springboot

# 概述

Springboot 相对于Spring , SpringMVC 更为轻量级的web开发框架，另外Springboot集成了tomcat 这样我们的项目连同web容器都可以直接打成jar包 ，当我们要部署的时候就可以直接启动jar包即可

Springboot在SpringCloud起着举足轻重的地位

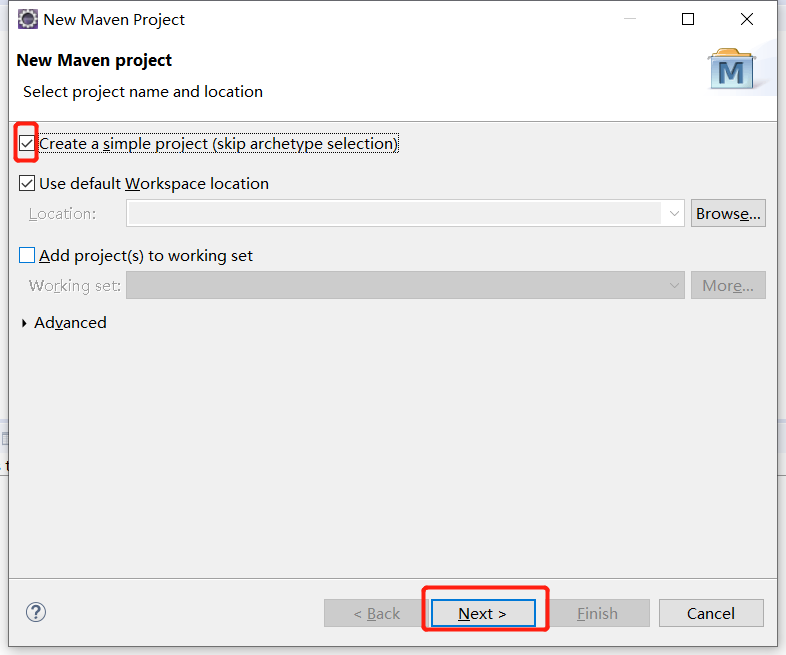
优点：易上手，开箱即用(简化了spring的配置，内嵌了web容器)

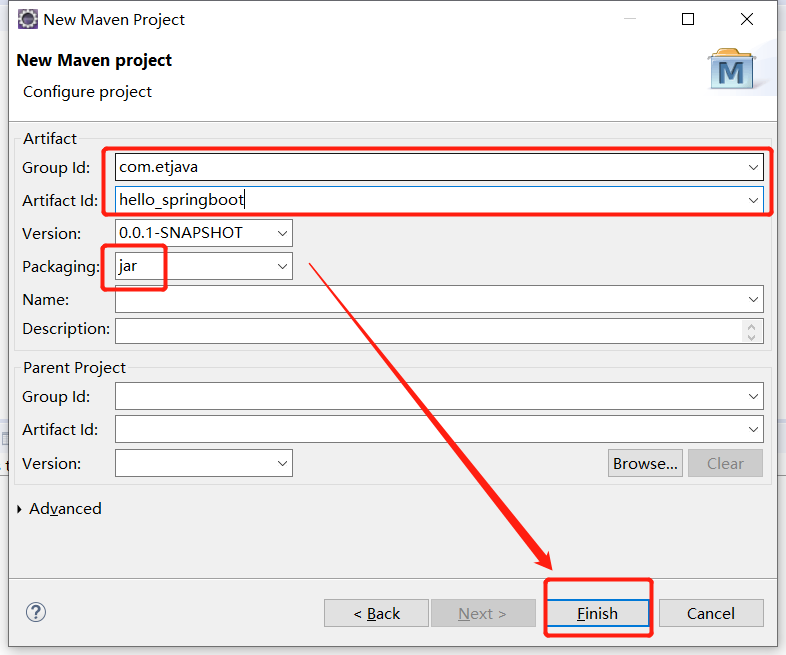
# 第一个springboot程序

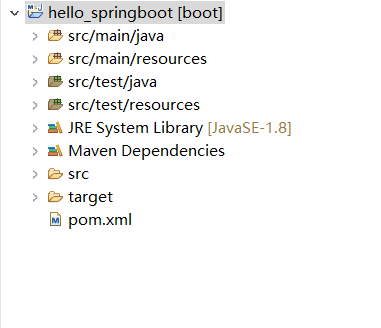
搭建步骤

## 创建mave工程

- maven project







## 添加springboot依赖包

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.7.RELEASE</version>  <relativePath />  </parent>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

## 业务处理类

|  |
| --- |
| @RestController  **public** **class** TestController {    @RequestMapping("/hello")  **public** String hello() {  **return** "hello springboot by - "+**new** Date().getTime();  }  } |

## 启动springboot项目

## 启动方式1 根目录下新建启动类 [推荐]

这样做的好处是springboot可以自动扫描平级以及子包中的带有各种注解的类进行实例化

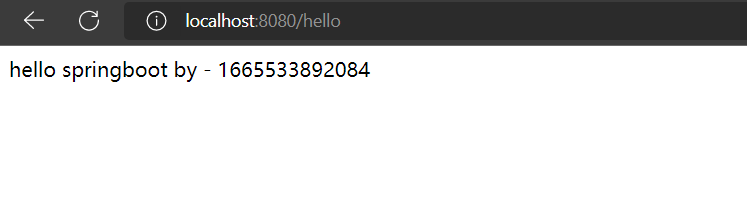
|  |
| --- |
| @SpringBootApplication  **public** **class** App {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(App.**class**, args);  }  } |

## 启动方式2

当前包中添加@EnableAutoConfiguration注解标记当前为spring的启动类 然后编写main方法即可 这样只能扫描到本包及本包下子包的内容 跨包无法扫描

