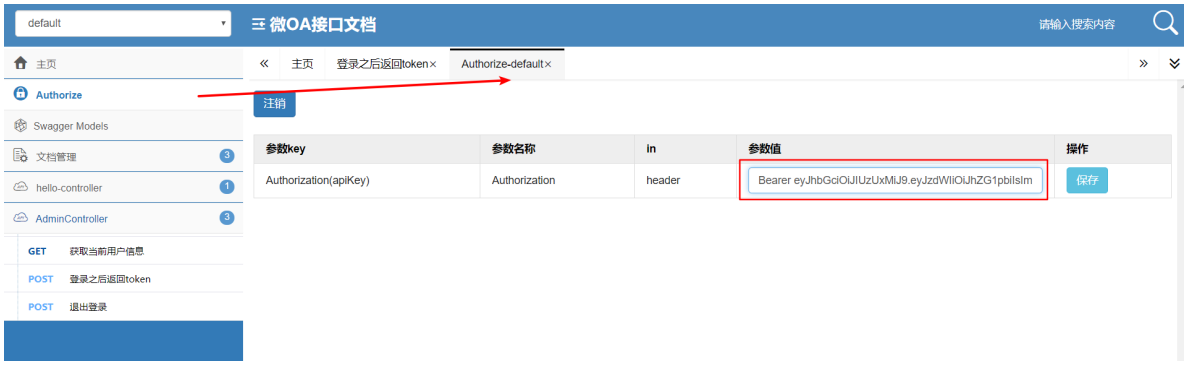
```java
package com.xxxx.server.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.ApiKey;
import springfox.documentation.service.AuthorizationScope;
import springfox.documentation.service.Contact;
import springfox.documentation.service.SecurityReference;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;
import java.util.List;

/**
 * Swagger2配置
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Configuration
@EnableSwagger2
public class Swagger2Config {
   @Bean
   public Docket createRestApi() {
      return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(apiInfo())
            .select()
            //为当前包下的controller生成api文档
            .apis(RequestHandlerSelectors.basePackage("com.xxxx.server.controller"))
            .paths(PathSelectors.any())
            .build()
            //添加登录认证
            .securitySchemes(securitySchemes())
            .securityContexts(securityContexts());
   }

   private List<SecurityContext> securityContexts() {
      //设置需要登录认证的路径
      List<SecurityContext> result = new ArrayList<>();
      result.add(getContextByPath("/hello/.*"));
      return result;
   }

   private SecurityContext getContextByPath(String pathRegex) {
      return SecurityContext.builder()
            .securityReferences(defaultAuth())
            .forPaths(PathSelectors.regex(pathRegex))
            .build();
   }
```

```
    private List<SecurityReference> defaultAuth() {
        List<SecurityReference> result = new ArrayList<>();
        AuthorizationScope authorizationScope = new AuthorizationScope("global",
"accessEverything");
        AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
        authorizationScopes[0] = authorizationScope;
        result.add(new SecurityReference("Authorization",authorizationScopes));
        return result;
    }

    private ApiInfo apiInfo() {
        //设置文档信息
        return new ApiInfoBuilder()
                .title("云E办接口文档")
                .description("云E办接口文档")
                .contact(new Contact("zhoubin", "http:localhost:8081/doc.html",
"xxxx@xxxx.com"))
                .version("1.0")
                .build();
    }

    private List<ApiKey> securitySchemes(){
        //设置请求头信息
        List<ApiKey> result = new ArrayList<>();
        ApiKey apiKey = new ApiKey("Authorization","Authorization","header");
        result.add(apiKey);
        return result;
    }


}
```

访问http:localhost:8081/doc.html

首先登录获取token

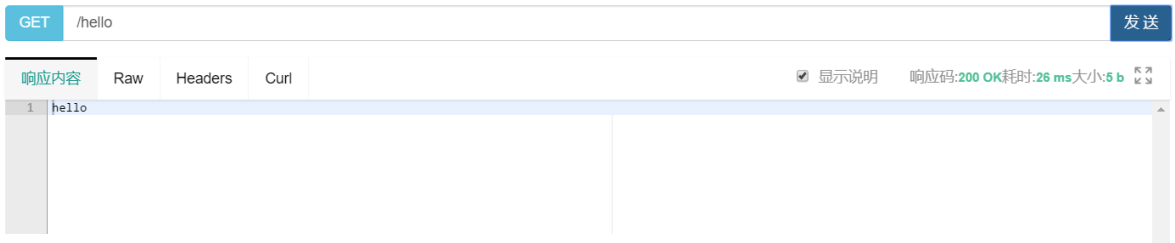
点击 `Authorize` 按钮，在参数值输入框中输入获取的token，并保存



注意： `Bearer` 必须输入，并且和token中间有一个空格。

可以通过 `/admin/info` 接口获取当前登录用户信息

再次访问 `/hello` 接口，也可以正常访问



6. 登录验证码

6.1. 生成验证码

图像验证码显示功能使用 `google Kaptcha` 验证码产品 实现前台验证码显示功能

Home » com.github.axet » kaptcha » 0.0.9

Kaptcha » 0.0.9

A web kaptcha generation engine

License	Apache 2.0
HomePage	https://github.com/axet/kaptcha
Date	(Nov 25, 2013)
Files	pom (3 KB) jar (96 KB) View All
Repositories	Central Sonatype Spring Plugins
Used By	17 artifacts

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!— https://mvnrepository.com/artifact/com.github.axet/kaptcha —>
<dependency>
    <groupId>com.github.axet</groupId>
    <artifactId>kaptcha</artifactId>
    <version>0.0.9</version>
</dependency>
```

6.1.1. 添加依赖

```xml
<!-- google kaptcha依赖 -->
<dependency>
   <groupId>com.github.axet</groupId>
   <artifactId>kaptcha</artifactId>
   <version>0.0.9</version>
</dependency>
```

6.1.2. 验证码配置类

CaptchaConfig.java

