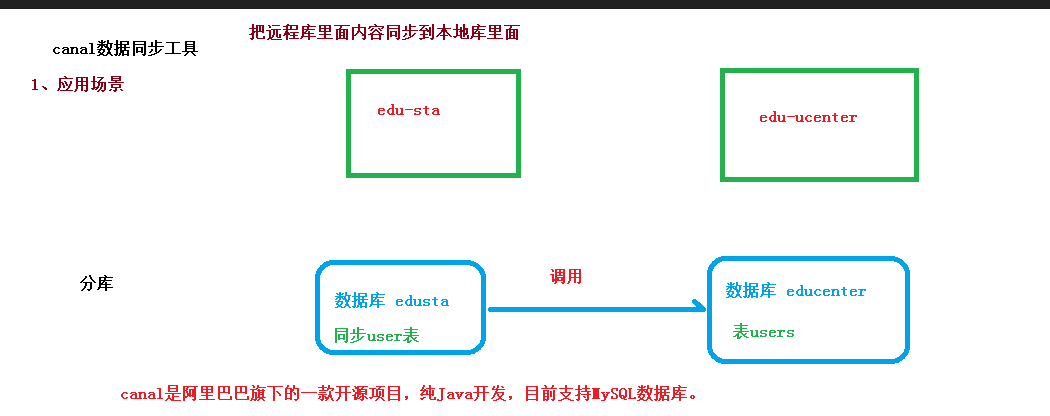
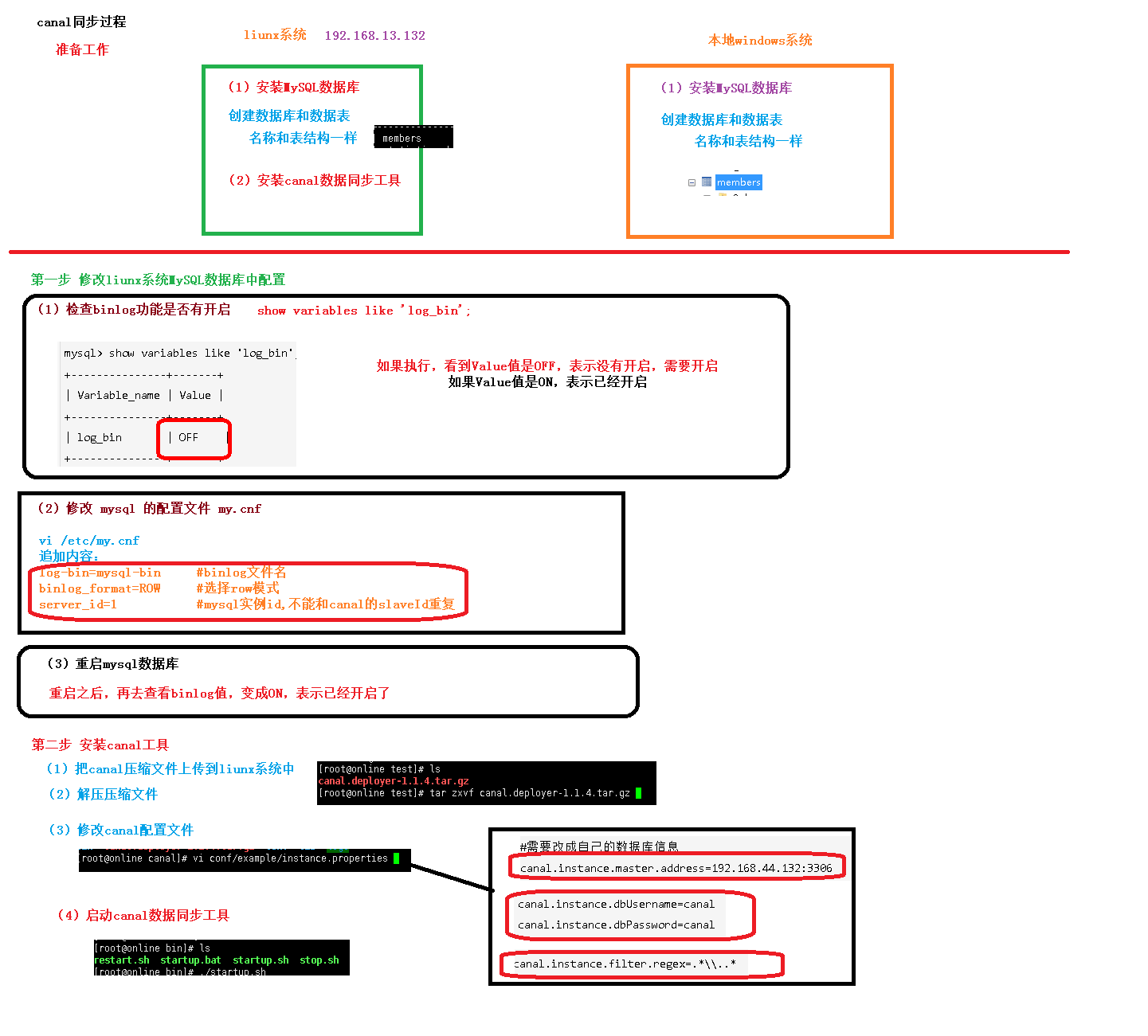


# 1、Canal数据同步工具

## 一、Canal介绍



远程库 linux在系统中



### 1、应用场景

在前面的统计分析功能中，我们采取了服务调用获取统计数据，这样耦合度高，效率相对较低，目前我采取另一种实现方式，通过实时同步数据库表的方式实现，例如我们要统计每天注册与登录人数，我们只需把会员表同步到统计库中，实现本地统计就可以了，这样效率更高，耦合度更低，Canal就是一个很好的数据库同步工具。canal是阿里巴巴旗下的一款开源项目，纯Java开发。基于数据库增量日志解析，提供增量数据订阅&消费，目前主要支持了MySQL。

### 2、Canal环境搭建

**canal的原理是基于mysql binlog技术，所以这里一定需要开启mysql的binlog写入功能**

**开启mysql服务：**  service mysql start  **（或者 systemctl start mysqld.service）**

#### （1）检查binlog功能是否有开启

mysql> show variables like 'log\_bin';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| log\_bin | OFF |

+---------------+-------+

1 row in set (0.00 sec)

#### （2）如果显示状态为OFF表示该功能未开启，开启binlog功能

**1，修改 mysql 的配置文件 my.cnf**

vi /etc/my.cnf

**追加内容：**

log-bin=mysql-bin #binlog文件名

binlog\_format=ROW #选择row模式

server\_id=1 #mysql实例id,不能和canal的slaveId重复

**2，重启 mysql：**

service mysql restart

**3，登录 mysql 客户端，查看 log\_bin 变量**

mysql> show variables like 'log\_bin';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| log\_bin | ON|

+---------------+-------+

1 row in set (0.00 sec)

————————————————

如果显示状态为ON表示该功能已开启

#### （3）在mysql里面添加以下的相关用户和权限

CREATE USER 'canal'@'%' IDENTIFIED BY 'canal';

GRANT SHOW VIEW, SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'canal'@'%';

FLUSH PRIVILEGES;

### 3、下载安装Canal服务

下载地址：

<https://github.com/alibaba/canal/releases>

#### （1）下载之后，放到目录中，解压文件

cd /usr/local/canal

canal.deployer-1.1.4.tar.gz

tar zxvf canal.deployer-1.1.4.tar.gz

#### （2）修改配置文件

vi conf/example/instance.properties

#需要改成自己的数据库信息

canal.instance.master.address=192.168.44.132:3306

#需要改成自己的数据库用户名与密码

canal.instance.dbUsername=canal

canal.instance.dbPassword=canal

#需要改成同步的数据库表规则，例如只是同步一下表

#canal.instance.filter.regex=.\*\\..\*

canal.instance.filter.regex=guli\_ucenter.ucenter\_member

注：

mysql 数据解析关注的表，Perl正则表达式.

多个正则之间以逗号(,)分隔，转义符需要双斜杠(\\)

常见例子：

1.  所有表：.\*   or  .\*\\..\*

2.  canal schema下所有表： canal\\..\*

3.  canal下的以canal打头的表：canal\\.canal.\*

4.  canal schema下的一张表：canal.test1

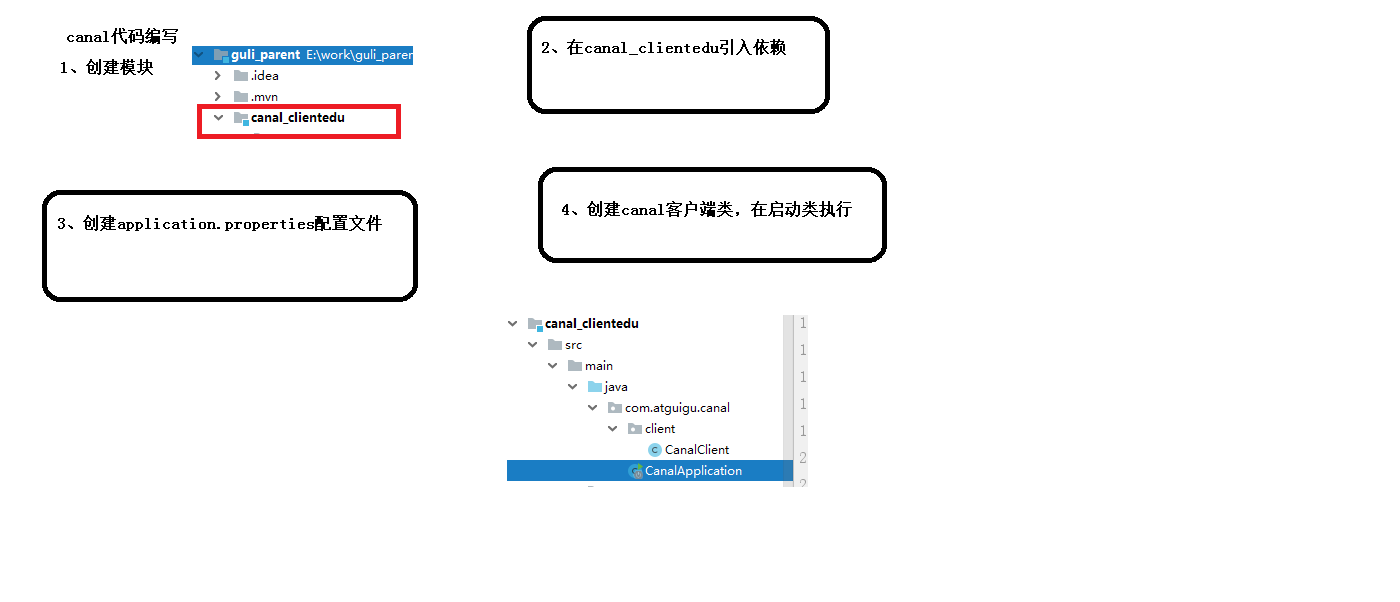
5.  多个规则组合使用：canal\\..\*,mysql.test1,mysql.test2 (逗号分隔)

注意：此过滤条件只针对row模式的数据有效(ps. mixed/statement因为不解析sql，所以无法准确提取tableName进行过滤)