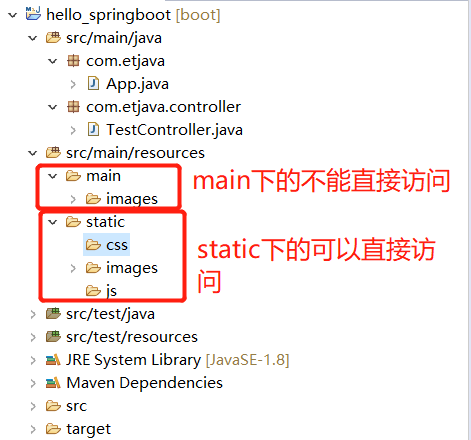
|  |
| --- |
|  |

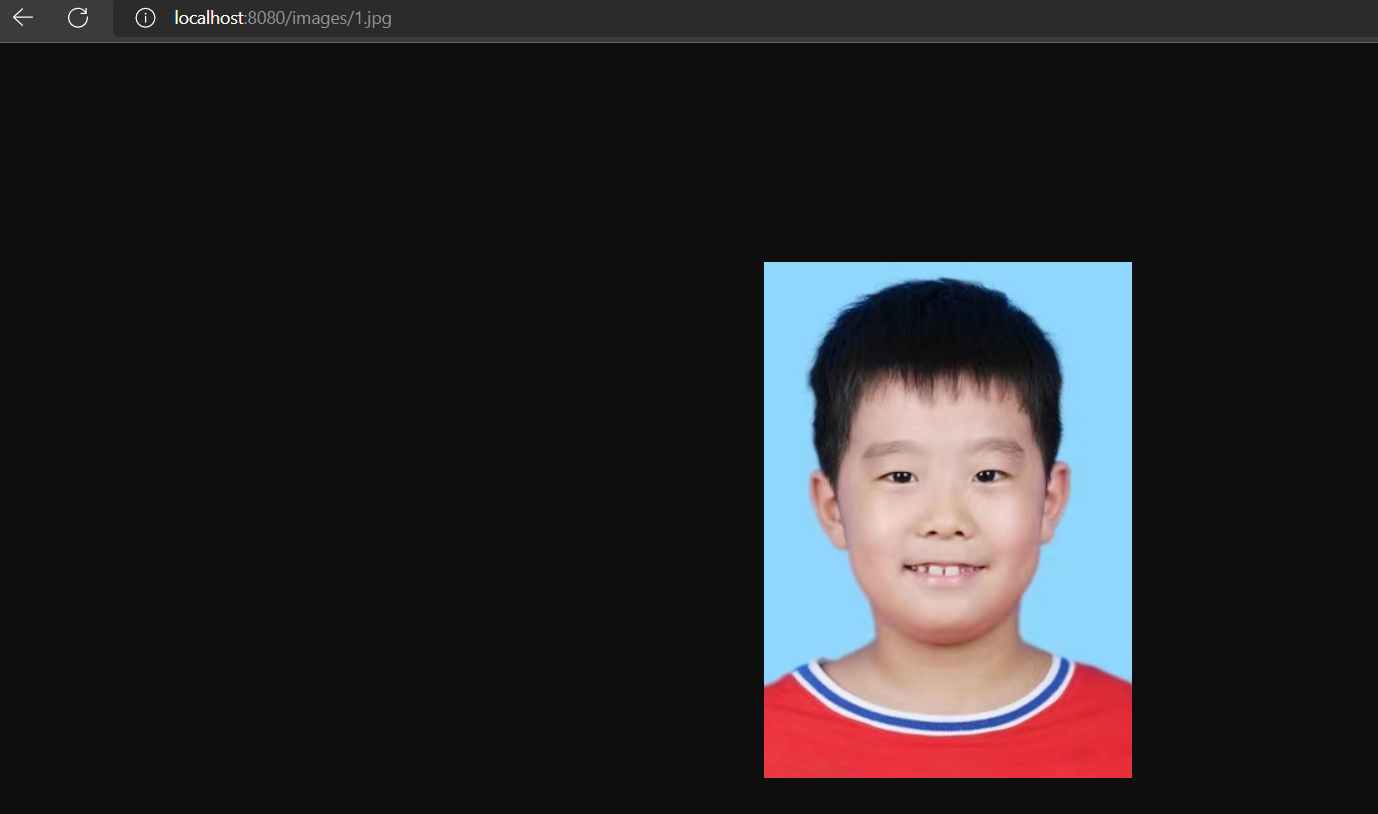
## 测试

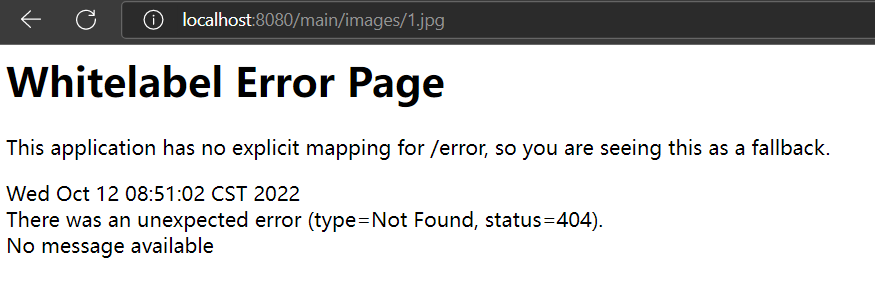


# 静态资源访问

springboot默认会扫描src/main/resource下的static目录 并且该目录可以直接访问，我们可以在该目录下存放css,js,img等一些可以直接访问的静态资源文件，另外springboot默认会加载src/main/resource下的application.properties文件 该文件是springboot的配置文件 通常我们使用yml格式