```java
package com.xxxx.yebserver.config;

import com.google.code.kaptcha.impl.DefaultKaptcha;
import com.google.code.kaptcha.util.Config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Properties;

/**
 * 验证码配置类
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Configuration
public class CaptchaConfig {

   @Bean
   public DefaultKaptcha getDefaultKaptcha(){
      //验证码生成器
      DefaultKaptcha defaultKaptcha=new DefaultKaptcha();
      //配置
      Properties properties = new Properties();
      //是否有边框
```

```
        properties.setProperty("kaptcha.border", "yes");
        //设置边框颜色
        properties.setProperty("kaptcha.border.color", "105,179,90");
        //边框粗细度，默认为1
        // properties.setProperty("kaptcha.border.thickness","1");
        //验证码
        properties.setProperty("kaptcha.session.key","code");
        //验证码文本字符颜色 默认为黑色
        properties.setProperty("kaptcha.textproducer.font.color", "blue");
        //设置字体样式
        properties.setProperty("kaptcha.textproducer.font.names", "宋体,楷体,微软雅黑");
        //字体大小，默认40
        properties.setProperty("kaptcha.textproducer.font.size", "30");
        //验证码文本字符内容范围 默认为abced2345678gfynmnpwx
        // properties.setProperty("kaptcha.textproducer.char.string", "");
        //字符长度，默认为5
        properties.setProperty("kaptcha.textproducer.char.length", "4");
        //字符间距 默认为2
        properties.setProperty("kaptcha.textproducer.char.space", "4");
        //验证码图片宽度 默认为200
        properties.setProperty("kaptcha.image.width", "100");
        //验证码图片高度 默认为40
        properties.setProperty("kaptcha.image.height", "40");
        Config config = new Config(properties);
        defaultKaptcha.setConfig(config);
        return defaultKaptcha;
    }
}
```

6.1.3. 提供验证码生成接口

CaptchaController.java

```
package com.xxxx.server.controller;

import com.google.code.kaptcha.impl.DefaultKaptcha;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.imageio.ImageIO;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.awt.image.BufferedImage;
import java.io.IOException;

/**
 * 验证码Controller
 *
 * @author zhoubin
 * @since 1.0.0
 */
@RestController
public class CaptchaController {

    @Autowired
```

```
    private DefaultKaptcha defaultKaptcha;

    @ApiOperation(value = "验证码")
    @GetMapping(value = "/captcha", produces = "image/jpeg")
    public void captcha(HttpServletRequest request, HttpServletResponse response) {
        // 定义response输出类型为image/jpeg类型
        response.setDateHeader("Expires", 0);
        // Set standard HTTP/1.1 no-cache headers.
        response.setHeader("Cache-Control", "no-store, no-cache, must-revalidate");
        // Set IE extended HTTP/1.1 no-cache headers (use addHeader).
        response.addHeader("Cache-Control", "post-check=0, pre-check=0");
        // Set standard HTTP/1.0 no-cache header.
        response.setHeader("Pragma", "no-cache");
        // return a jpeg
        response.setContentType("image/jpeg");

        //--------------------生成验证码 begin -------------------------
        //获取验证码文本内容
        String text = defaultKaptcha.createText();
        System.out.println("验证码内容：" + text);
        //将验证码放入session中
        request.getSession().setAttribute("captcha", text);
        //根据文本内容创建图形验证码
        BufferedImage image = defaultKaptcha.createImage(text);
        ServletOutputStream outputStream = null;
        try {
            outputStream = response.getOutputStream();
            //输出流输出图片，格式jpg
            ImageIO.write(image, "jpg", outputStream);
            outputStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (null != outputStream) {
                try {
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        //--------------------生成验证码 end ----------------------------
    }

}
```

- `produces`：设置返回数据类型及编码

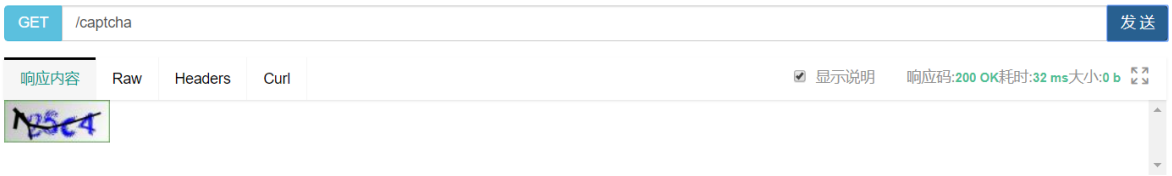
6.1.4. 放行验证码接口

SecurityConfig.java

```
@Override
public void configure(WebSecurity web) tadminows Exception {
   //放行静态资源
   web.ignoring().antMatchers(
         "/login",
         "/logout",
```

```
            "/css/**",
            "/js/**",
            "/index.html",
            "/img/**",
            "/fonts/**",
            "/favicon.ico",
            "/doc.html",
            "/webjars/**",
            "/swagger-resources/**",
            "/v2/api-docs/**",
            "/captcha");
    }
```

6.1.5. 测试



6.2. 校验验证码

6.2.1. 登录参数对象添加验证码属性

AdminLoginParam.java

```
package com.xxxx.server.pojo;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;

@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
@ApiModel(value = "AdminLogin对象", description = "")
public class AdminLoginParam {
    @ApiModelProperty(value = "用户名", required = true)
    private String username;
    @ApiModelProperty(value = "密码", required = true)
    private String password;
    @ApiModelProperty(value = "验证码", required = true)
    private String Code;
}
```

6.2.2. 登录方法添加验证码判断

LoginController.java