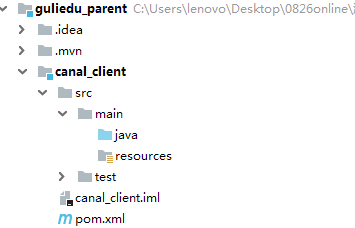
#### （3）进入bin目录下启动

**sh bin/startup.sh**

## 二、创建canal\_client模块



### 1、在guliedu\_parent下创建canal\_client模块

****

### 2、引入相关依赖

*<*dependencies*>*

*<*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-web*</*artifactId*>  
 </*dependency*>* <!--mysql-->  
 *<*dependency*>  
 <*groupId*>*mysql*</*groupId*>  
 <*artifactId*>*mysql-connector-java*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*commons-dbutils*</*groupId*>  
 <*artifactId*>*commons-dbutils*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-jdbc*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.alibaba.otter*</*groupId*>  
 <*artifactId*>*canal.client*</*artifactId*>  
 </*dependency*>  
</*dependencies*>*

### 3、创建application.properties配置文件

# 服务端口

server.port=10000  
# 服务名  
spring.application.name=canal-client  
# 环境设置：dev、test、prod  
spring.profiles.active=dev  
# mysql数据库连接  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/guli?serverTimezone=GMT%2B8  
spring.datasource.username=root  
spring.datasource.password=root

### 4、编写canal客户端类

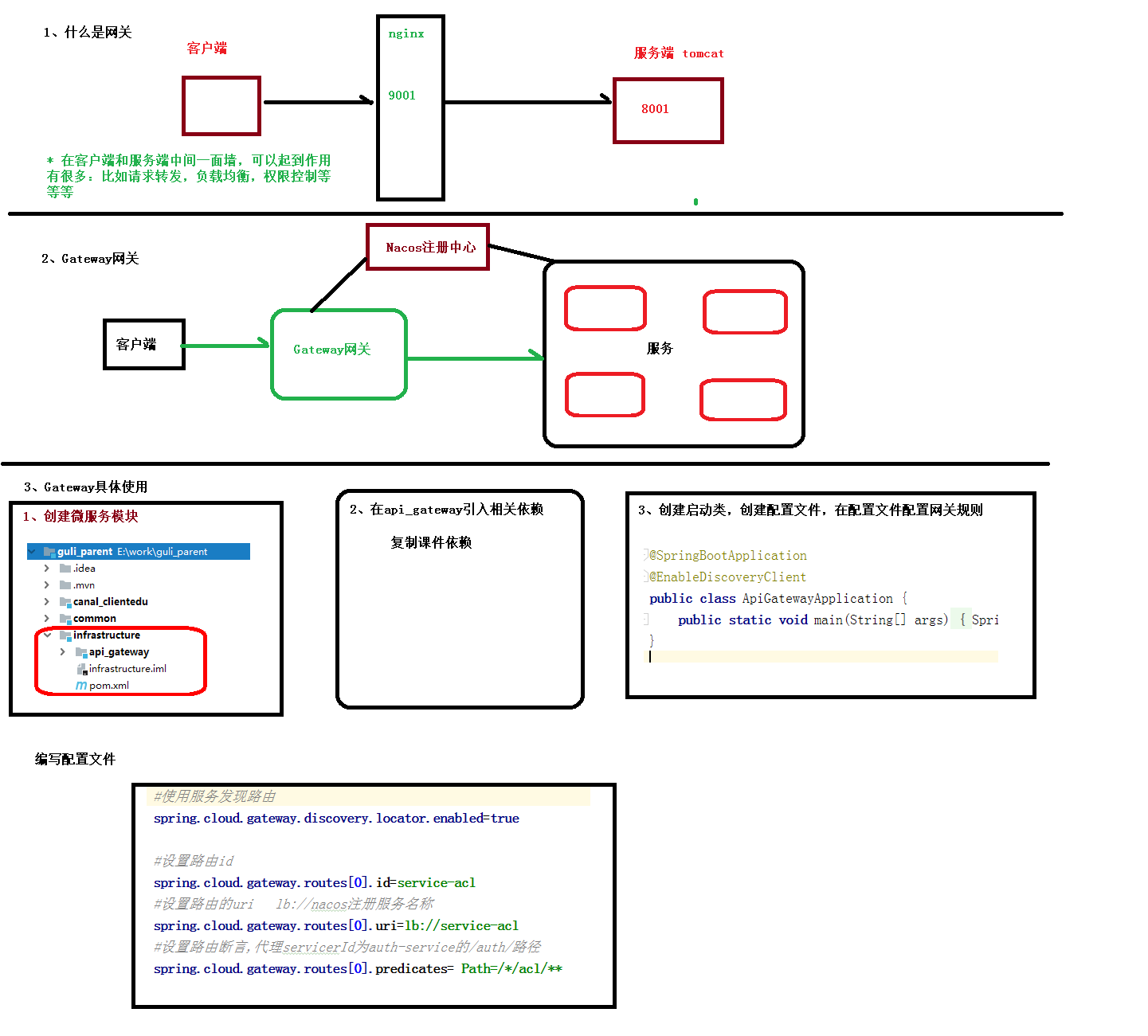
import com.alibaba.otter.canal.client.CanalConnector;  
import com.alibaba.otter.canal.client.CanalConnectors;  
import com.alibaba.otter.canal.protocol.CanalEntry.\*;  
import com.alibaba.otter.canal.protocol.Message;  
import com.google.protobuf.InvalidProtocolBufferException;  
import org.apache.commons.dbutils.DbUtils;  
import org.apache.commons.dbutils.QueryRunner;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.Resource;  
import javax.sql.DataSource;  
import java.net.InetSocketAddress;  
import java.sql.Connection;  
import java.sql.SQLException;  
import java.util.Iterator;  
import java.util.List;  
import java.util.Queue;  
import java.util.concurrent.ConcurrentLinkedQueue;  
  
@Component  
public class CanalClient *{* //sql队列  
 private Queue*<*String*>* SQL\_QUEUE = new ConcurrentLinkedQueue*<>()*;  
 @Resource  
 private DataSource dataSource;  
  
 */\*\*  
 \* canal入库方法  
 \*/* public void run*() {* CanalConnector connector = CanalConnectors.*newSingleConnector(* new InetSocketAddress*(*"192.168.64.129",//远程库ip  
 11111*)*, //canal默认端口  
 "example", //配置文件路径 vi conf/example/instance.properties  
 "canal",//用户名  
 "canal"*)*;//密码  
 int batchSize = 1000;  
 try *{* connector.connect*()*;  
 connector.subscribe*(*".\*\\..\*"*)*;  
 connector.rollback*()*;  
 try *{* while *(*true*) {* //尝试从master那边拉去数据batchSize条记录，有多少取多少  
 Message message = connector.getWithoutAck*(*batchSize*)*;  
 long batchId = message.getId*()*;  
 int size = message.getEntries*()*.size*()*;  
 if *(*batchId == -1 || size == 0*) {* Thread.*sleep(*1000*)*;  
 *}* else *{* dataHandle*(*message.getEntries*())*;  
 *}* connector.ack*(*batchId*)*;  
 //当队列里面堆积的sql大于一定数值的时候就模拟执行  
 if *(*SQL\_QUEUE.size*()* >= 1*) {* executeQueueSql*()*;  
 *}  
 }  
 }* catch *(*InterruptedException e*) {* e.printStackTrace*()*;  
 *}* catch *(*InvalidProtocolBufferException e*) {* e.printStackTrace*()*;  
 *}  
 }* finally *{* connector.disconnect*()*;  
 *}  
 }  
  
 /\*\*  
 \* 模拟执行队列里面的sql语句  
 \*/* public void executeQueueSql*() {* int size = SQL\_QUEUE.size*()*;  
 for *(*int i = 0; i < size; i++*) {* String sql = SQL\_QUEUE.poll*()*;  
 System.*out*.println*(*"[sql]----> " + sql*)*;  
 this.execute*(*sql.toString*())*;  
 *}  
 }  
  
 /\*\*  
 \* 数据处理  
 \*  
 \** ***@param*** *entrys  
 \*/* private void dataHandle*(*List*<*Entry*>* entrys*)* throws InvalidProtocolBufferException *{* for *(*Entry entry : entrys*) {* if *(*EntryType.*ROWDATA* == entry.getEntryType*()) {* RowChange rowChange = RowChange.*parseFrom(*entry.getStoreValue*())*;  
 EventType eventType = rowChange.getEventType*()*;  
 if *(*eventType == EventType.*DELETE) {* saveDeleteSql*(*entry*)*;  
 *}* else if *(*eventType == EventType.*UPDATE) {* saveUpdateSql*(*entry*)*;  
 *}* else if *(*eventType == EventType.*INSERT) {* saveInsertSql*(*entry*)*;  
 *}  
 }  
 }  
 }  
  
 /\*\*  
 \* 保存更新语句  
 \*  
 \** ***@param*** *entry  
 \*/* private void saveUpdateSql*(*Entry entry*) {* try *{* RowChange rowChange = RowChange.*parseFrom(*entry.getStoreValue*())*;  
 List*<*RowData*>* rowDatasList = rowChange.getRowDatasList*()*;  
 for *(*RowData rowData : rowDatasList*) {* List*<*Column*>* newColumnList = rowData.getAfterColumnsList*()*;  
 StringBuffer sql = new StringBuffer*(*"update " + entry.getHeader*()*.getTableName*()* + " set "*)*;  
 for *(*int i = 0; i < newColumnList.size*()*; i++*) {* sql.append*(*" " + newColumnList.get*(*i*)*.getName*()* + " = '" + newColumnList.get*(*i*)*.getValue*()* + "'"*)*;  
 if *(*i != newColumnList.size*()* - 1*) {* sql.append*(*","*)*;  
 *}  
 }* sql.append*(*" where "*)*;  
 List*<*Column*>* oldColumnList = rowData.getBeforeColumnsList*()*;  
 for *(*Column column : oldColumnList*) {* if *(*column.getIsKey*()) {* //暂时只支持单一主键  
 sql.append*(*column.getName*()* + "=" + column.getValue*())*;  
 break;  
 *}  
 }* SQL\_QUEUE.add*(*sql.toString*())*;  
 *}  
 }* catch *(*InvalidProtocolBufferException e*) {* e.printStackTrace*()*;  
 *}  
 }  
  
 /\*\*  
 \* 保存删除语句  
 \*  
 \** ***@param*** *entry  
 \*/* private void saveDeleteSql*(*Entry entry*) {* try *{* RowChange rowChange = RowChange.*parseFrom(*entry.getStoreValue*())*;  
 List*<*RowData*>* rowDatasList = rowChange.getRowDatasList*()*;  
 for *(*RowData rowData : rowDatasList*) {* List*<*Column*>* columnList = rowData.getBeforeColumnsList*()*;  
 StringBuffer sql = new StringBuffer*(*"delete from " + entry.getHeader*()*.getTableName*()* + " where "*)*;  
 for *(*Column column : columnList*) {* if *(*column.getIsKey*()) {* //暂时只支持单一主键  
 sql.append*(*column.getName*()* + "=" + column.getValue*())*;  
 break;  
 *}  
 }* SQL\_QUEUE.add*(*sql.toString*())*;  
 *}  
 }* catch *(*InvalidProtocolBufferException e*) {* e.printStackTrace*()*;  
 *}  
 }  
  