# 全局异常捕获

## controller中模拟发生异常

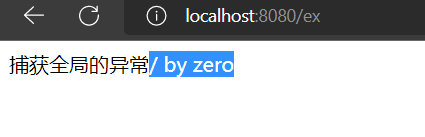


## 定义全局捕获异常类

首先通过ControllerAdvice定义切面

然后在通过@ExceptionHandler配置需要捕获的异常 如果是直接返回JSON格式的数据 添加@ResponseBody注解即可

|  |
| --- |
| /\*\*  \* 测试全局捕获异常  \* 该类相当于动态代理或AOP配置全局捕获异常 只是这里使用注解形式 更为方便  \*  \* 这里的@ControllerAdvice可以理解为定义该类为AOP切面  \* **@author** etjav  \*  \*/  @ControllerAdvice  **public** **class** GloableException {  /\*\*  \* 当有controller发生异常时 这里可以进行捕获处理 例如栈溢出或内存不足等异常  \* **@param** e  \* **@return**  \*/  @ResponseBody  @ExceptionHandler({RuntimeException.**class**,IOException.**class**})  **public** String runtimeException(RuntimeException e) {  **return** "捕获全局的异常"+e.getMessage();  }  } |



# 集成thymeleaf

## 概述

thymeleaf 是springboot官方推荐的前端模板 目前使用最多的就是thymeleaf 和 freeMaker ,因为JSP是servlet编写的 相对于thymeleaf较重，thymeleaf也可以动态取值等操作

## application.properties中配置thymeleaf相关信息

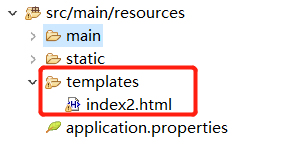
|  |
| --- |
| spring.thymeleaf.suffix=.html  spring.thymeleaf.mode=HTML5  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.content-type=text/html |

## 添加thymeleaf相关依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency> |

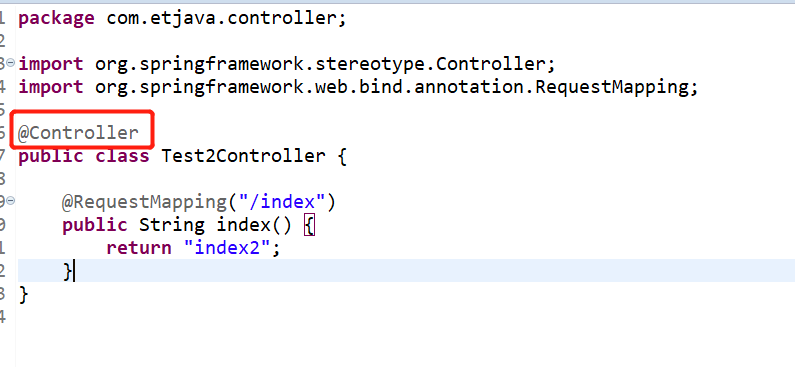
## 创建templates目录

因为springboot会扫描这个目录 该目录是用来存放静态页面文件的

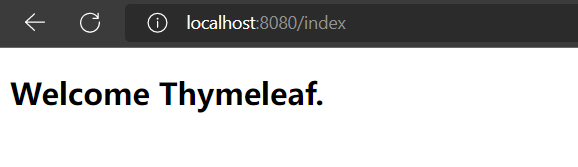


## 编写测试controller

注意 该controller不能返回json格式数据 而是返回页面 因此需要使用@Controller主机



## 开测

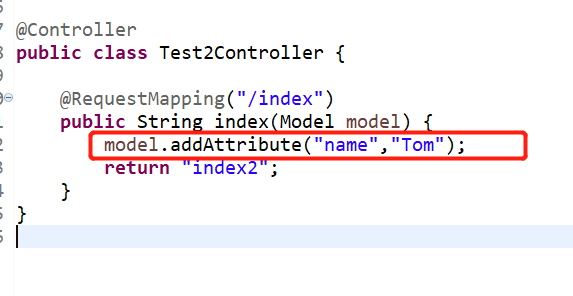


# thymeleaf模板语法

接上述案例 这里只针对thymeleaf中模板语法进行测试

## 后台封装数据

数据可以封装到Model中 前端页面通过key进行获取

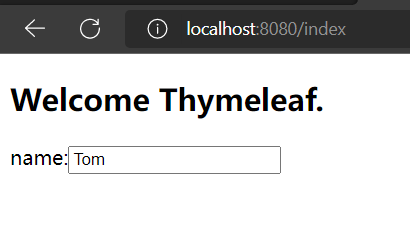
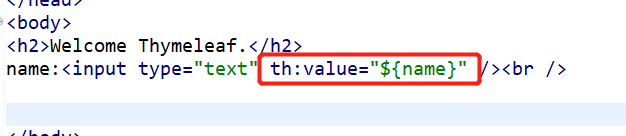


## 文本取值

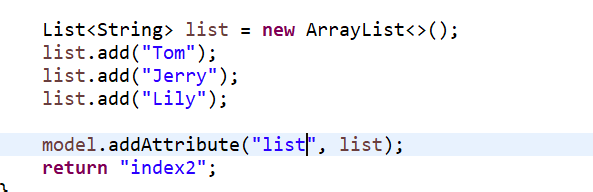
### th:text

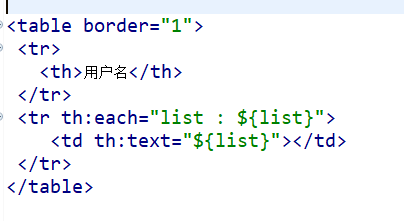


### th:value



### th:each





### th:src

img标签或style 又或是script和a标签 引入其它静态文件或连接时使用

src里面的@符号用来获取当前项目的根路径



# 集成Mybatis

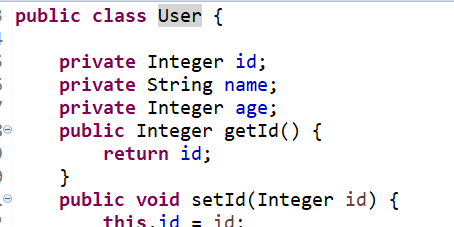
## 依赖包

|  |
| --- |
| <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.21</version>  </dependency> |