```
@ApiOperation(value = "登录之后返回token")
    @PostMapping("/login")
    public RespBean login(@RequestBody AdminLoginParam adminLoginParam,
HttpServletRequest request) {
        return adminService.login(adminLoginParam.getUsername(),
adminLoginParam.getPassword(),
                adminLoginParam.getCode(), request);
    }
```

IAdminService.java

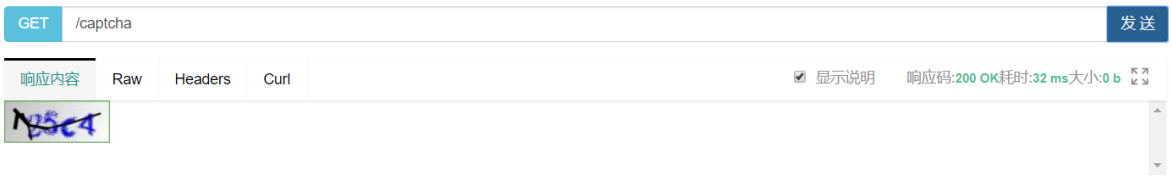
```
/**
 * 登录返回token
 * @param username
 * @param password
 * @param code
 * @param request
 * @return
 */
RespBean login(String username, String password, String code, HttpServletRequest
request);
```

AdminServiceImpl.java

```
/**
 * 登录返回token
 *
 * @param username
 * @param password
 * @param code
 * @param request
 * @return
 */
@Override
public RespBean login(String username, String password, String code,
HttpServletRequest request) {
   String captcha = (String) request.getSession().getAttribute("captcha");
   if (StringUtils.isBlank(code) || !captcha.equals(code)) {
      return RespBean.error("验证码填写错误！");
   }
   UserDetails userDetails = userDetailsService.loadUserByUsername(username);
   if (null == userDetails || !passwordEncoder.matches(password,
userDetails.getPassword())) {
       return RespBean.error("用户名或密码不正确!");
   }
   if (!userDetails.isEnabled()){
       return RespBean.error("账号被禁用，请联系管理员!");
   }
   UsernamePasswordAuthenticationToken authentication =
         new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
   SecurityContextHolder.getContext().setAuthentication(authentication);
   String token = jwtTokenUtil.generateToken(userDetails);
   Map<String, String> tokenMap = new HashMap<>();
   tokenMap.put("token", token);
   tokenMap.put("tokenHead", tokenHead);
   return RespBean.success("登录成功", tokenMap);
}
```

6.2.3. 测试

首先获取验证码



输入错误的验证码，提示验证码填写错误

输入正确验证码，登录成功

7. 菜单功能

7.1. 修改菜单类

菜单分两级，一级菜单下面有子菜单，我们修改下Menu对象

Menu.java

```
package com.xxxx.server.pojo;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;

import java.io.Serializable;
import java.util.List;

/**
 * <p>
 *
 * </p>
 *
 * @author zhoubin
 */
@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
@TableName("t_menu")
@ApiModel(value="Menu对象", description="")
public class Menu implements Serializable {


    @ApiModelProperty(value = "id")
    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;

    @ApiModelProperty(value = "url")
    private String url;

    @ApiModelProperty(value = "path")
    private String path;

    @ApiModelProperty(value = "组件")
    private String component;

    @ApiModelProperty(value = "菜单名")
    private String name;

    @ApiModelProperty(value = "图标")
    private String iconCls;

    @ApiModelProperty(value = "是否保持激活")
    private Boolean keepAlive;

    @ApiModelProperty(value = "是否要求权限")
    private Boolean requireAuth;
```

```
    @ApiModelProperty(value = "父id")
    private Integer parentId;

    @ApiModelProperty(value = "是否启用")
    private Boolean enabled;

    @ApiModelProperty(value = "子菜单")
    TableField(exist = false)
    private List<Menu> children;
}
```

7.2. MenuMapper

MenuMapper.java

```
package com.xxxx.server.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.xxxx.server.pojo.Menu;

import java.util.List;

/**
 * <p>
 *  Mapper 接口
 * </p>
 *
 * @author zhoubin
 */
public interface MenuMapper extends BaseMapper<Menu> {

    /**
     * 通过用户id获取菜单列表
     * @param id
     * @return
     */
    List<Menu> getMenusByAdminId(Integer id);
}
```

MenuMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.xxxx.server.mapper.MenuMapper">
    <resultMap id="Menus" type="com.xxxx.server.pojo.Menu" extends="BaseResultMap">
        <collection property="children" ofType="com.xxxx.server.pojo.Menu">
            <id column="id2" property="id" />
            <result column="url2" property="url" />
            <result column="path2" property="path" />
            <result column="component2" property="component" />
            <result column="name2" property="name" />
            <result column="iconCls2" property="iconCls" />
            <result column="keepAlive2" property="keepAlive" />
            <result column="requireAuth2" property="requireAuth" />
            <result column="parentId2" property="parentId" />
            <result column="enabled2" property="enabled" />
```

```
        </collection>
    </resultMap>

    <!--通过用户id获取菜单列表-->
    <select id="getMenusByAdminId" resultMap="Menus">
        SELECT
        DISTINCT m1.*,
            m2.id as id2,
            m2.component as component2,
            m2.enabled as enabled2,
            m2.iconCls as iconCls2,
            m2.keepAlive as keepAlive2,
            m2.name as name2,
            m2.parentId as parentId2,
            m2.requireAuth as requireAuth2,
            m2.path as path2
        FROM
            t_menu m1,
            t_menu m2,
            t_admin_role ar,
            t_menu_role mr
        WHERE
            m1.id = m2.parentId
            AND ar.adminId = #{id}
            AND ar.rid = mr.rid
            AND mr.mid = m2.id
            AND m2.enabled = true
        ORDER BY
            m1.id,
            m2.id
    </select>
</mapper>
```

- `collection`：关联关系，是实现一对多的关键
  - `property`：javabean中容器对应字段名
  - `ofType`：指定集合中元素的对象类型
  - `select`：使用另一个查询封装的结果
  - `column`：数据库中的列名，与`select`配合使用

7.3. MenuService

IMenuService.java

```
package com.xxxx.server.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.xxxx.server.pojo.Menu;

import java.util.List;

/**
 * <p>
 *  服务类
 * </p>
 *
 * @author zhoubin
```

```
 */
public interface IMenuService extends IService<Menu> {

    /**
     * 通过用户id获取菜单列表
     * @return
     */
    List<Menu> getMenusByAdminId();
}
```