 /\*\*  
 \* 保存插入语句  
 \*  
 \** ***@param*** *entry  
 \*/* private void saveInsertSql*(*Entry entry*) {* try *{* RowChange rowChange = RowChange.*parseFrom(*entry.getStoreValue*())*;  
 List*<*RowData*>* rowDatasList = rowChange.getRowDatasList*()*;  
 for *(*RowData rowData : rowDatasList*) {* List*<*Column*>* columnList = rowData.getAfterColumnsList*()*;  
 StringBuffer sql = new StringBuffer*(*"insert into " + entry.getHeader*()*.getTableName*()* + " ("*)*;  
 for *(*int i = 0; i < columnList.size*()*; i++*) {* sql.append*(*columnList.get*(*i*)*.getName*())*;  
 if *(*i != columnList.size*()* - 1*) {* sql.append*(*","*)*;  
 *}  
 }* sql.append*(*") VALUES ("*)*;  
 for *(*int i = 0; i < columnList.size*()*; i++*) {* sql.append*(*"'" + columnList.get*(*i*)*.getValue*()* + "'"*)*;  
 if *(*i != columnList.size*()* - 1*) {* sql.append*(*","*)*;  
 *}  
 }* sql.append*(*")"*)*;  
 SQL\_QUEUE.add*(*sql.toString*())*;  
 *}  
 }* catch *(*InvalidProtocolBufferException e*) {* e.printStackTrace*()*;  
 *}  
 }  
  
 /\*\*  
 \* 入库  
 \*  
 \** ***@param*** *sql  
 \*/* public void execute*(*String sql*) {* Connection con = null;  
 try *{* if *(*null == sql*)* return;  
 con = dataSource.getConnection*()*;  
 QueryRunner qr = new QueryRunner*()*;  
 int row = qr.execute*(*con, sql*)*;  
 System.*out*.println*(*"update: " + row*)*;  
 *}* catch *(*SQLException e*) {* e.printStackTrace*()*;  
 *}* finally *{* DbUtils.*closeQuietly(*con*)*;  
 *}  
 }  
}*

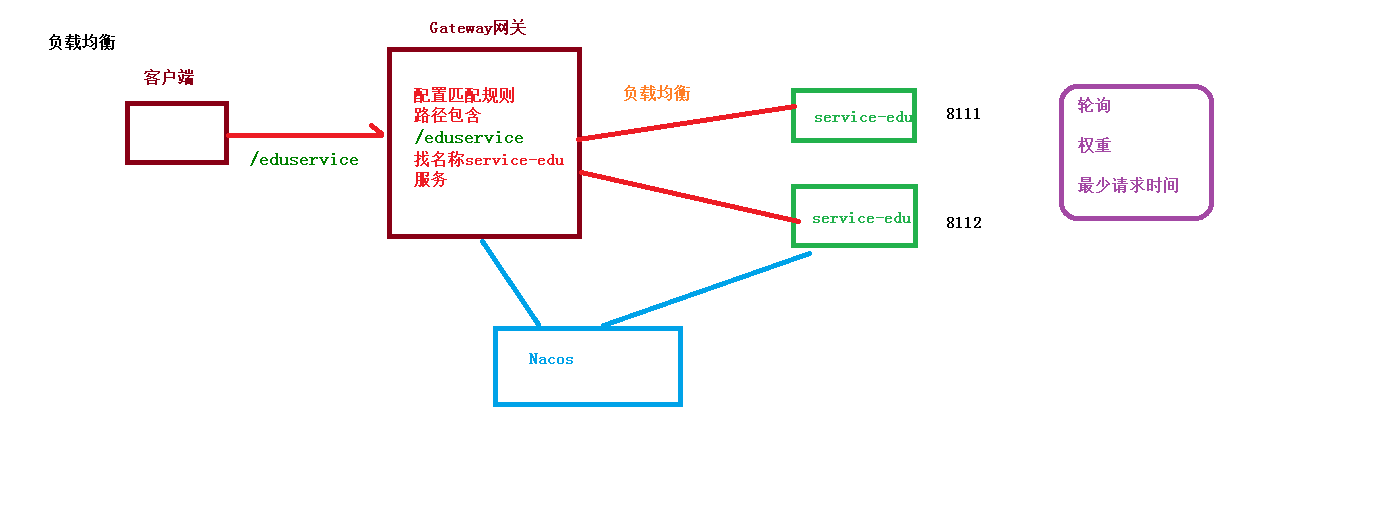
### 5、创建启动类

@SpringBootApplication

public class CanalApplication implements CommandLineRunner *{* @Resource  
 private CanalClient canalClient;  
  
 public static void main*(*String*[]* args*) {* SpringApplication.*run(*CanalApplication.class, args*)*;  
 *}* @Override  
 public void run*(*String... strings*)* throws Exception *{* //项目启动，执行canal客户端监听  
 canalClient.run*()*;  
 *}  
}*

# 2 SpringCloud【GateWay网关】





## 一、网关基本概念

### 1、API网关介绍

API 网关出现的原因是微服务架构的出现，不同的微服务一般会有不同的网络地址，而外部客户端可能需要调用多个服务的接口才能完成一个业务需求，如果让客户端直接与各个微服务通信，会有以下的问题：

（1）客户端会多次请求不同的微服务，增加了客户端的复杂性。

（2）存在跨域请求，在一定场景下处理相对复杂。

（3）认证复杂，每个服务都需要独立认证。

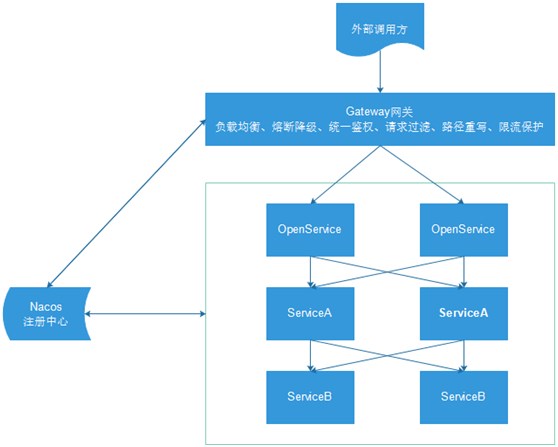
（4）难以重构，随着项目的迭代，可能需要重新划分微服务。例如，可能将多个服务合并成一个或者将一个服务拆分成多个。如果客户端直接与微服务通信，那么重构将会很难实施。

（5）某些微服务可能使用了防火墙 / 浏览器不友好的协议，直接访问会有一定的困难。

以上这些问题可以借助 API 网关解决。API 网关是介于客户端和服务器端之间的中间层，所有的外部请求都会先经过 API 网关这一层。也就是说，API 的实现方面更多的考虑业务逻辑，而安全、性能、监控可以交由 API 网关来做，这样既提高业务灵活性又不缺安全性

### 2、Spring Cloud Gateway

**Spring cloud gateway**是spring官方基于Spring 5.0、Spring Boot2.0和Project Reactor等技术开发的网关，Spring Cloud Gateway旨在为微服务架构提供简单、有效和统一的API路由管理方式，Spring Cloud Gateway作为Spring Cloud生态系统中的网关，目标是替代Netflix Zuul，其不仅提供统一的路由方式，并且还基于Filer链的方式提供了网关基本的功能，例如：安全、监控/埋点、限流等。



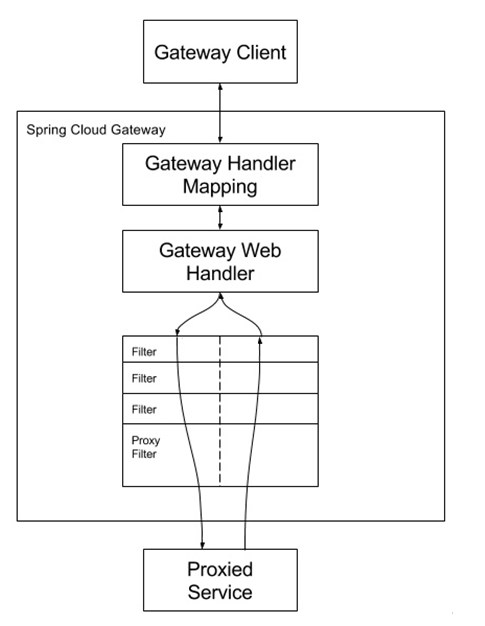
### 3、Spring Cloud Gateway核心概念

网关提供API全托管服务，丰富的API管理功能，辅助企业管理大规模的API，以降低管理成本和安全风险，包括协议适配、协议转发、安全策略、防刷、流量、监控日志等贡呢。一般来说网关对外暴露的URL或者接口信息，我们统称为路由信息。如果研发过网关中间件或者使用过Zuul的人，会知道网关的核心是Filter以及Filter Chain（Filter责任链）。Sprig Cloud Gateway也具有路由和Filter的概念。下面介绍一下Spring Cloud Gateway中几个重要的概念。

（1）路由。路由是网关最基础的部分，路由信息有一个ID、一个目的URL、一组断言和一组Filter组成。如果断言路由为真，则说明请求的URL和配置匹配

（2）断言。Java8中的断言函数。Spring Cloud Gateway中的断言函数输入类型是Spring5.0框架中的ServerWebExchange。Spring Cloud Gateway中的断言函数允许开发者去定义匹配来自于http request中的任何信息，比如请求头和参数等。

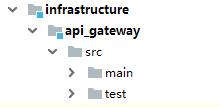
（3）过滤器。一个标准的Spring webFilter。Spring cloud gateway中的filter分为两种类型的Filter，分别是Gateway Filter和Global Filter。过滤器Filter将会对请求和响应进行修改处理



如上图所示，Spring cloud Gateway发出请求。然后再由Gateway Handler Mapping中找到与请求相匹配的路由，将其发送到Gateway web handler。Handler再通过指定的过滤器链将请求发送到我们实际的服务执行业务逻辑，然后返回。

## 二、创建api-gateway模块（网关服务）

### 1、在infrastructure模块下创建api\_gateway模块

****

### 2、在pom.xml引入依赖

*<*dependencies*>*

*<*dependency*>  
 <*groupId*>*com.atguigu*</*groupId*>  
 <*artifactId*>*common\_utils*</*artifactId*>  
 <*version*>*0.0.1-SNAPSHOT*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-alibaba-nacos-discovery*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-gateway*</*artifactId*>  
 </*dependency*>* <!--gson-->  
 *<*dependency*>  
 <*groupId*>*com.google.code.gson*</*groupId*>  
 <*artifactId*>*gson*</*artifactId*>  
 </*dependency*>* <!--服务调用-->  
 *<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-openfeign*</*artifactId*>  
 </*dependency*>  
</*dependencies*>*

### 3、编写application.properties配置文件

# 服务端口

server.port=8222  
# 服务名  
spring.application.name=service-gateway  
# nacos服务地址  
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848  
  