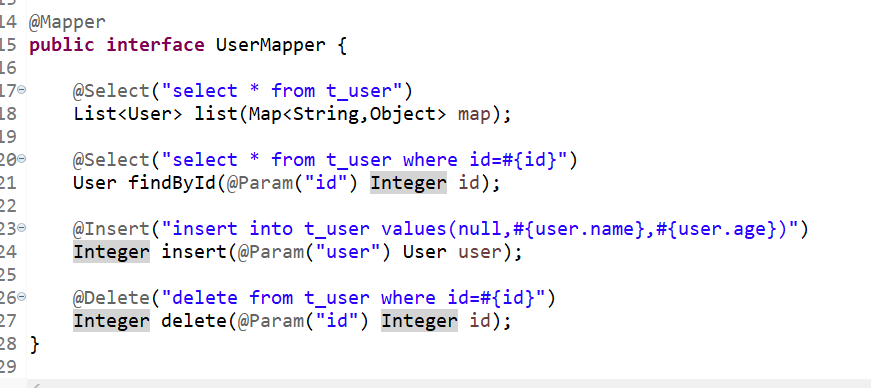
## application.properties添加mybatis配置

|  |
| --- |
| spring.thymeleaf.suffix=.html  spring.thymeleaf.mode=HTML5  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.content-type=text/html  spring.datasource.url=jdbc:mysql://192.168.199.108:3306/db\_springboot  spring.datasource.username=root  spring.datasource.password=Karen@1234  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  # 扫描自定义的mapper映射文件  #mybatis.mapper-locations=classpath:mapper/\*.xml |

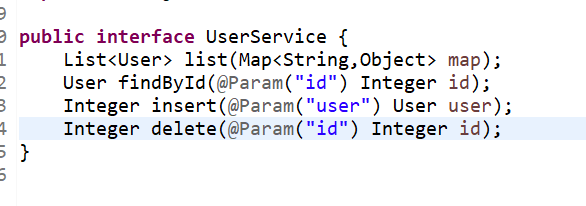
## 创建实体类



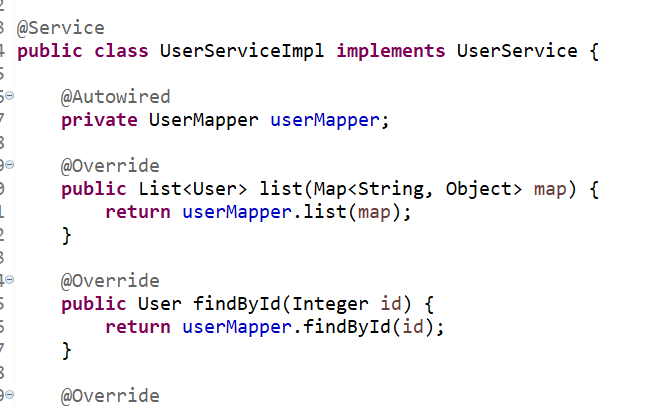
## mapper接口



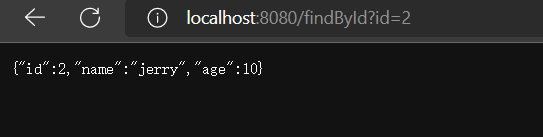
## service接口



## service实现类



## 测试



# 事务管理

事务指的是多个SQL作为单一逻辑单元进行执行的操作，要么全部成功 要么全部执行不成功。

特性 ACID

原子性：多个SQL语句要么都成功 要么都不成功

一致性：多个SQL语句执行完成后 所有的数据必须保持一致

隔离性：当前事务的修改必须与其它事务隔离开   
不能同时操作同一个数据

持久性：事务执行完毕后 对数据的影响是持久性的

怎么用？

在需要控制的service上添加@Transactional注解即可 会在方法执行完成之后在统一提交事务 如果方法抛出了异常 那么整个事务将不会提交

核心代码：



# 多数据源配置

通常情况下不需要在单个项目中使用多数据源的，多数据源常见于微服务项目，单体项目不建议使用 因为会增加项目的额外开销

## application中配置数据源

|  |
| --- |
| # datasource1  spring.datasource.test1.url=jdbc:mysql://192.168.199.108:3306/db\_springboot  spring.datasource.test1.username=root  spring.datasource.test1.password=Karen@1234  spring.datasource.test1.driver-class-name=com.mysql.jdbc.Driver  # datasource2  spring.datasource.test2.url=jdbc:mysql://192.168.199.108:3306/db\_springboot2  spring.datasource.test2.username=root  spring.datasource.test2.password=Karen@1234  spring.datasource.test2.driver-class-name=com.mysql.jdbc.Driver |