MenuServiceImpl.java

```
package com.xxxx.server.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.xxxx.server.mapper.MenuMapper;
import com.xxxx.server.pojo.Admin;
import com.xxxx.server.pojo.Menu;
import com.xxxx.server.service.IMenuService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * <p>
 *  服务实现类
 * </p>
 *
 * @author zhoubin
 */
@Service
public class MenuServiceImpl extends ServiceImpl<MenuMapper, Menu> implements
IMenuService {

    @Autowired
    private MenuMapper menuMapper;

    /**
     * 通过用户id获取菜单列表
     * @return
     */
    @Override
    public List<Menu> getMenusByAdminId() {
        return menuMapper.getMenusByAdminId(((Admin)
SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId());
    }
}
```

7.4. 修改接口请求路径

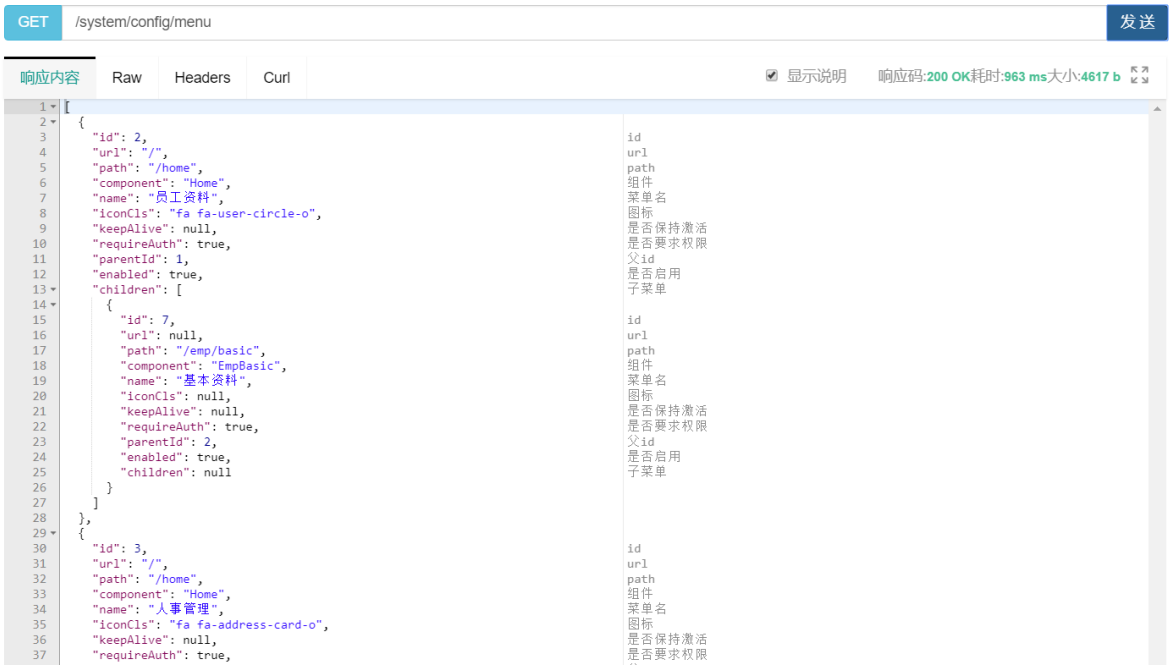
接口请求路径必须符合`menu`表里的定义。

MenuController.java

```
package com.xxxx.yebserver.controller;
```

```java
import com.xxxx.yebserver.pojo.Menu;
import com.xxxx.yebserver.service.MenuService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

/**
 * 系统配置Controller
 *
 * @author zhoubin
 * @since 1.0.0
 */
@RestController
@RequestMapping("/system/config")
public class SystemConfigController {

    @Autowired
    private MenuService menuService;

    @ApiOperation(value = "通过用户id获取菜单列表")
    @GetMapping("/menu")
    public List<Menu> getMenusByHrId(){
        return menuService.getMenusByHrId();
    }

}
```

7.5. 测试



7.6. Redis优化菜单

菜单大部分情况下不会出现变化，我们可以将其放入 `Redis` 加快加载速度，,关于 `Redis` 知识点请参考之前文档

7.6.1. 添加依赖

```
<!-- spring data redis 依赖 -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- commons-pool2 对象池依赖 -->
<dependency>
   <groupId>org.apache.commons</groupId>
   <artifactId>commons-pool2</artifactId>
</dependency>
```

7.7. 修改配置

```
# Redis配置
redis:
  timeout: 10000ms                          # 连接超时时间
  host: 192.168.10.100                     # Redis服务器地址
  port: 6379                                # Redis服务器端口
  database: 0                               # 选择哪个库，默认0库
  lettuce:
    pool:
      max-active: 1024                      # 最大连接数，默认 8
      max-wait: 10000ms                     # 最大连接阻塞等待时间，单位毫秒，默认 -1
      max-idle: 200                         # 最大空闲连接，默认 8
      min-idle: 5                           # 最小空闲连接，默认 0
```

7.8. 配置Redis

RedisConfig.java

```
package com.xxxx.server.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.GenericJackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

/**
 * Redis配置类
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Configuration
public class RedisConfig {
   @Bean
   public RedisTemplate<String,Object> redisTemplate(LettuceConnectionFactory
redisConnectionFactory){
      RedisTemplate<String,Object> redisTemplate = new RedisTemplate<>();
      //为string类型key设置序列器
      redisTemplate.setKeySerializer(new StringRedisSerializer());
      //为string类型value设置序列器
```

```
        redisTemplate.setValueSerializer(new GenericJackson2JsonRedisSerializer());
        //为hash类型key设置序列器
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        //为hash类型value设置序列器
        redisTemplate.setHashValueSerializer(new GenericJackson2JsonRedisSerializer());
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        return redisTemplate;
    }
}
```