  
#使用服务发现路由  
spring.cloud.gateway.discovery.locator.enabled=true  
#服务路由名小写  
#spring.cloud.gateway.discovery.locator.lower-case-service-id=true  
#设置路由id 可以随意 一般服务名  
spring.cloud.gateway.routes[0].id=service-acl  
#设置路由的uri 【nacos服务名】  
spring.cloud.gateway.routes[0].uri=lb://service-acl  
#设置路由断言,代理servicerId为auth-service的/auth/路径 匹配规则  
spring.cloud.gateway.routes[0].predicates=Path=/\*/acl/\*\*  
#配置service-edu服务  
spring.cloud.gateway.routes[1].id=service-edu  
spring.cloud.gateway.routes[1].uri=lb://service-edu  
spring.cloud.gateway.routes[1].predicates=Path=/eduservice/\*\*  
#配置service-ucenter服务  
spring.cloud.gateway.routes[2].id=service-ucenter  
spring.cloud.gateway.routes[2].uri=lb://service-ucenter  
spring.cloud.gateway.routes[2].predicates=Path=/ucenterservice/\*\*  
#配置service-ucenter服务  
spring.cloud.gateway.routes[3].id=service-cms  
spring.cloud.gateway.routes[3].uri=lb://service-cms  
spring.cloud.gateway.routes[3].predicates=Path=/cmsservice/\*\*  
spring.cloud.gateway.routes[4].id=service-msm  
spring.cloud.gateway.routes[4].uri=lb://service-msm  
spring.cloud.gateway.routes[4].predicates=Path=/edumsm/\*\*  
spring.cloud.gateway.routes[5].id=service-order  
spring.cloud.gateway.routes[5].uri=lb://service-order  
spring.cloud.gateway.routes[5].predicates=Path=/orderservice/\*\*  
spring.cloud.gateway.routes[6].id=service-order  
spring.cloud.gateway.routes[6].uri=lb://service-order  
spring.cloud.gateway.routes[6].predicates=Path=/orderservice/\*\*  
spring.cloud.gateway.routes[7].id=service-oss  
spring.cloud.gateway.routes[7].uri=lb://service-oss  
spring.cloud.gateway.routes[7].predicates=Path=/eduoss/\*\*  
spring.cloud.gateway.routes[8].id=service-statistic  
spring.cloud.gateway.routes[8].uri=lb://service-statistic  
spring.cloud.gateway.routes[8].predicates=Path=/staservice/\*\*  
spring.cloud.gateway.routes[9].id=service-vod  
spring.cloud.gateway.routes[9].uri=lb://service-vod  
spring.cloud.gateway.routes[9].predicates=Path=/eduvod/\*\*  
spring.cloud.gateway.routes[10].id=service-edu  
spring.cloud.gateway.routes[10].uri=lb://service-edu  
spring.cloud.gateway.routes[10].predicates=Path=/eduuser/\*\*

### yml文件：

server:

port: 8222

spring:

application:

cloud:

gateway:

discovery:

locator:

enabled: true

routes:

- id: SERVICE-ACL

uri: lb://SERVICE-ACL

predicates:

- Path=/\*/acl/\*\* # 路径匹配

- id: SERVICE-EDU

uri: lb://SERVICE-EDU

predicates:

- Path=/eduservice/\*\* # 路径匹配

- id: SERVICE-UCENTER

uri: lb://SERVICE-UCENTER

predicates:

- Path=/ucenter/\*\* # 路径匹配

nacos:

discovery:

server-addr: 127.0.0.1:8848

### 4、编写启动类

@SpringBootApplication、

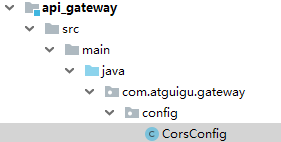
@EnableDiscoveryClient//nacos

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

## 三、网关相关配置

### 1、网关解决跨域问题

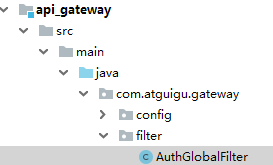
#### （1）创建配置类



import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;  
import org.springframework.web.cors.CorsConfiguration;  
  
import org.springframework.web.cors.reactive.CorsWebFilter;  
import org.springframework.web.cors.reactive.UrlBasedCorsConfigurationSource;  
import org.springframework.web.util.pattern.PathPatternParser;  
  
@Configuration  
public class CorsConfig *{* @Bean  
 public CorsWebFilter corsFilter*() {* CorsConfiguration config = new CorsConfiguration*()*;  
 config.addAllowedMethod*(*"\*"*)*;  
 config.addAllowedOrigin*(*"\*"*)*;  
 config.addAllowedHeader*(*"\*"*)*;  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource*(*new PathPatternParser*())*;  
 source.registerCorsConfiguration*(*"/\*\*", config*)*;  
 return new CorsWebFilter*(*source*)*;  
 *}  
}*

### 2、全局Filter，统一处理会员登录与外部不允许访问的服务

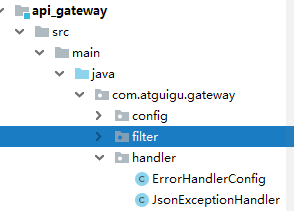


import com.google.gson.JsonObject;

import org.springframework.cloud.gateway.filter.GatewayFilterChain;  
import org.springframework.cloud.gateway.filter.GlobalFilter;  
import org.springframework.core.Ordered;  
import org.springframework.core.io.buffer.DataBuffer;  
import org.springframework.http.server.reactive.ServerHttpRequest;  
import org.springframework.http.server.reactive.ServerHttpResponse;  
import org.springframework.stereotype.Component;  
import org.springframework.util.AntPathMatcher;  
import org.springframework.web.server.ServerWebExchange;  
import reactor.core.publisher.Mono;  
  
import java.nio.charset.StandardCharsets;  
import java.util.List;  
  
*/\*\*  
 \* <p>  
 \* 全局Filter，统一处理会员登录与外部不允许访问的服务  
 \* </p>  
 \*/*@Component  
public class AuthGlobalFilter implements GlobalFilter, Ordered *{* private AntPathMatcher antPathMatcher = new AntPathMatcher*()*;  
  
 @Override  
 public Mono*<*Void*>* filter*(*ServerWebExchange exchange, GatewayFilterChain chain*) {* ServerHttpRequest request = exchange.getRequest*()*;  
 String path = request.getURI*()*.getPath*()*;  
 //谷粒学院api接口，校验用户必须登录  
 if *(*antPathMatcher.match*(*"/api/\*\*/auth/\*\*", path*)) {* List*<*String*>* tokenList = request.getHeaders*()*.get*(*"token"*)*;  
 if *(*null == tokenList*) {* ServerHttpResponse response = exchange.getResponse*()*;  
 return out*(*response*)*;  
 *}* else *{*// Boolean isCheck = JwtUtils.checkToken(tokenList.get(0));  
// if(!isCheck) {  
 ServerHttpResponse response = exchange.getResponse*()*;  
 return out*(*response*)*;  
// }  
 *}  
 }* //内部服务接口，不允许外部访问  
 if *(*antPathMatcher.match*(*"/\*\*/inner/\*\*", path*)) {* ServerHttpResponse response = exchange.getResponse*()*;  
 return out*(*response*)*;  
 *}* return chain.filter*(*exchange*)*;  
 *}* @Override  
 public int getOrder*() {* return 0;  
 *}* private Mono*<*Void*>* out*(*ServerHttpResponse response*) {* JsonObject message = new JsonObject*()*;  
 message.addProperty*(*"success", false*)*;  
 message.addProperty*(*"code", 28004*)*;  
 message.addProperty*(*"data", "鉴权失败"*)*;  
 byte*[]* bits = message.toString*()*.getBytes*(*StandardCharsets.*UTF\_8)*;  
 DataBuffer buffer = response.bufferFactory*()*.wrap*(*bits*)*;  
 //response.setStatusCode(HttpStatus.UNAUTHORIZED);  
 //指定编码，否则在浏览器中会中文乱码  
 response.getHeaders*()*.add*(*"Content-Type", "application/json;charset=UTF-8"*)*;  
 return response.writeWith*(*Mono.*just(*buffer*))*;  
 *}  
}*

### 3、自定义异常处理

**服务网关调用服务时可能会有一些异常或服务不可用，它返回错误信息不友好，需要我们覆盖处理**

****

**ErrorHandlerConfig：**

impor org.springframework.beans.factory.ObjectProvider;

import org.springframework.boot.autoconfigure.web.ResourceProperties;  
import org.springframework.boot.autoconfigure.web.ServerProperties;  
import org.springframework.boot.context.properties.EnableConfigurationProperties;  
import org.springframework.boot.web.reactive.error.ErrorAttributes;  
import org.springframework.boot.web.reactive.error.ErrorWebExceptionHandler;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.core.Ordered;  
import org.springframework.core.annotation.Order;  
import org.springframework.http.codec.ServerCodecConfigurer;  
import org.springframework.web.reactive.result.view.ViewResolver;  
  
import java.util.Collections;  
import java.util.List;  
  
*/\*\*  
 \* 覆盖默认的异常处理  
 \*/*@Configuration  
@EnableConfigurationProperties*({*ServerProperties.class, ResourceProperties.class*})*public class ErrorHandlerConfig *{* private final ServerProperties serverProperties;  
 private final ApplicationContext applicationContext;  
 private final ResourceProperties resourceProperties;  
 private final List*<*ViewResolver*>* viewResolvers;  
 private final ServerCodecConfigurer serverCodecConfigurer;  
  
 public ErrorHandlerConfig*(*ServerProperties serverProperties,  
 ResourceProperties resourceProperties,  
 ObjectProvider*<*List*<*ViewResolver*>>* viewResolversProvider,  
 ServerCodecConfigurer serverCodecConfigurer,  
 ApplicationContext applicationContext*) {* this.serverProperties = serverProperties;  
 this.applicationContext = applicationContext;  
 this.resourceProperties = resourceProperties;  
 this.viewResolvers = viewResolversProvider.getIfAvailable*(*Collections::*emptyList)*;  
 this.serverCodecConfigurer = serverCodecConfigurer;  
 *}* @Bean  
 @Order*(*Ordered.*HIGHEST\_PRECEDENCE)* public ErrorWebExceptionHandler errorWebExceptionHandler*(*ErrorAttributes errorAttributes*) {* JsonExceptionHandler exceptionHandler = new JsonExceptionHandler*(* errorAttributes,  
 this.resourceProperties,  
 this.serverProperties.getError*()*,  
 this.applicationContext*)*;  
 exceptionHandler.setViewResolvers*(*this.viewResolvers*)*;  
 exceptionHandler.setMessageWriters*(*this.serverCodecConfigurer.getWriters*())*;  
 exceptionHandler.setMessageReaders*(*this.serverCodecConfigurer.getReaders*())*;  
 return exceptionHandler;  
 *}  
}*

**JsonExceptionHandler：**

import org.springframework.boot.autoconfigure.web.ErrorProperties;

import org.springframework.boot.autoconfigure.web.ResourceProperties;  
import org.springframework.boot.autoconfigure.web.reactive.error.DefaultErrorWebExceptionHandler;  
import org.springframework.boot.web.reactive.error.ErrorAttributes;  
import org.springframework.context.ApplicationContext;  
import org.springframework.http.HttpStatus;  
import org.springframework.web.reactive.function.server.\*;  
  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \* 自定义异常处理  
 \*  
 \* <p>异常时用JSON代替HTML异常信息<p>  
 \*/*public class JsonExceptionHandler extends DefaultErrorWebExceptionHandler *{* public JsonExceptionHandler*(*ErrorAttributes errorAttributes, ResourceProperties resourceProperties,  
 ErrorProperties errorProperties, ApplicationContext applicationContext*) {* super*(*errorAttributes, resourceProperties, errorProperties, applicationContext*)*;  
 *}  
  
 /\*\*  
 \* 获取异常属性  
 \*/* @Override  
 protected Map*<*String, Object*>* getErrorAttributes*(*ServerRequest request, boolean includeStackTrace*) {* Map*<*String, Object*>* map = new HashMap*<>()*;  
 map.put*(*"success", false*)*;  
 map.put*(*"code", 20005*)*;  
 map.put*(*"message", "网关失败"*)*;  
 map.put*(*"data", null*)*;  
 return map;  
 *}  
  
 /\*\*  
 \* 指定响应处理方法为JSON处理的方法  
 \*  
 \** ***@param*** *errorAttributes  
 \*/* @Override  
 protected RouterFunction*<*ServerResponse*>* getRoutingFunction*(*ErrorAttributes errorAttributes*) {* return RouterFunctions.*route(*RequestPredicates.*all()*, this::renderErrorResponse*)*;  
 *}  
  