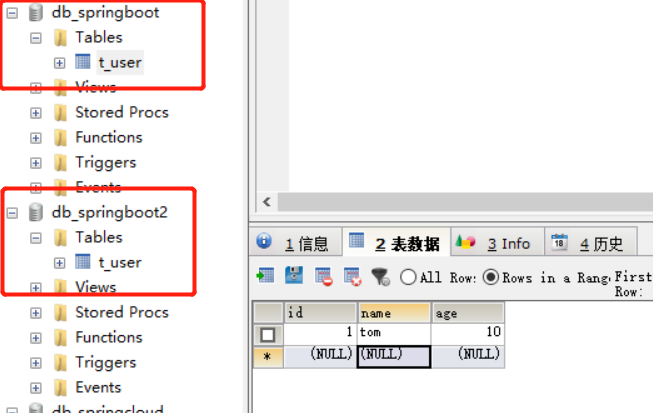
## 注入多数据源

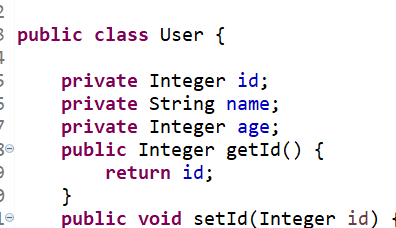
|  |
| --- |
| package com.etjava.config;  import javax.sql.DataSource;  import org.apache.ibatis.session.SqlSessionFactory;  import org.mybatis.spring.SqlSessionFactoryBean;  import org.mybatis.spring.SqlSessionTemplate;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Primary;  import org.springframework.jdbc.datasource.DataSourceTransactionManager;  @Configuration  @MapperScan(basePackages = "com.etjava.mapper.test1", sqlSessionTemplateRef = "test1SqlSessionTemplate")  public class DataSource1Config {  @Bean(name = "test1DataSource")  @ConfigurationProperties(prefix = "spring.datasource.test1")  //**primary代表主库 仅有这个可以使用@primary**  @Primary  public DataSource testDataSource() {  return DataSourceBuilder.create().build();  }  @Bean(name = "test1SqlSessionFactory")  @Primary  public SqlSessionFactory testSqlSessionFactory(@Qualifier("test1DataSource") DataSource dataSource)  throws Exception {  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(dataSource);  return bean.getObject();  }  @Bean(name = "test1TransactionManager")  @Primary  public DataSourceTransactionManager testTransactionManager(@Qualifier("test1DataSource") DataSource dataSource) {  return new DataSourceTransactionManager(dataSource);  }  @Bean(name = "test1SqlSessionTemplate")  @Primary  public SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("test1SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |

第二个数据源配置

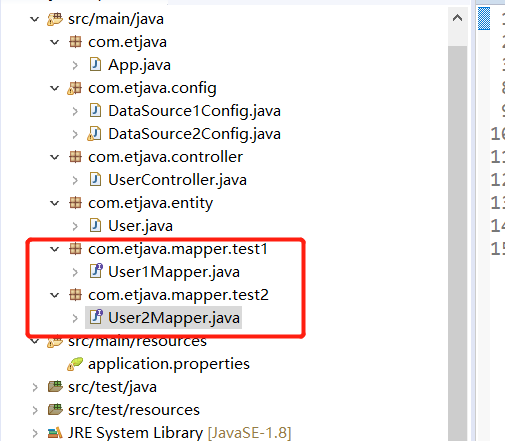


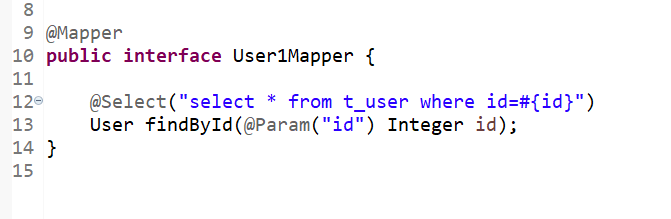
## 新建数据库及对应的实体对象

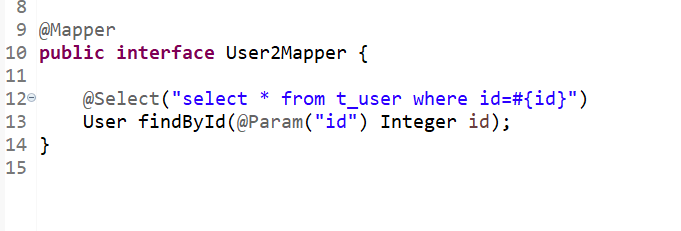




## Mapper接口



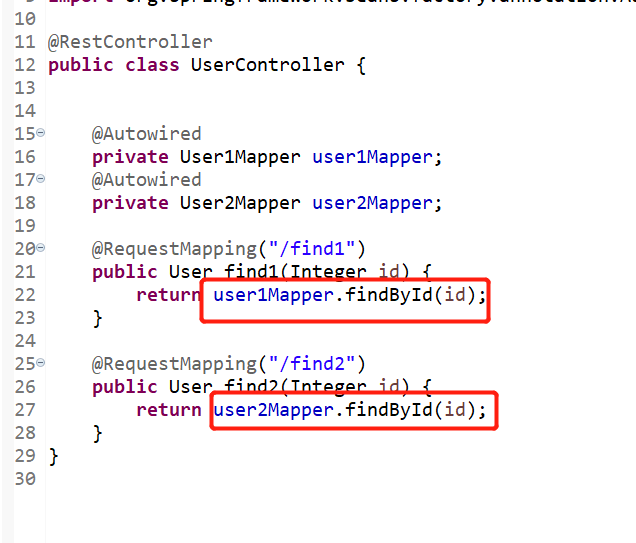




## 测试

两个find方法 分别调用不同的mapper接口

因为数据源注入的时候分别为两个mapper注入了不同的库



# 集成MybatisPlus

MybatisPlus是国内封装的一款方便简化开发的Mybatis组件

特性：封装了常用的一些CURD操作 及代码的自动生成

## 常用操作

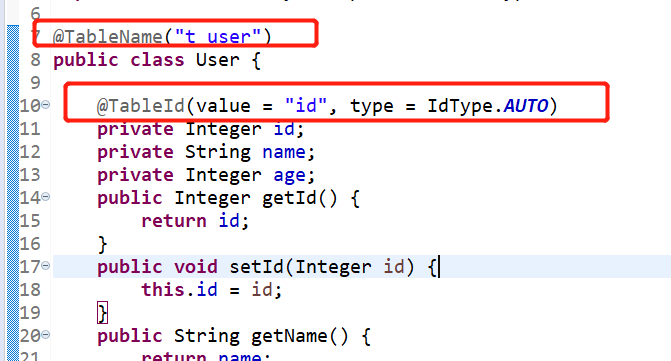
### 依赖包

MP依赖于Mybatis的 因此需要mybatis的支持

|  |
| --- |
| <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus</artifactId>  <version>2.1.4</version>  </dependency> |