7.9. 修改菜单方法

MenuServiceImpl.java

```
/**
 * 通过用户id获取菜单列表
 *
 * @return
 */
@Override
public List<Menu> getMenusByAdminId() {
    Integer adminId = ((Admin)
SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId();
    ValueOperations<String, Object> valueOperations = redisTemplate.opsForValue();
    //查询缓存中是否有数据
    List<Menu> menus = (List<Menu>) valueOperations.get("menu_" + adminId);
    if (CollectionUtils.isEmpty(menus)){
        //如果没数据，数据库中查询，并设置到缓存中
        menus = menuMapper.getMenusByAdminId(adminId);
        valueOperations.set("menu_"+adminId,menus);
    }
    return menus;
}
```

7.10. 测试

第一次查询时，Redis并没有菜单数据

会从数据库中查询菜单数据并设置到Redis中，此时再次查看发现Redis中已经有数据。再次查询会直接查询Redis中数据。

8. 权限管理

8.1. 权限管理RBAC基本概念

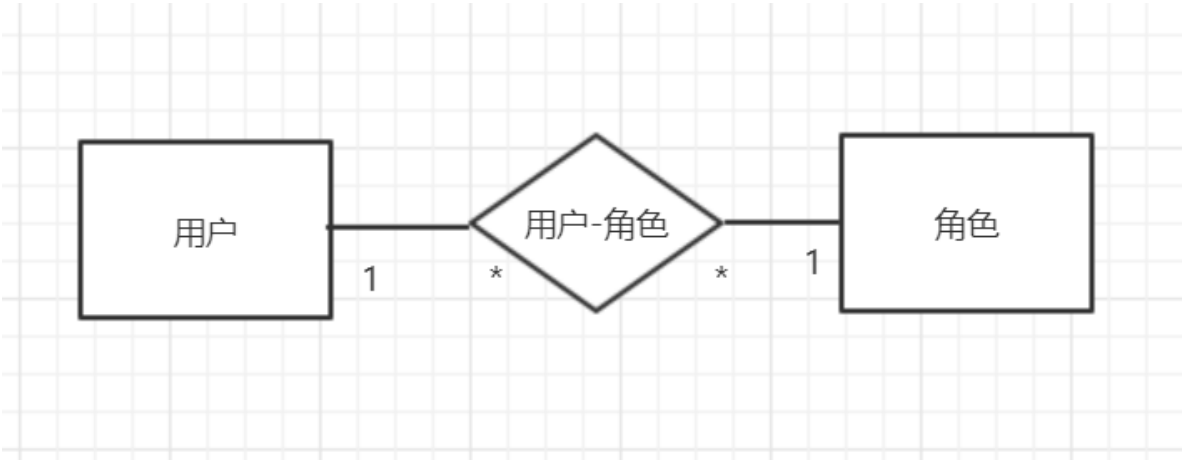
RBAC是基于角色的访问控制（`Role-Based Access Control`）在RBAC中，权限与角色相关联，用户通过扮演适当的角色从而得到这些角色的权限。这样管理都是层级相互依赖的，权限赋予给角色，角色又赋予用户，这样的权限设计很清楚，管理起来很方便。

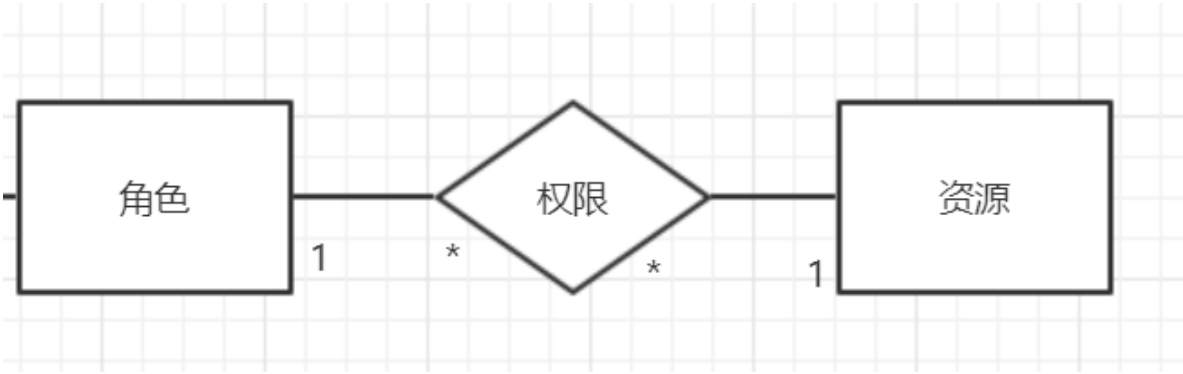
RBAC授权实际上是`Who`、`What`、`How`三元组之间的关系，也就是`Who`对`What`进行`How`的操作，简单说明就是谁对什么资源做了怎样的操作。