 /\*\*  
 \* 根据code获取对应的HttpStatus  
 \*  
 \** ***@param*** *errorAttributes  
 \*/* @Override  
 protected int getHttpStatus*(*Map*<*String, Object*>* errorAttributes*) {* return HttpStatus.*OK*.value*()*;  
 *}  
}*

# 3 权限管理功能（接口）

## 01、权限管理需求描述

**不同角色的用户登录后台管理系统拥有不同的菜单权限与功能权限，权限管理包含三个功能模块：菜单管理、角色管理和用户管理**

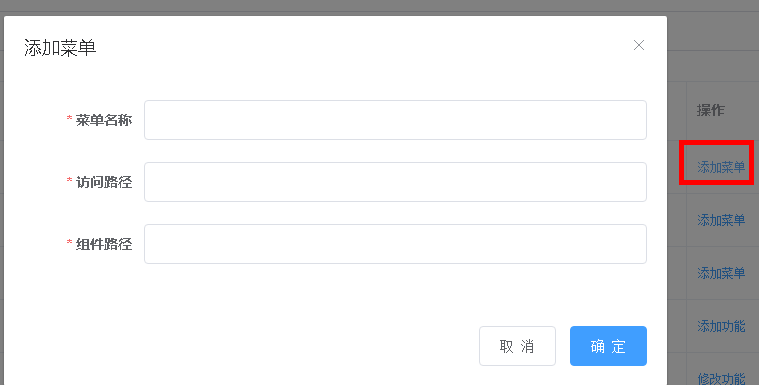
****

### 1、菜单管理

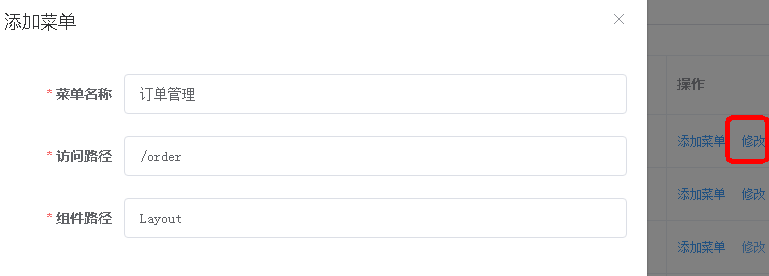
#### （1）菜单列表：使用树形结构显示菜单列表



#### （2）添加菜单：点击添加菜单，弹框进行添加

****

#### （3）修改菜单

****

#### （4）删除菜单

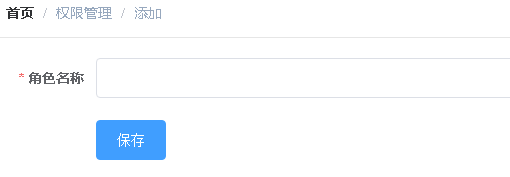
****

### 2、角色管理

#### （1）角色列表：实现角色的条件查询带分页功能

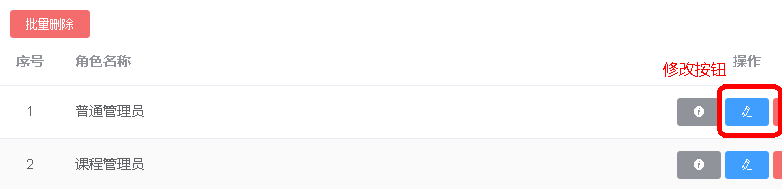


#### （2）角色添加



#### （3）角色修改

**点击修改按钮**

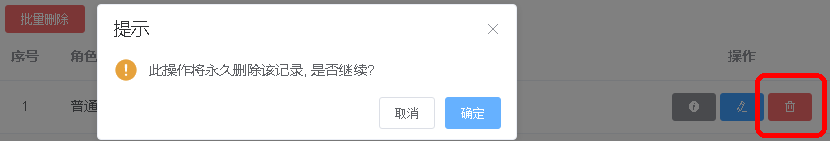


**数据回显，进行修改**



#### （4）角色删除

**普通删除**

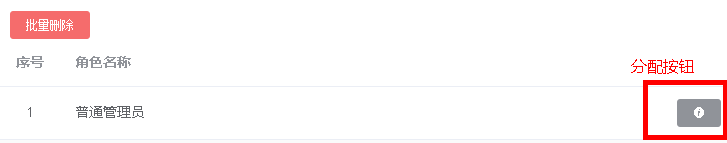


**批量删除**



#### （5）角色分配菜单

**点击分配按钮**

****

**给角色分配菜单**

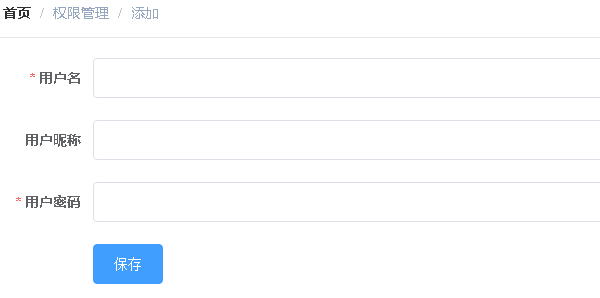
****

### 3、用户管理

#### （1）用户列表



#### （2）用户添加

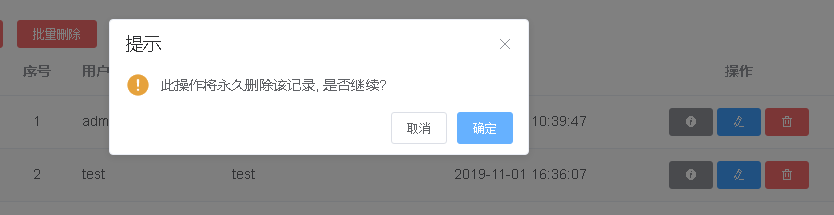


#### （3）用户修改



#### （4）用户删除

**普通删除和批量删除**



#### （5）用户分配角色

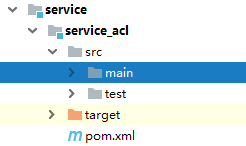


## 02-开发权限管理接口

### 创建权限管理服务



#### 1、在service模块下创建子模块service-acl



#### 2、在service\_acl模块中引入依赖

<dependencies>

<dependency>

<groupId>com.atguigu</groupId>

<artifactId>spring\_security</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

<dependency>

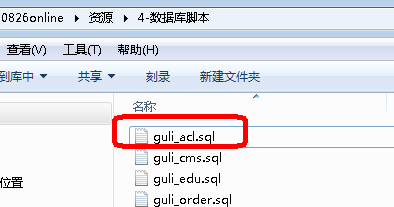
<groupId>com.alibaba</groupId>

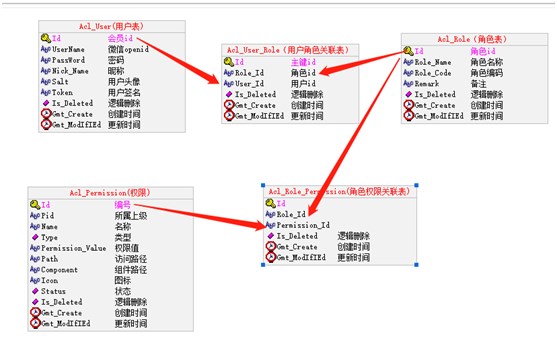
<artifactId>fastjson</artifactId>

</dependency>

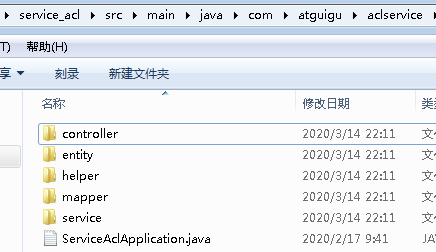
</dependencies>

#### 3、创建权限管理相关的表



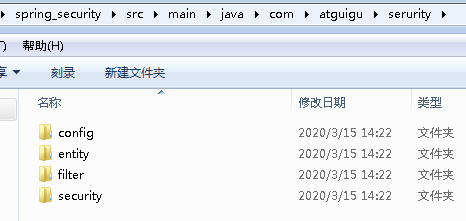


#### 4、复制权限管理接口代码



#### 5、复制整合Spring Security代码

**（1）在common模块下创建子模块spring\_security**

****

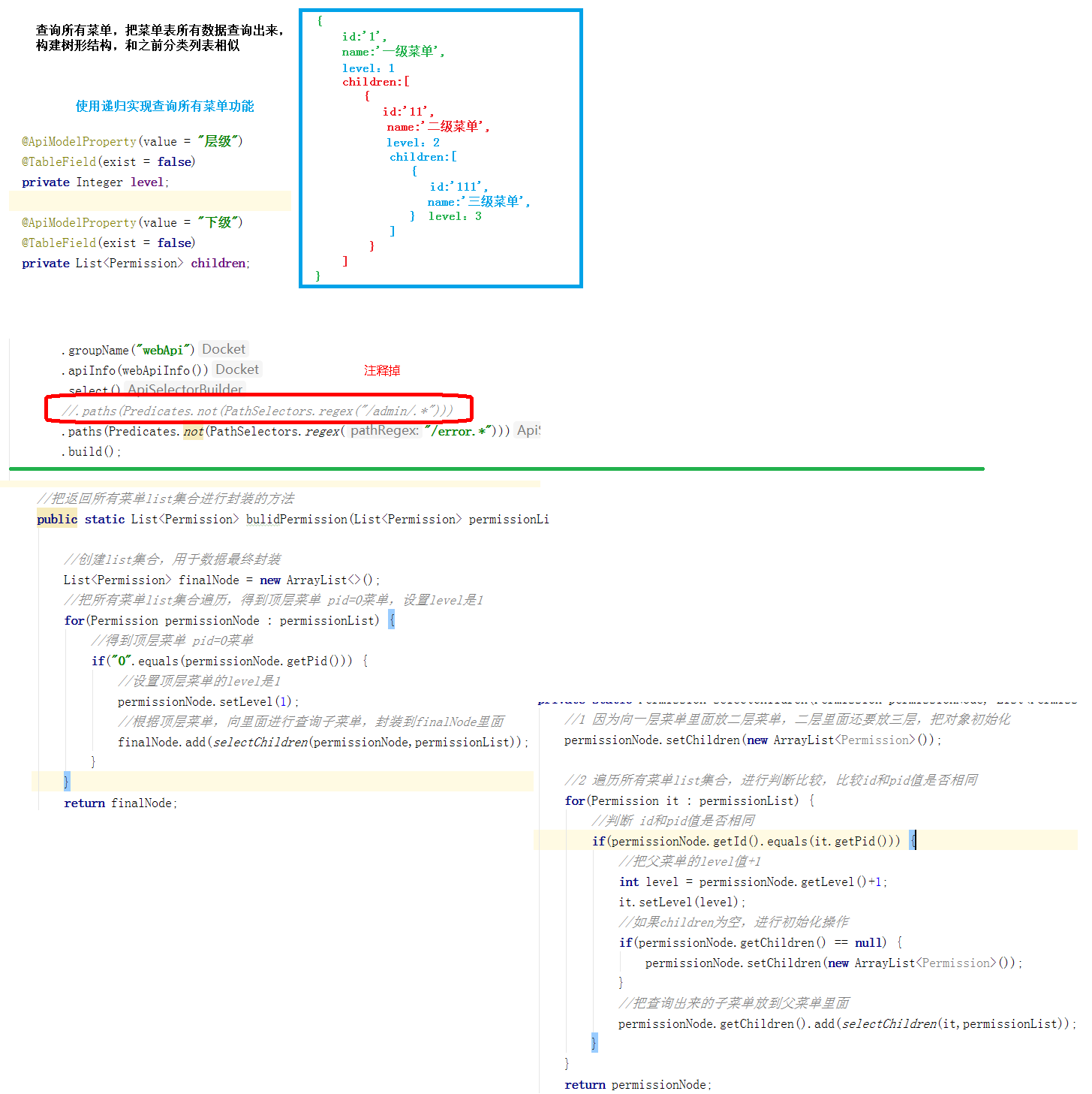
#### 6、编写application.properties配置文件

# 服务端口

server.port=8009  
# 服务名  
spring.application.name=service-acl  
# mysql数据库连接  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/guli?serverTimezone=GMT%2B8  
spring.datasource.username=root  
spring.datasource.password=root  
#返回json的全局时间格式  
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss  
spring.jackson.time-zone=GMT+8  
spring.redis.host=192.168.44.132  
spring.redis.port=6379  
spring.redis.database=0  
spring.redis.timeout=1800000  
spring.redis.lettuce.pool.max-active=20  
spring.redis.lettuce.pool.max-wait=-1  
#最大阻塞等待时间(负数表示没限制)  
spring.redis.lettuce.pool.max-idle=5  
spring.redis.lettuce.pool.min-idle=0  
#最小空闲  
#配置mapper xml文件的路径  
mybatis-plus.mapper-locations=classpath:com/atguigu/aclservice/mapper/xml/\*.xml  
#指定注册中心地址  
eureka.client.service-url.defaultZone=http://127.0.0.1:8761/eureka/  
#eureka服务器上获取的是服务器的ip地址，否则是主机名  
eureka.instance.prefer-ip-address=true  
#mybatis日志  
mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl

### 二、开发权限管理接口

#### 1、递归获取所有菜单



##### （1）controller

@RestController

@RequestMapping*(*"/admin/acl/permission")  
@CrossOrigin  
public class PermissionController {  
 @Autowired  
 private PermissionService permissionService;  
  
 //获取全部菜单  
 @GetMapping  
 public R indexAllPermission() {  
 List<Permission> list = permissionService.queryAllMenu();  
 return R.ok().data("children", list);  
 }  
}

##### （2）service

//获取全部菜单

@Override  
public List*<*Permission*>* queryAllMenu*() {* QueryWrapper*<*Permission*>* wrapper = new QueryWrapper*<>()*;  
 wrapper.orderByDesc*(*"id"*)*;  
 List*<*Permission*>* permissionList = baseMapper.selectList*(*wrapper*)*;  
 List*<*Permission*>* result = bulid*(*permissionList*)*;  
 return result;  
*}*

##### （3）在Permission实体类添加属性

@ApiModelProperty*(*value = "层级"*)*

@TableField*(*exist = false*)*private Integer level;  
@ApiModelProperty*(*value = "下级"*)*@TableField*(*exist = false*)*private List*<*Permission*>* children;  
@ApiModelProperty*(*value = "是否选中"*)*@TableField*(*exist = false*)*private boolean isSelect;

##### （4）编写工具类，根据菜单构建数据

import com.atguigu.aclservice.entity.Permission;

import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* <p>  
 \* 根据权限数据构建菜单数据  
 \* </p>  
 \*/*public class PermissionHelper *{  
 /\*\*  
 \* 使用递归方法建菜单  
 \*  
 \** ***@param*** *treeNodes  
 \** ***@return*** *\*/* public static List*<*Permission*>* bulid*(*List*<*Permission*>* treeNodes*) {* List*<*Permission*>* trees = new ArrayList*<>()*;  
 for *(*Permission treeNode : treeNodes*) {* if *(*"0".equals*(*treeNode.getPid*())) {*//查询一级菜单 入口  
 treeNode.setLevel*(*1*)*;  
 trees.add*(findChildren(*treeNode, treeNodes*))*;  
 *}  
 }* return trees;  
 *}  
  
 /\*\*  
 \* 递归查找子节点  
 \*  
 \** ***@param*** *treeNodes  
 \** ***@return*** *\*/* public static Permission findChildren*(*Permission treeNode, List*<*Permission*>* treeNodes*) {* treeNode.setChildren*(*new ArrayList*<*Permission*>())*;  
 for *(*Permission it : treeNodes*) {* if *(*treeNode.getId*()*.equals*(*it.getPid*())) {* int level = treeNode.getLevel*()* + 1;  
 it.setLevel*(*level*)*;  
 if *(*treeNode.getChildren*()* == null*) {* treeNode.setChildren*(*new ArrayList*<>())*;  
 *}* treeNode.getChildren*()*.add*(findChildren(*it, treeNodes*))*;  
 *}  
 }* return treeNode;  
 *}  
}*

#### 2、递归删除菜单



##### （1）controller

@ApiOperation*(*value = "递归删除菜单"*)*

@DeleteMapping*(*"remove/{id}"*)*public R remove*(*@PathVariable String id*) {* permissionService.removeChildById*(*id*)*;  
 return R.ok();  
}

##### （2）service

//递归删除菜单  
@Override

public void removeChildById*(*String id*) {* List*<*String*>* idList = new ArrayList*<>()*;  
 this.selectChildListById*(*id, idList);  
 //把根据节点id放到list中  
 idList.add*(*id*)*;  
 baseMapper.deleteBatchIds*(*idList*)*;  
*}  
  
/\*\*  
 \* 递归获取子节点  
 \*  
 \** ***@param*** *id  
 \** ***@param*** *idList  
 \*/*private void selectChildListById*(*String id, List*<*String*>* idList*) {* List*<*Permission*>* childList = baseMapper.selectList*(*new QueryWrapper*<*Permission*>()*.eq*(*"pid", id*)*.select*(*"id"*))*;  
 childList.stream*()*.forEach*(*item -> *{* idList.add*(*item.getId*())*;  
 this.selectChildListById*(*item.getId(), idList);  
 *})*;  
*}*

#### 3、给角色分配权限



##### （1）controller

@ApiOperation*(*value = "给角色分配权限"*)*

@PostMapping*(*"/doAssign"*)*public R doAssign*(*String roleId, String*[]* permissionId*) {* permissionService.saveRolePermissionRealtionShip*(*roleId, permissionId*)*;  
 return R.ok();  
}

##### （2）service

//给角色分配权限

@Override  
public void saveRolePermissionRealtionShip(String roleId, String[] permissionIds) *{* rolePermissionService.remove*(*new QueryWrapper*<*RolePermission*>()*.eq*(*"role\_id", roleId*))*;  
 List*<*RolePermission*>* rolePermissionList = new ArrayList*<>()*;  
 for *(*String permissionId : permissionIds*) {* if *(*StringUtils.*isEmpty(*permissionId*))* continue;  
 RolePermission rolePermission = new RolePermission*()*;  
 rolePermission.setRoleId*(*roleId*)*;  
 rolePermission.setPermissionId*(*permissionId*)*;  
 rolePermissionList.add*(*rolePermission*)*;  
 *}* rolePermissionService.saveBatch*(*rolePermissionList*)*;  
*}*

# 04- 整合Spring Security

## 一、Spring Security介绍

### 1、框架介绍

Spring 是一个非常流行和成功的 Java 应用开发框架。Spring Security 基于 Spring 框架，提供了一套 Web 应用安全性的完整解决方案。一般来说，Web 应用的安全性包括**用户认证（Authentication）和用户授权（Authorization）**两个部分。

（1）用户认证指的是：验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。

（2）用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个文件来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。

**Spring Security其实就是用filter，多请求的路径进行过滤。**

（1）如果是基于Session，那么Spring-security会对cookie里的sessionid进行解析，找到服务器存储的sesion信息，然后判断当前用户是否符合请求的要求。

（2）如果是token，则是解析出token，然后将当前请求加入到Spring-security管理的权限信息中去

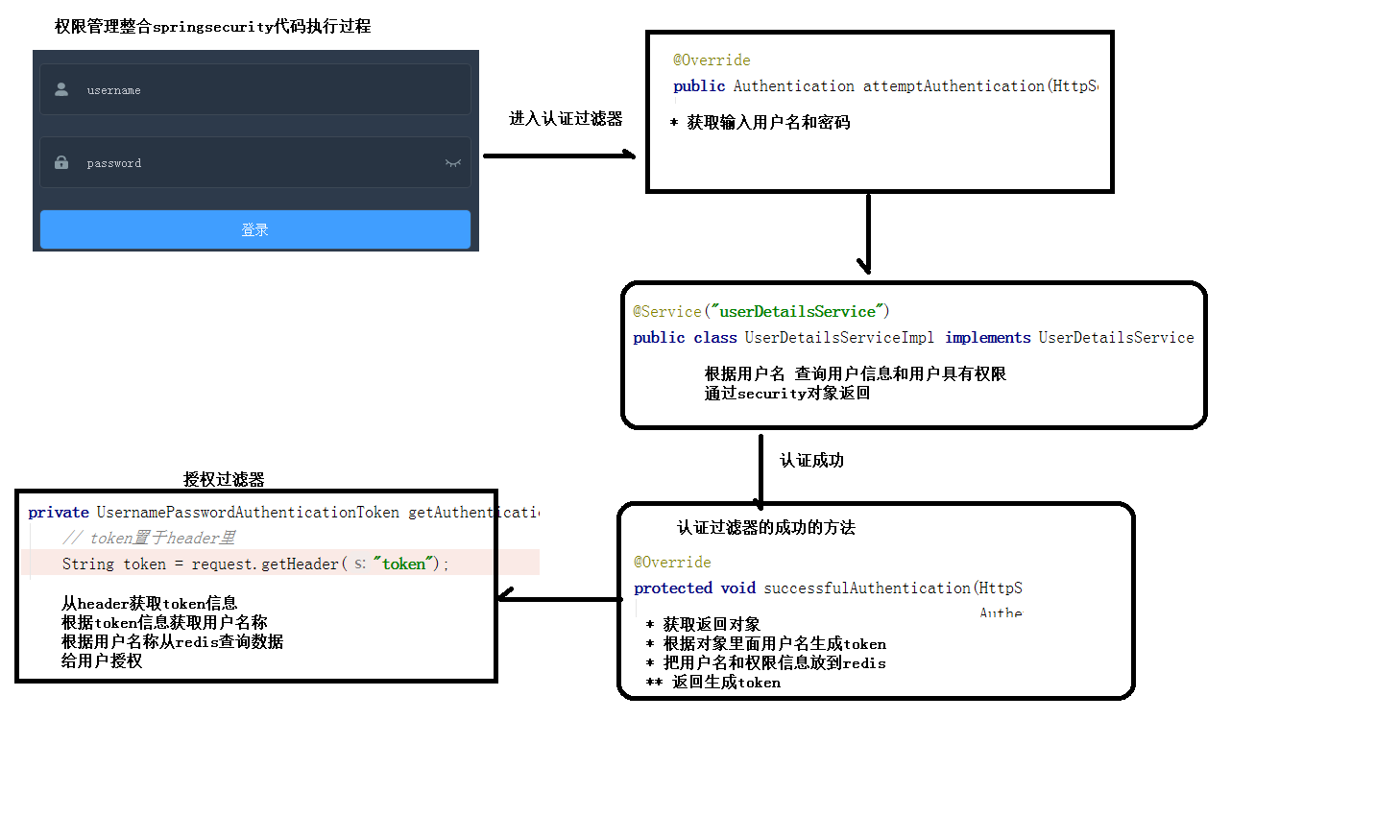
### 2、认证与授权实现思路

如果系统的模块众多，每个模块都需要就行授权与认证，所以我们选择基于token的形式进行授权与认证，用户根据用户名密码认证成功，然后获取当前用户角色的一系列权限值，并以用户名为key，权限列表为value的形式存入redis缓存中，根据用户名相关信息生成token返回，浏览器将token记录到cookie中，每次调用api接口都默认将token携带到header请求头中，Spring-security解析header头获取token信息，解析token获取当前用户名，根据用户名就可以从redis中获取权限列表，这样Spring-security就能够判断当前请求是否有权限访问