### Bean对象

在创建Bean时 需要通过@Table注解指定对应的表



### Mapper接口

Mapper接口只需要集成BaseMapper<T>接口就可以了



### MP的配置类

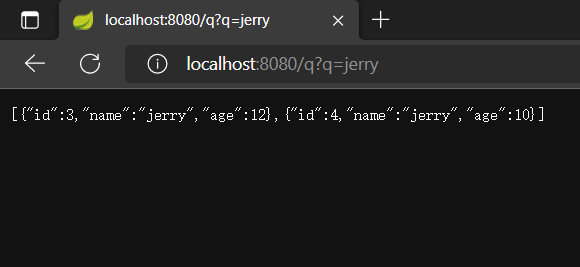
该类用来配置mybatisplus

|  |
| --- |
| **package** com.etjava.config;  **import** javax.sql.DataSource;  **import** org.apache.ibatis.mapping.DatabaseIdProvider;  **import** org.apache.ibatis.plugin.Interceptor;  **import** org.mybatis.spring.boot.autoconfigure.MybatisProperties;  **import** org.mybatis.spring.boot.autoconfigure.SpringBootVFS;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.core.io.DefaultResourceLoader;  **import** org.springframework.core.io.ResourceLoader;  **import** org.springframework.util.ObjectUtils;  **import** org.springframework.util.StringUtils;  **import** com.baomidou.mybatisplus.MybatisConfiguration;  **import** com.baomidou.mybatisplus.MybatisXMLLanguageDriver;  **import** com.baomidou.mybatisplus.plugins.PaginationInterceptor;  **import** com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean;  @Configuration  **public** **class** MybatisPlusConfig {  @Autowired  **private** DataSource dataSource;  @Autowired  **private** MybatisProperties properties;  @Autowired  **private** ResourceLoader resourceLoader = **new** DefaultResourceLoader();  @Autowired(required = **false**)  **private** Interceptor[] interceptors;  @Autowired(required = **false**)  **private** DatabaseIdProvider databaseIdProvider;  /\*\*  \* mybatis-plus分页插件  \*/  @Bean  **public** PaginationInterceptor paginationInterceptor() {  PaginationInterceptor page = **new** PaginationInterceptor();  page.setDialectType("mysql");  **return** page;  }  /\*\*  \* 这里全部使用mybatis-autoconfigure 已经自动加载的资源。不手动指定  \* 配置文件和mybatis-boot的配置文件同步  \* **@return**  \*/  @Bean  **public** MybatisSqlSessionFactoryBean mybatisSqlSessionFactoryBean() {  MybatisSqlSessionFactoryBean mybatisPlus = **new** MybatisSqlSessionFactoryBean();  mybatisPlus.setDataSource(dataSource);  mybatisPlus.setVfs(SpringBootVFS.**class**);  **if** (StringUtils.*hasText*(**this**.properties.getConfigLocation())) {  mybatisPlus.setConfigLocation(**this**.resourceLoader.getResource(**this**.properties.getConfigLocation()));  }  mybatisPlus.setConfiguration(properties.getConfiguration());  **if** (!ObjectUtils.*isEmpty*(**this**.interceptors)) {  mybatisPlus.setPlugins(**this**.interceptors);  }  MybatisConfiguration mc = **new** MybatisConfiguration();  mc.setDefaultScriptingLanguage(MybatisXMLLanguageDriver.**class**);  mybatisPlus.setConfiguration(mc);  **if** (**this**.databaseIdProvider != **null**) {  mybatisPlus.setDatabaseIdProvider(**this**.databaseIdProvider);  }  **if** (StringUtils.*hasLength*(**this**.properties.getTypeAliasesPackage())) {  mybatisPlus.setTypeAliasesPackage(**this**.properties.getTypeAliasesPackage());  }  **if** (StringUtils.*hasLength*(**this**.properties.getTypeHandlersPackage())) {  mybatisPlus.setTypeHandlersPackage(**this**.properties.getTypeHandlersPackage());  }  **if** (!ObjectUtils.*isEmpty*(**this**.properties.resolveMapperLocations())) {  mybatisPlus.setMapperLocations(**this**.properties.resolveMapperLocations());  }  **return** mybatisPlus;  }  } |

### 测试

这里略过service 直接在controller中进行测试

|  |
| --- |
| **package** com.etjava.controller;  **import** org.apache.ibatis.session.RowBounds;  **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** com.baomidou.mybatisplus.mapper.EntityWrapper;  **import** com.baomidou.mybatisplus.mapper.Wrapper;  **import** com.etjava.entity.User;  **import** com.etjava.mapper.UserMapper;  @RestController  **public** **class** UserController {  @Autowired  **private** UserMapper userMapper;  @GetMapping("/find")  **public** Object fidOne(Integer id) {  **return** userMapper.selectById(id);  }  @RequestMapping("/save")  **public** Object save(User user) {  **return** userMapper.insert(user);  }  /\*\*  \* 分页查询  \* **@param** pagenum  \* **@param** pageSize  \* **@return**  \*/  @RequestMapping("/list")  **public** Object list(Integer pagenum,Integer pageSize) {  Wrapper<User> warpper = **new** EntityWrapper<>();  RowBounds bonds = **new** RowBounds((pagenum-1)\*pageSize,pageSize);  **return** userMapper.selectPage(bonds, warpper);  }  /\*\*  \* 条件查询  \* **@param** q  \* **@return**  \*/  @RequestMapping("/q")  **public** Object q(String q) {  Wrapper<User> wrapper = **new** EntityWrapper<>();  wrapper.eq("name", q);  wrapper.between("id", 0, 100);  wrapper.groupBy("name");  wrapper.isNotNull("name");  wrapper.orderBy("id", **false**);  **return** userMapper.selectList(wrapper);  }  } |