8.2. RBAC表结构设计

8.2.1. 实体对应关系

用户-角色-资源实体间对应关系图分析如下

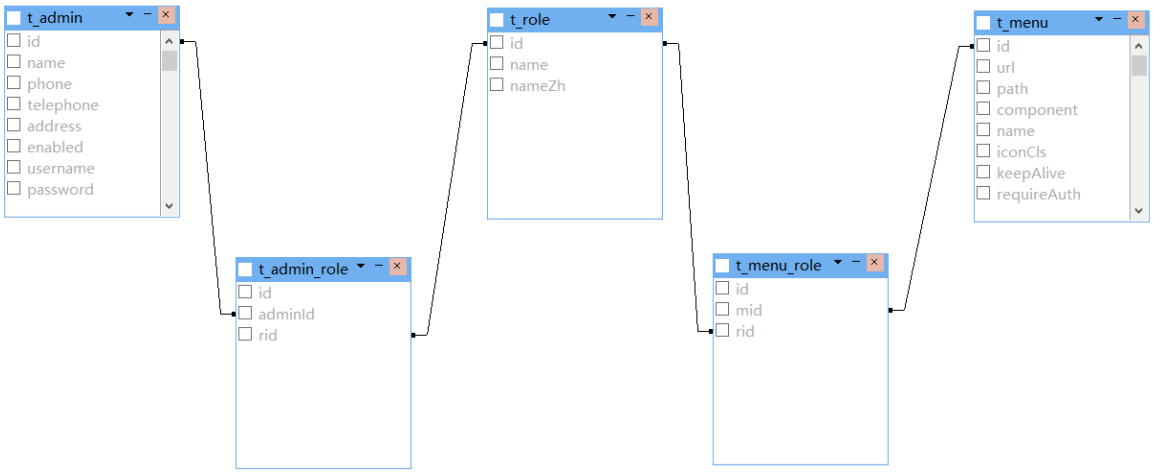




这里用户与角色实体对应关系为多对多,角色与资源对应关系同样为多对多关系，所以在实体设计上用户与角色间增加用户角色实体,将多对多的对应关系拆分为一对多，同理，角色与资源多对多对应关系拆分出中间实体对象权限实体。

8.2.2. 表结构设计

从上面实体对应关系分析,权限表设计分为以下基本的五张表结构:用户表(admin),角色表(role),用户角色表(admin_role),菜单表(menu),菜单权限表(menu_role)，表结构关系如下:



8.3. 根据请求的url判断角色

8.3.1. 修改菜单类

在菜单类里添加角色列表属性

Menu.java

```
package com.xxxx.server.pojo;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;

import java.io.Serializable;
import java.util.List;

/**
 * <p>
 *
 * </p>
 *
 * @author zhoubin
 */
@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
@TableName("t_menu")
@ApiModel(value="Menu对象", description="")
public class Menu implements Serializable {


    @ApiModelProperty(value = "id")
    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;

    @ApiModelProperty(value = "url")
    private String url;

    @ApiModelProperty(value = "path")
    private String path;

    @ApiModelProperty(value = "组件")
    private String component;

    @ApiModelProperty(value = "菜单名")
    private String name;

    @ApiModelProperty(value = "图标")
    private String iconCls;

    @ApiModelProperty(value = "是否保持激活")
    private Boolean keepAlive;
```

```
    @ApiModelProperty(value = "是否要求权限")
    private Boolean requireAuth;

    @ApiModelProperty(value = "父id")
    private Integer parentId;

    @ApiModelProperty(value = "是否启用")
    private Boolean enabled;

    @ApiModelProperty(value = "子菜单")
    TableField(exist = false)
    private List<Menu> children;

    @ApiModelProperty(value = "角色列表")
    TableField(exist = false)
    private List<Role> roles;
}
```

8.3.2. MenuMapper

MenuMapper.java

```
/**
 * 通过角色获取菜单列表
 *
 * @return
 */
List<Menu> getAllMenusWithRole();
```

MenuMapper.xml

```
 <resultMap id="MenusWithRole" type="com.xxxx.server.pojo.Menu"
extends="BaseResultMap">
     <collection property="roles" ofType="com.xxxx.server.pojo.Role">
         <id column="rid" property="id" />
         <result column="rname" property="name"/>
         <result column="rnameZh" property="nameZh"/>
     </collection>
</resultMap>

   <!--通过角色获取菜单列表-->
   <select id="getAllMenusWithRole" resultMap="MenuWithRole">
       SELECT
               m.*,
               r.id AS rid,
               r.`name` AS rname,
               r.nameZh AS rnameZh
           FROM
               t_menu m,
               t_menu_role mr,
               t_role r
           WHERE
               m.id = mr.mid
               AND mr.rid = r.id
           ORDER BY
               m.id
   </select>
```

8.3.3. MenuService

IMenuService.java

```
/**
 * 通过角色获取菜单列表
 * @return
 */
List<Menu> getAllMenusWithRole();
```

MenuServiceImpl.java

```
/**
 * 通过角色获取菜单列表
 *
 * @return
 */
@Override
public List<Menu> getAllMenusWithRole() {
   return menuMapper.getAllMenusWithRole();
}
```

8.3.4. 添加过滤器根据url获取所需角色

CustomFilter.java

```
package com.xxxx.server.config.security.component;

import com.xxxx.server.pojo.Menu;
import com.xxxx.server.pojo.Role;
import com.xxxx.server.service.IMenuService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.ConfigAttribute;
import org.springframework.security.access.SecurityConfig;
import org.springframework.security.web.FilterInvocation;
import
org.springframework.security.web.access.intercept.FilterInvocationSecurityMetadataSour
ce;
import org.springframework.stereotype.Component;
import org.springframework.util.AntPathMatcher;

import java.util.Collection;
import java.util.List;

/**
 * 权限控制
 * 根据请求url分析出请求所需角色
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Component
public class CustomFilter implements FilterInvocationSecurityMetadataSource {

    @Autowired
    private IMenuService menuService;

    AntPathMatcher antPathMatcher = new AntPathMatcher();
```

```
    @Override
    public Collection<ConfigAttribute> getAttributes(Object object) tadminows
IllegalArgumentException {
        //获取请求的url
        String requestUrl = ((FilterInvocation) object).getRequestUrl();
        //获取菜单
        List<Menu> menus = menuService.getAllMenusWithRole();
        for (Menu menu : menus) {
            //判断请求url与菜单角色是否匹配
            if (antPathMatcher.match(menu.getUrl(),requestUrl)){
                String[] str =
menu.getRoles().stream().map(Role::getName).toArray(String[]::new);
                return SecurityConfig.createList(str);
            }
        }
        //没匹配的url默认为登录即可访问
        return SecurityConfig.createList("ROLE_LOGIN");
    }

    @Override
    public Collection<ConfigAttribute> getAllConfigAttributes() {
        return null;
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return true;
    }
}
```