## 二、整合Spring Security

### 课堂笔记

###### 整合Spring Security权限框架（框架介绍）

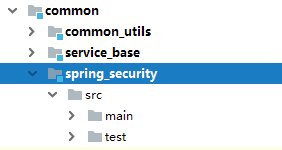


|  |
| --- |
| 1、Spring Security主要包含有两部分：用户认证 和 用户授权  （1）用户认证：  进入用户登录时候，输入用户名和密码，查询数据库，输入用户名和密码是否正确，如果正确的话，认证成功了  （2）用户授权：  登录了系统，登录用户可能是不同的角色，比如现在登录的用户是管理员，管理员操作所有功能，比如登录用户普通用户，操作功能肯定比管理员少很多  2、Spring Security本质上就是过滤器Filter，对请求进行过滤  3、**认证与授权实现思路** |

###### 整合Spring Security权限框架（代码整合）

|  |
| --- |
| **1、创建模块**    **2、在spring\_security模块引入相关依赖**  <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu</**groupId**>  <**artifactId**>common\_utils</**artifactId**>  <**version**>0.0.1-SNAPSHOT</**version**>  </**dependency**>  *<!-- Spring Security依赖 -->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-security</**artifactId**>  </**dependency**>  <**dependency**>  <**groupId**>io.jsonwebtoken</**groupId**>  <**artifactId**>jjwt</**artifactId**>  </**dependency**> </**dependencies**>  **3、到源码里面复制整合代码**    **4、复制工具类到common\_utils**      **5、整合代码结构**    **6、在service\_acl模块中引入spring\_security依赖**    **7、创建查询登录和用户权限类** |

### 1、在common下创建spring\_security模块



### 2、在spring\_security引入相关依赖

<dependencies>

<dependency>

<groupId>com.atguigu</groupId>

<artifactId>common\_utils</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

<!-- Spring Security依赖 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

</dependency>

</dependencies>

### 3、在service\_acl引入spring\_security依赖

<dependency>

<groupId>com.atguigu</groupId>

<artifactId>spring\_security</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

**0、代码结构说明：**

****

### 4、创建spring security核心配置类

****

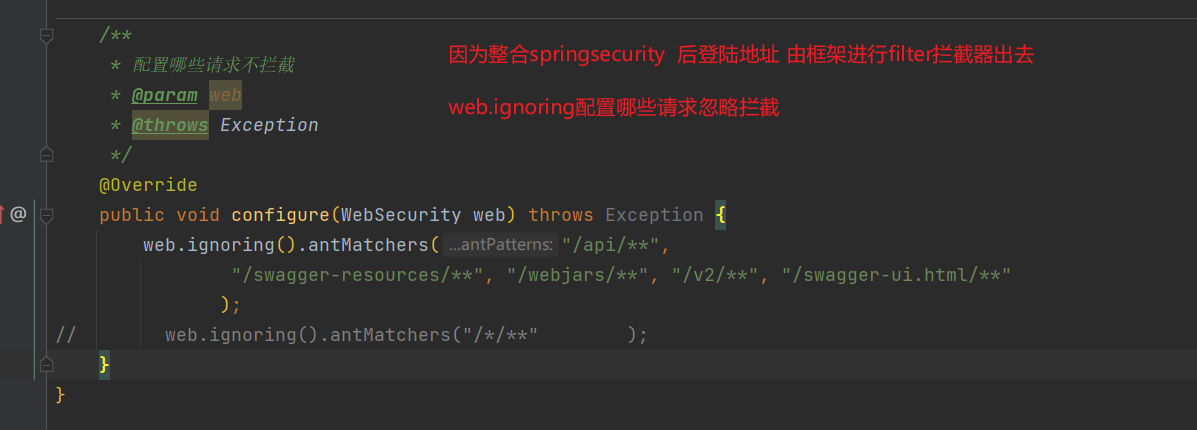
Spring Security的核心配置就是继承WebSecurityConfigurerAdapter并注解@EnableWebSecurity的配置。

这个配置指明了用户名密码的处理方式、请求路径的开合、登录登出控制等和安全相关的配置

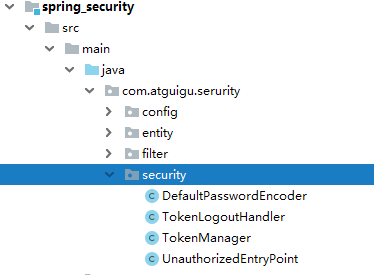
import com.atguigu.serurity.filter.TokenAuthenticationFilter;

import com.atguigu.serurity.filter.TokenLoginFilter;  
import com.atguigu.serurity.security.DefaultPasswordEncoder;  
import com.atguigu.serurity.security.TokenLogoutHandler;  
import com.atguigu.serurity.security.TokenManager;  
import com.atguigu.serurity.security.UnauthorizedEntryPoint;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
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.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.core.userdetails.UserDetailsService;  
  
*/\*\*  
 \* <p>  
 \* Security配置类  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-18  
 \*/*@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity*(*prePostEnabled = true*)*public class TokenWebSecurityConfig extends WebSecurityConfigurerAdapter *{* private UserDetailsService userDetailsService;  
 private TokenManager tokenManager;  
 private DefaultPasswordEncoder defaultPasswordEncoder;  
 private RedisTemplate redisTemplate;  
  
 @Autowired  
 public TokenWebSecurityConfig*(*UserDetailsService userDetailsService, DefaultPasswordEncoder defaultPasswordEncoder,  
 TokenManager tokenManager, RedisTemplate redisTemplate*) {* this.userDetailsService = userDetailsService;  
 this.defaultPasswordEncoder = defaultPasswordEncoder;  
 this.tokenManager = tokenManager;  
 this.redisTemplate = redisTemplate;  
 *}  
  
 /\*\*  
 \* 配置设置  
 \** ***@param*** *http  
 \** ***@throws*** *Exception  
 \*/* @Override  
 protected void configure*(*HttpSecurity http*)* throws Exception *{* http.exceptionHandling*()* .authenticationEntryPoint*(*new UnauthorizedEntryPoint*())* .and*()*.csrf*()*.disable*()* .authorizeRequests*()* .anyRequest*()*.authenticated*()* .and*()*.logout*()*.logoutUrl*(*"/admin/acl/index/logout"*)* .addLogoutHandler*(*new TokenLogoutHandler*(*tokenManager,redisTemplate*))*.and*()* .addFilter*(*new TokenLoginFilter*(*authenticationManager*()*, tokenManager, redisTemplate*))* .addFilter*(*new TokenAuthenticationFilter*(*authenticationManager*()*, tokenManager, redisTemplate*))*.httpBasic*()*;  
 *}  
  
 /\*\*  
 \* 密码处理  
 \** ***@param*** *auth  
 \** ***@throws*** *Exception  
 \*/* @Override  
 public void configure*(*AuthenticationManagerBuilder auth*)* throws Exception *{* auth.userDetailsService*(*userDetailsService*)*.passwordEncoder*(*defaultPasswordEncoder*)*;  
 *}  
  
 /\*\*  
 \* 配置哪些请求不拦截  
 \** ***@param*** *web  
 \** ***@throws*** *Exception  
 \*/* @Override  
 public void configure*(*WebSecurity web*)* throws Exception *{* web.ignoring*()*.antMatchers*(*"/api/\*\*",  
 "/swagger-resources/\*\*", "/webjars/\*\*", "/v2/\*\*", "/swagger-ui.html/\*\*"  
 *)*;  
// web.ignoring().antMatchers("/\*/\*\*" );  
 *}*

*}*



### 5、创建认证授权相关的工具类

****

#### （1）DefaultPasswordEncoder：密码处理的方法

import com.atguigu.commonutils.MD5;

import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* <p>  
 \* t密码的处理方法类型  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*@Component  
public class DefaultPasswordEncoder implements PasswordEncoder *{* public DefaultPasswordEncoder*() {* this*(*-1*)*;  
 *}  
  
 /\*\*  
 \** ***@param*** *strength  
 \* the log rounds to use, between 4 and 31  
 \*/* public DefaultPasswordEncoder*(*int strength*) {  
  
 }* public String encode*(*CharSequence rawPassword*) {* return MD5.*encrypt(*rawPassword.toString*())*;  
 *}* public boolean matches*(*CharSequence rawPassword, String encodedPassword*) {* return encodedPassword.equals*(*MD5.*encrypt(*rawPassword.toString*()))*;  
 *}  
}*

#### （2）TokenManager：token操作的工具类

import io.jsonwebtoken.CompressionCodecs;

import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
  
*/\*\*  
 \* <p>  
 \* token管理  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*@Component  
public class TokenManager *{* private long tokenExpiration = 24\*60\*60\*1000;  
 private String tokenSignKey = "123456";  
  
 public String createToken*(*String username*) {* String token = Jwts.*builder()*.setSubject*(*username*)* .setExpiration*(*new Date*(*System.*currentTimeMillis()* + tokenExpiration*))* .signWith*(*SignatureAlgorithm.*HS512*, tokenSignKey*)*.compressWith*(*CompressionCodecs.*GZIP)*.compact*()*;  
 return token;  
 *}* public String getUserFromToken*(*String token*) {* String user = Jwts.*parser()*.setSigningKey*(*tokenSignKey*)*.parseClaimsJws*(*token*)*.getBody*()*.getSubject*()*;  
 return user;  
 *}* public void removeToken*(*String token*) {* //jwttoken无需删除，客户端扔掉即可。  
 *}  
  
}*

#### （3）TokenLogoutHandler：退出实现

import com.atguigu.commonutils.R;

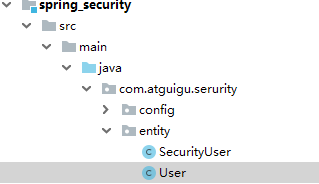
import com.atguigu.commonutils.ResponseUtil;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.web.authentication.logout.LogoutHandler;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
*/\*\*  
 \* <p>  
 \* 登出业务逻辑类  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*public class TokenLogoutHandler implements LogoutHandler *{* private TokenManager tokenManager;  
 private RedisTemplate redisTemplate;  
  
 public TokenLogoutHandler*(*TokenManager tokenManager, RedisTemplate redisTemplate*) {* this.tokenManager = tokenManager;  
 this.redisTemplate = redisTemplate;  
 *}* @Override  
 public void logout*(*HttpServletRequest request, HttpServletResponse response, Authentication authentication*) {* String token = request.getHeader*(*"token"*)*;  
 if *(*token != null*) {* tokenManager.removeToken*(*token*)*;  
  
 //清空当前用户缓存中的权限数据  
 String userName = tokenManager.getUserFromToken*(*token*)*;  
 redisTemplate.delete*(*userName*)*;  
 *}* ResponseUtil.*out(*response, R.*ok())*;  
 *}  
  
}*

#### （4）UnauthorizedEntryPoint：未授权统一处理

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
*/\*\*  
 \* <p>  
 \* 未授权的统一处理方式  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*public class UnauthorizedEntryPoint implements AuthenticationEntryPoint *{* @Override  
 public void commence*(*HttpServletRequest request, HttpServletResponse response,  
 AuthenticationException authException*)* throws IOException, ServletException *{* ResponseUtil.*out(*response, R.*error())*;  
 *}  
}*