## 代码生成

https://baomidou.gitee.io/mybatis-plus-doc/#/generate-code

### 代码生成需要的依赖包

后边在做补充…

# 集成lombok

MP(MybatisPlus) 是封装了各种Base接口 简化我们的开发

Lombok 可以让帮我们简化bean的开发

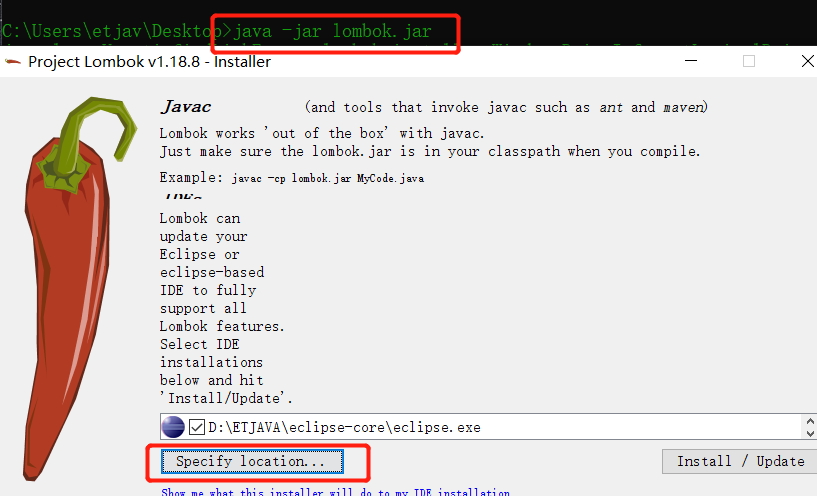
可以单独通过@Setter @Getter 可以帮我们生成get set方法

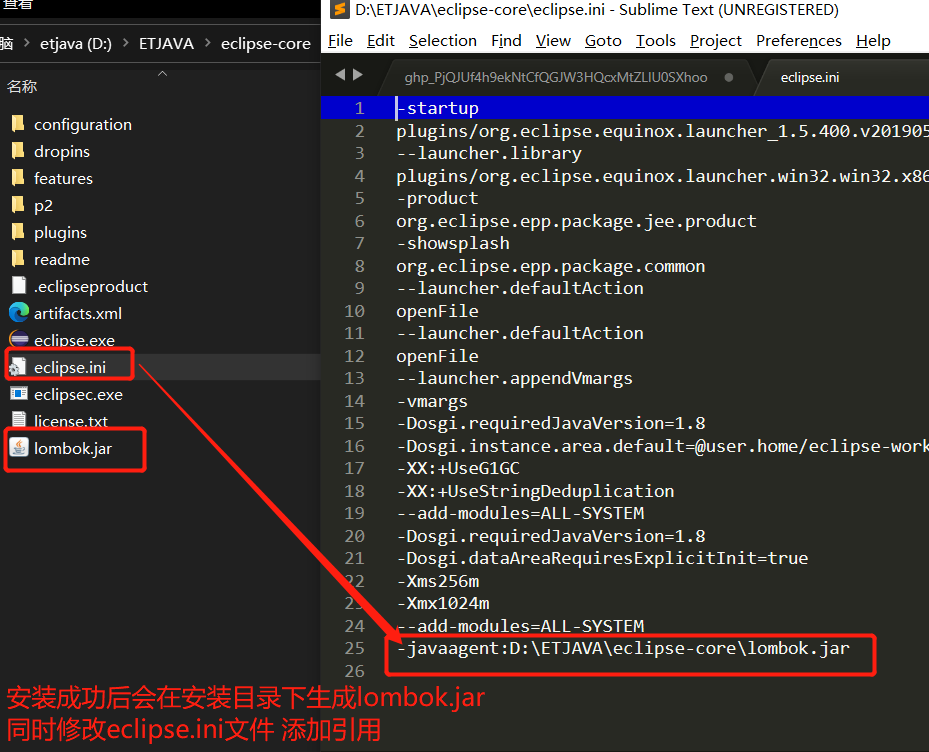
也可以通过@Data来将Bean中的常用方法都注入进来

例如 构造方法，toString方法，get set方法，hashCode等

## 安装lombok

运行lombok.jar





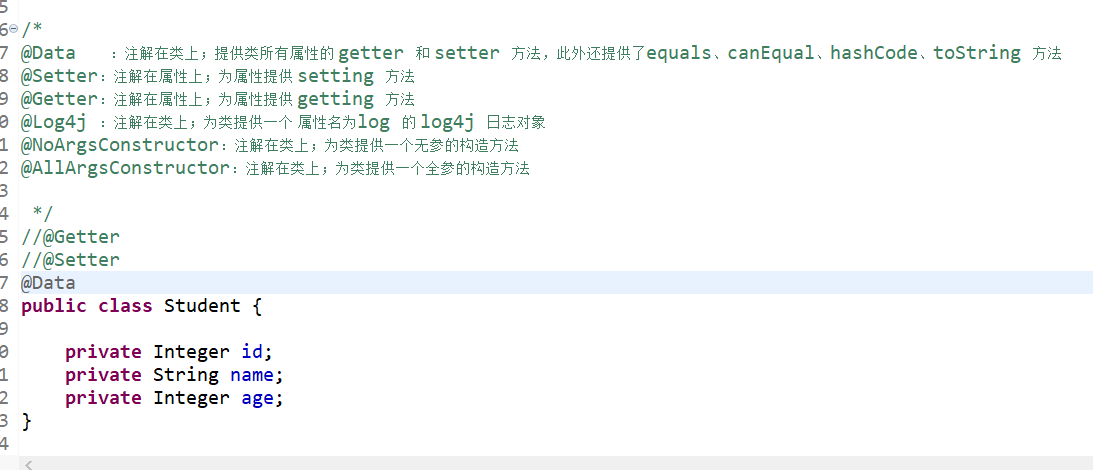
安装完成后建议重启下IDE

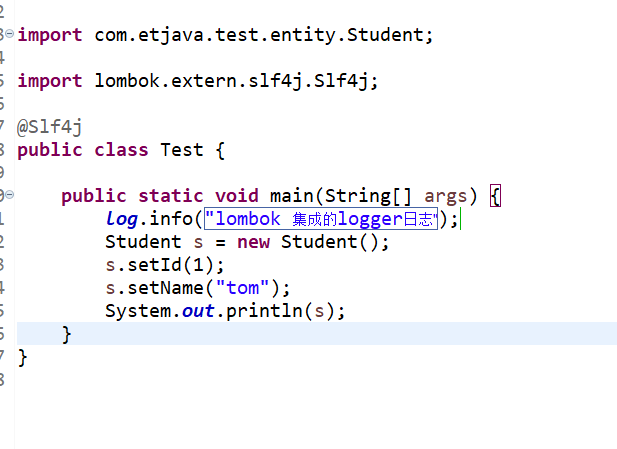
## 添加依赖

|  |
| --- |
| <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency> |

## 测试

首先创建bean





# 集成Shiro

shiro是一个安全框架 主要用于身份认证，权限管理，授权，会话等

相比Spring Security功能上较为简单

**此DEMO将身份认证及授权放在一起 另外这里忽略了service层 实际开发中应该在service中处理业务逻辑**

## 添加依赖

|  |
| --- |
| <!-- shiro 安全框架 -->  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>1.4.0</version>  </dependency>    <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-ehcache</artifactId>  <version>1.4.0</version>  </dependency> |

## 创建Shiro缓存配置文件 ehcache-shiro.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <ehcache updateCheck=*"false"* name=*"cacheManagerConfigFile"*>  <defaultCache  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"120"*  timeToLiveSeconds=*"120"*  overflowToDisk=*"false"*  diskPersistent=*"false"*  diskExpiryThreadIntervalSeconds=*"120"*  memoryStoreEvictionPolicy=*"LRU"*/>  <cache name=*"shiro-activeSessionCache"*  eternal=*"false"*  maxElementsInMemory=*"10000"*  overflowToDisk=*"false"*  timeToIdleSeconds=*"0"*  timeToLiveSeconds=*"0"*  statistics=*"true"*/>  </ehcache> |

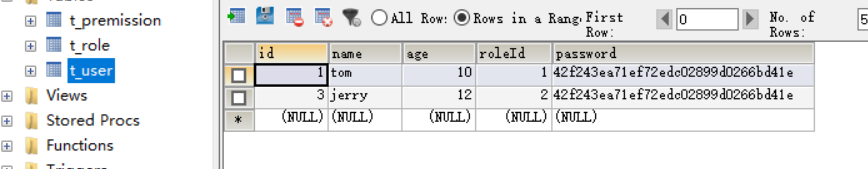
## Shiro配置

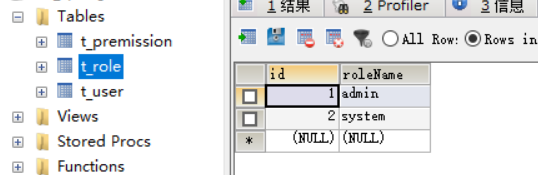