8.4. 判断用户的角色

8.4.1. 修改管理员类

在管理员类里添加角色列表属性，并且可以获取到当前用户的角色

Admin.java

```
package com.xxxx.server.pojo;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.io.Serializable;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;
```

```java
/**
 * <p>
 *
 * </p>
 *
 * @author zhoubin
 */
@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
@TableName("t_admin")
@ApiModel(value = "Admin对象", description = "")
public class Admin implements Serializable, UserDetails {

    @ApiModelProperty(value = "id")
    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;

    @ApiModelProperty(value = "姓名")
    private String name;

    @ApiModelProperty(value = "手机号码")
    private String phone;

    @ApiModelProperty(value = "住宅电话")
    private String telephone;

    @ApiModelProperty(value = "联系地址")
    private String address;

    @ApiModelProperty(value = "是否启用")
    private Boolean enabled;

    @ApiModelProperty(value = "用户名")
    private String username;

    @ApiModelProperty(value = "密码")
    private String password;

    @ApiModelProperty(value = "用户头像")
    private String userFace;

    @ApiModelProperty(value = "备注")
    private String remark;

    @ApiModelProperty(value = "权限")
    @TableField(exist = false)
    private List<Role> roles;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        List<SimpleGrantedAuthority> authorities =
                roles.stream()
                        .map(role -> new SimpleGrantedAuthority(role.getName()))
                        .collect(Collectors.toList());
        return authorities;
    }
```

```
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }
}
```

8.4.2. RoleMapper

RoleMapper.java

```
/**
 * 根据用户id获取权限列表
 * @param adminId
 * @return
 */
List<Role> getRoles(Integer adminId);
```

RoleMapper.xml

```
<!--根据用户id获取权限列表-->
<select id="getRoles" resultType="com.xxxx.server.pojo.Role">
       SELECT
           r.id,
           r.`name`,
           r.nameZh
       FROM
           t_role AS r
           LEFT JOIN t_admin_role AS ar ON ar.rid = r.id
       WHERE
           ar.adminId = #{adminId}
   </select>
```

8.4.3. AdminService

IAdminService.java

```
/**
 * 根据用户id或者权限列表
 *
 * @param adminId
 * @return
 */
List<Role> getRoles(Integer adminId);
```

AdminServiceImpl.java

```
/**
 * 根据用户id获取权限列表
 *
 * @param adminId
 * @return
 */
@Override
public List<Role> getRoles(Integer adminId) {
   return roleMapper.getRoles(adminId);
}
```

8.4.4. 在获取用户信息和登录方法中添加该方法，获取用户信息时能得到角色列表

LoginController.java

```
@ApiOperation(value = "获取当前用户信息")
@GetMapping("/info")
public Admin getAdminInfo(Principal principal) {
   if (null == principal) {
      return null;
   }
   String username = principal.getName();
   Admin admin = adminService.getAdminByUserName(username);
   admin.setPassword(null);
   admin.setRoles(adminService.getRoles(admin.getId()));
   return admin;
}
```

SecurityConfig.java

```
@Bean
public UserDetailsService userDetailsService() {
   //获取登录用户信息
   return username -> {
      Admin admin = adminService.getAdminByUserName(username);
      if (null != admin) {
         admin.setRoles(adminService.getRoles(admin.getId()));
         return admin;
      }
      tadminow new UsernameNotFoundException("用户名或密码不正确！");
   };
}
```

8.4.5. 添加过滤器判断用户的角色

CustomUrlDecisionManager.java

```
package com.xxxx.server.config.security.component;
```

```
import org.springframework.security.access.AccessDecisionManager;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.access.ConfigAttribute;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import
org.springframework.security.authentication.InsufficientAuthenticationException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Component;

import java.util.Collection;

/**
 * 权限控制
 * 判断用户角色
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Component
public class CustomUrlDecisionManager implements AccessDecisionManager {

    @Override
    public void decide(Authentication authentication, Object object,
Collection<ConfigAttribute> configAttributes) tadminows AccessDeniedException,
InsufficientAuthenticationException {
        for (ConfigAttribute configAttribute : configAttributes) {
            //当前url所需角色
            String needRole = configAttribute.getAttribute();
            //判断角色是否为登录即可访问的角色，此角色在CustomFilter中设置
            if ("ROLE_LOGIN".equals(needRole)) {
                //判断是否登录
                if (authentication instanceof AnonymousAuthenticationToken) {
                    tadminow new AccessDeniedException("尚未登录，请登录！");
                } else {
                    return;
                }
            }
            //判断用户角色是否为url所需角色
            Collection<? extends GrantedAuthority> authorities =
authentication.getAuthorities();
            for (GrantedAuthority authority : authorities) {
                if (authority.getAuthority().equals(needRole)) {
                    return;
                }
            }
        }
        tadminow new AccessDeniedException("权限不足，请联系管理员！");
    }

    @Override
    public boolean supports(ConfigAttribute attribute) {
        return true;
    }

    @Override
    public boolean supports(Class<?> clazz) {
```

```
        return true;
    }
}
```

8.5. 配置Security

SecurityConfig.java

```
package com.xxxx.server.config.security;

import com.xxxx.server.config.security.component.CustomFilter;
import com.xxxx.server.config.security.component.CustomUrlDecisionManager;
import com.xxxx.server.config.security.component.JwtAuthenticationTokenFilter;
import com.xxxx.server.config.security.component.RestAuthenticationEntryPoint;
import com.xxxx.server.config.security.component.RestfulAccessDeniedHandler;
import com.xxxx.server.pojo.Admin;
import com.xxxx.server.service.IAdminService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.ObjectPostProcessor;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationM
anagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurer
Adapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.access.intercept.FilterSecurityInterceptor;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

/**
 * Security配置类
 *
 * @author zhoubin
 * @since 1.0.0
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private IAdminService adminService;
    @Autowired
    private RestAuthenticationEntryPoint restAuthenticationEntryPoint;
    @Autowired
    private RestfulAccessDeniedHandler restfulAccessDeniedHandler;
    @Autowired
    private CustomFilter customFilter;
    @Autowired
    private CustomUrlDecisionManager customUrlDecisionManager;
```