### 6、创建认证授权实体类

****

**（1）SecutityUser**

import lombok.Data;

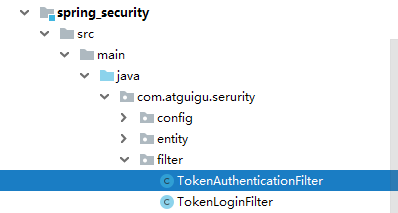
import lombok.extern.slf4j.Slf4j;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.util.StringUtils;  
  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.List;  
  
*/\*\*  
 \* <p>  
 \* 安全认证用户详情信息  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*@Data  
@Slf4j  
public class SecurityUser **implements UserDetails** *{* //当前登录用户  
 private transient User currentUserInfo;  
  
 //当前权限  
 private List*<*String*>* permissionValueList;  
  
 public SecurityUser*() {  
 }* public SecurityUser*(*User user*) {* if *(*user != null*) {* this.currentUserInfo = user;  
 *}  
 }* @Override  
 public Collection*<*? extends GrantedAuthority*>* getAuthorities*() {* Collection*<*GrantedAuthority*>* authorities = new ArrayList*<>()*;  
 for*(*String permissionValue : permissionValueList*) {* if*(*StringUtils.*isEmpty(*permissionValue*))* continue;  
 SimpleGrantedAuthority authority = new SimpleGrantedAuthority*(*permissionValue*)*;  
 authorities.add*(*authority*)*;  
 *}* return authorities;  
 *}* @Override  
 public String getPassword*() {* return currentUserInfo.getPassword*()*;  
 *}* @Override  
 public String getUsername*() {* return currentUserInfo.getUsername*()*;  
 *}* @Override  
 public boolean isAccountNonExpired*() {* return true;  
 *}* @Override  
 public boolean isAccountNonLocked*() {* return true;  
 *}* @Override  
 public boolean isCredentialsNonExpired*() {* return true;  
 *}* @Override  
 public boolean isEnabled*() {* return true;  
 *}  
}*

**（2）User**

@Data

@ApiModel*(*description = "用户实体类"*)*public class User implements Serializable *{* private static final long *serialVersionUID* = 1L;  
  
 @ApiModelProperty*(*value = "微信openid"*)* private String username;  
  
 @ApiModelProperty*(*value = "密码"*)* private String password;  
  
 @ApiModelProperty*(*value = "昵称"*)* private String nickName;  
  
 @ApiModelProperty*(*value = "用户头像"*)* private String salt;  
  
 @ApiModelProperty*(*value = "用户签名"*)* private String token;  
  
*}*

### 7、创建认证和授权的filter

****

#### （1）TokenLoginFilter：认证的filter

import com.atguigu.commonutils.R;

import com.atguigu.commonutils.ResponseUtil;  
import com.atguigu.serurity.entity.SecurityUser;  
import com.atguigu.serurity.entity.User;  
import com.atguigu.serurity.security.TokenManager;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.AuthenticationException;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;  
  
import javax.servlet.FilterChain;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.ArrayList;  
  
*/\*\*  
 \* <p>  
 \* 登录过滤器，继承UsernamePasswordAuthenticationFilter，对用户名密码进行登录校验  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*public class TokenLoginFilter extends UsernamePasswordAuthenticationFilter *{* private AuthenticationManager authenticationManager;  
 private TokenManager tokenManager;  
 private RedisTemplate redisTemplate;  
  
 public TokenLoginFilter*(*AuthenticationManager authenticationManager, TokenManager tokenManager, RedisTemplate redisTemplate*) {* this.authenticationManager = authenticationManager;  
 this.tokenManager = tokenManager;  
 this.redisTemplate = redisTemplate;  
 this.setPostOnly*(*false*)*;  
 this.setRequiresAuthenticationRequestMatcher*(*new AntPathRequestMatcher*(*"/admin/acl/login","POST"*))*;  
 *}* @Override  
 public Authentication attemptAuthentication*(*HttpServletRequest req, HttpServletResponse res*)* throws AuthenticationException *{* try *{* User user = new ObjectMapper*()*.readValue*(*req.getInputStream*()*, User.class*)*;  
  
 return authenticationManager.authenticate*(*new UsernamePasswordAuthenticationToken*(*user.getUsername*()*, user.getPassword*()*, new ArrayList*<>()))*;  
 *}* catch *(*IOException e*) {* throw new RuntimeException*(*e*)*;  
 *}  
  
 }  
  
 /\*\*  
 \* 登录成功  
 \** ***@param*** *req  
 \** ***@param*** *res  
 \** ***@param*** *chain  
 \** ***@param*** *auth  
 \** ***@throws*** *IOException  
 \** ***@throws*** *ServletException  
 \*/* @Override  
 protected void successfulAuthentication*(*HttpServletRequest req, HttpServletResponse res, FilterChain chain,  
 Authentication auth*)* throws IOException, ServletException *{* SecurityUser user = *(*SecurityUser*)* auth.getPrincipal*()*;  
 String token = tokenManager.createToken*(*user.getCurrentUserInfo*()*.getUsername*())*;  
 redisTemplate.opsForValue*()*.set*(*user.getCurrentUserInfo*()*.getUsername*()*, user.getPermissionValueList*())*;  
  
 ResponseUtil.*out(*res, R.*ok()*.data*(*"token", token*))*;  
 *}  
  
 /\*\*  
 \* 登录失败  
 \** ***@param*** *request  
 \** ***@param*** *response  
 \** ***@param*** *e  
 \** ***@throws*** *IOException  
 \** ***@throws*** *ServletException  
 \*/* @Override  
 protected void unsuccessfulAuthentication*(*HttpServletRequest request, HttpServletResponse response,  
 AuthenticationException e*)* throws IOException, ServletException *{* ResponseUtil.*out(*response, R.*error())*;  
 *}  
}*

#### （2）TokenAuthenticationFilter：授权filter

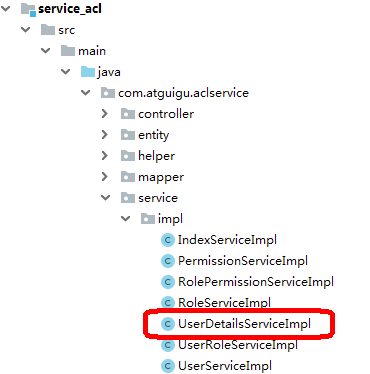
import com.atguigu.commonutils.R;

import com.atguigu.commonutils.ResponseUtil;  
import com.atguigu.serurity.security.TokenManager;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;  
import org.springframework.util.StringUtils;  
  
import javax.servlet.FilterChain;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.List;  
  
*/\*\*  
 \* <p>  
 \* 访问过滤器  
 \* </p>  
 \*  
 \** ***@author*** *qy  
 \** ***@since*** *2019-11-08  
 \*/*public class TokenAuthenticationFilter extends BasicAuthenticationFilter *{* private TokenManager tokenManager;  
 private RedisTemplate redisTemplate;  
  
 public TokenAuthenticationFilter*(*AuthenticationManager authManager, TokenManager tokenManager,RedisTemplate redisTemplate*) {* super*(*authManager*)*;  
 this.tokenManager = tokenManager;  
 this.redisTemplate = redisTemplate;  
 *}* @Override  
 protected void doFilterInternal*(*HttpServletRequest req, HttpServletResponse res, FilterChain chain*)* throws IOException, ServletException *{* logger.info*(*"================="+req.getRequestURI*())*;  
 if*(*req.getRequestURI*()*.indexOf*(*"admin"*)* == -1*) {* chain.doFilter*(*req, res*)*;  
 return;  
 *}* UsernamePasswordAuthenticationToken authentication = null;  
 try *{* authentication = getAuthentication*(*req*)*;  
 *}* catch *(*Exception e*) {* ResponseUtil.*out(*res, R.*error())*;  
 *}* if *(*authentication != null*) {* SecurityContextHolder.*getContext()*.setAuthentication*(*authentication*)*;  
 *}* else *{* ResponseUtil.*out(*res, R.*error())*;  
 *}* chain.doFilter*(*req, res*)*;  
 *}* private UsernamePasswordAuthenticationToken getAuthentication*(*HttpServletRequest request*) {* // token置于header里  
 String token = request.getHeader*(*"token"*)*;  
 if *(*token != null && !"".equals*(*token.trim*())) {* String userName = tokenManager.getUserFromToken*(*token*)*;  
  
 List*<*String*>* permissionValueList = *(*List*<*String*>)* redisTemplate.opsForValue*()*.get*(*userName*)*;  
 Collection*<*GrantedAuthority*>* authorities = new ArrayList*<>()*;  
 for*(*String permissionValue : permissionValueList*) {* if*(*StringUtils.*isEmpty(*permissionValue*))* continue;  
 SimpleGrantedAuthority authority = new SimpleGrantedAuthority*(*permissionValue*)*;  
 authorities.add*(*authority*)*;  
 *}* if *(*!StringUtils.*isEmpty(*userName*)) {* return new UsernamePasswordAuthenticationToken*(*userName, token, authorities*)*;  
 *}* return null;  
 *}* return null;  
 *}  
}*

# 05 创建查询用户类和前端对接

## 一、创建自定义查询用户类

**（1）在service\_acl模块创建，因为其他模板不会用到**

****

@Service*(*"userDetailsService"*)*

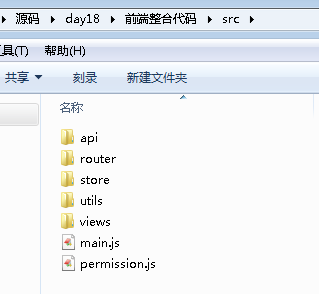
public class UserDetailsServiceImpl **implements UserDetailsService** *{* @Autowired  
 private UserService userService;  
  
 @Autowired  
 private PermissionService permissionService;  
  
 */\*\*\*  
 \* 根据账号获取用户信息  
 \** ***@param*** *username:  
 \** ***@return:*** *org.springframework.security.core.userdetails.UserDetails  
 \*/* @Override  
 public UserDetails loadUserByUsername*(*String username*)* throws UsernameNotFoundException *{* // 从数据库中取出用户信息  
 User user = userService.selectByUsername*(*username*)*;  
  
 // 判断用户是否存在  
 if *(*null == user*){* //throw new UsernameNotFoundException("用户名不存在！");  
 *}* // 返回UserDetails实现类  
 com.atguigu.serurity.entity.User curUser = new com.atguigu.serurity.entity.User*()*;  
 BeanUtils.*copyProperties(*user,curUser*)*;  
  
 List*<*String*>* authorities = permissionService.selectPermissionValueByUserId*(*user.getId*())*;  
 SecurityUser securityUser = new SecurityUser*(*curUser*)*;  
 securityUser.setPermissionValueList*(*authorities*)*;  
 return securityUser;  
 *}  
  
}*

## 二、后端接口和前端页面对接

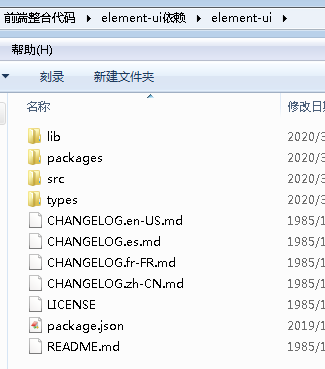
**1、在前端项目中下载依赖**

npm install --save vuex-persistedstate

**2、替换相关文件**

****

**3、在node\_modules文件夹中替换element-ui依赖**

****