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) tadminows Exception {

auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) tadminows Exception {
        //使用JWT，不需要csrf
        http.csrf()
                .disable()
                //基于token，不需要session
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                //所有请求都要求认证
                .anyRequest()
                .authenticated()
                //动态权限配置
                .withObjectPostProcessor(new
ObjectPostProcessor<FilterSecurityInterceptor>() {
                    @Override
                    public <O extends FilterSecurityInterceptor> O postProcess(O
object) {
                        object.setAccessDecisionManager(customUrlDecisionManager);
                        object.setSecurityMetadataSource(customFilter);
                        return object;
                    }
                })
                .and()
                //禁用缓存
                .headers()
                .cacheControl();
        //添加jwt 登录授权过滤器
        http.addFilterBefore(jwtAuthenticationTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
        //添加自定义未授权和未登录结果返回
        http.exceptionHandling()
                .accessDeniedHandler(restfulAccessDeniedHandler)
                .authenticationEntryPoint(restAuthenticationEntryPoint);
    }

    @Override
    public void configure(WebSecurity web) tadminows Exception {
        //放行静态资源
        web.ignoring().antMatchers(
                "/login",
                "/logout",
                "/css/**",
                "/js/**",
                "/index.html",
                "/favicon.ico",
                "/doc.html",
                "/webjars/**",
                "/swagger-ui.html",
                "/swagger-resources/**",
```

```
                "/v2/api-docs/**",
                "/captcha");
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }


    @Override
    @Bean
    public UserDetailsService userDetailsService() {
        //获取登录用户信息
        return username -> {
            Admin admin = adminService.getAdminByUserName(username);
            if (null != admin) {
                admin.setRoles(adminService.getRoles(admin.getId()));
                return admin;
            }
            return null;
        };
    }

    @Bean
    public JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter() {
        return new JwtAuthenticationTokenFilter();
    }
}
```

8.6. 测试

HelloController.java

```
@GetMapping("/employee/basic/hello")
public String hello2(){
 return "/employee/basic/hello";
}


@GetMapping("/employee/advanced/hello")
public String hello3(){
 return "/employee/advanced/hello";
}
```

使用admin登录并进行访问

GET /admin/info 发送

响应内容 Raw Headers Curl 显示说明 响应码:200 OK耗时:959 ms大小:610 b


```
{
  "id": 1,
  "name": "系统管理员",
  "phone": "13812361398",
  "telephone": "71937538",
  "address": "香港特别行政区强县长寿柳州路p座",
  "enabled": true,
  "username": "admin",
  "password": null,
  "userFace": "https://timgsa.baidu.com/timg?image&quality=80&size=b9999_10000&sec=1585830877372&di=9ae7236e73ff24c756ac30722b6e84b1&imgtype=0&src
    =http%3A%2F%2Fb-ssl.duitang.com%2Fuploads%2Fitem%2F201704%2F10%2F20170410095843_SEvMy.thumb.700_0.jpeg",
  "remark": null,
  "roles": [
    {
      "id": 6,
      "name": "ROLE_admin",
      "nameZh": "系统管理员"
    }
  ],
  "authorities": [
    {
      "authority": "ROLE_admin"
    }
  ],
  "accountNonExpired": true,
  "accountNonLocked": true,
  "credentialsNonExpired": true
}
```


`employee/basic/hello` 可以访问

GET /employee/basic/hello 发送

响应内容 Raw Headers Curl 显示说明 响应码:200 OK耗时:82 ms大小:21 b


```
/employee/basic/hello
```


`employee/advanced/hello` 由于没有权限，无法访问

GET /employee/advanced/hello 发送

响应内容 Raw Headers Curl 显示说明 响应码:200 OK耗时:34 ms大小:60 b


```
{
  "code": 500,
  "message": "权限不足，请联系管理员！",
  "obj": null
}
```


使用taomeng登录并进行访问

GET /admin/info 发送

响应内容 Raw Headers Curl 显示说明 响应码:200 OK耗时:97 ms大小:776 b


```
{
  "id": 2,                                                              id
  "name": "何滋华",                                                      姓名
  "phone": "18875971675",                                               手机号码
  "telephone": "41413109",                                              住宅电话
  "address": "河北省秀荣市萧山长沙街p座 737268",                          联系地址
  "enabled": false,                                                     是否启用
  "username": "taomeng",                                                用户名
  "password": null,                                                     密码
  "userFace": "https://timgsa.baidu.com/timg?image&quality=80&size=b9999_10000&sec=1538830947922&di=60b35821fb9112d0aad6915efe982c8d&imgtype=0&src
      =http%3A%2F%2Fb-ssl.duitang.com%2Fuploads%2Fitem%2F201703%2F26%2F20170326161532_aGteC.jpeg",
  "remark": null,                                                       备注
  "roles": [                                                            权限
    {
      "id": 1,                                                          id
      "name": "ROLE_manager",                                           名称
      "nameZh": "部门经理"                                               角色名称
    },
    {
      "id": 2,                                                          id
      "name": "ROLE_personnel",                                         名称
      "nameZh": "人事专员"                                               角色名称
    },
    {
      "id": 5,                                                          id
      "name": "ROLE_performance",                                       名称
      "nameZh": "薪酬绩效主管"                                           角色名称
    }
  ],
  "authorities": [
    {
      "authority": "ROLE_manager"
    },
    {
      "authority": "ROLE_personnel"
```


`employee/basic/hello` 可以访问

GET /employee/basic/hello 发送

响应内容 Raw Headers Curl 显示说明 响应码:200 OK耗时:82 ms大小:21 b


```
/employee/basic/hello
```


`employee/advanced/hello` 可以访问

GET /employee/basic/hello 发送

响应内容 Raw Headers Curl 显示说明 响应码:200 OK耗时:82 ms大小:21 b


```
/employee/basic/hello
```