SSM时 该配置在application.xml中进行设置

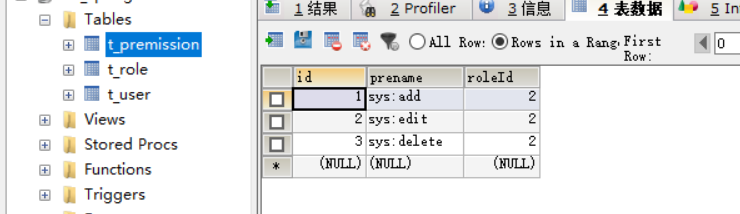
Springboot 则在config类中进行配置

|  |
| --- |
| package com.etjava.config;  import java.util.HashMap;  import java.util.LinkedHashMap;  import java.util.Map;  import javax.servlet.Filter;  import org.apache.shiro.cache.ehcache.EhCacheManager;  import org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator;  import org.apache.shiro.session.mgt.eis.MemorySessionDAO;  import org.apache.shiro.spring.LifecycleBeanPostProcessor;  import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;  import org.apache.shiro.spring.web.ShiroFilterFactoryBean;  import org.apache.shiro.web.mgt.DefaultWebSecurityManager;  import org.apache.shiro.web.servlet.SimpleCookie;  import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;  import org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import com.etjava.realm.UserRealm;  @Configuration  public class ShiroConfiguration {  /\*\*  \* ShiroFilterFactoryBean 处理拦截资源文件问题。  \* 注意：单独一个ShiroFilterFactoryBean配置是或报错的，以为在  \* 初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager  \* Filter Chain定义说明 1、一个URL可以配置多个Filter，使用逗号分隔 2、当设置多个过滤器时，全部验证通过，才视为通过  \* 3、部分过滤器可指定参数，如perms，roles  \*/  @Bean  public ShiroFilterFactoryBean shirFilter(org.apache.shiro.mgt.SecurityManager securityManager) {    ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();    // 必须设置 SecurityManager  shiroFilterFactoryBean.setSecurityManager(securityManager);    // 拦截器.  Map<String, String> filterChainDefinitionMap = new LinkedHashMap<String, String>();  //配置静态资源允许访问  filterChainDefinitionMap.put("/js/\*\*","anon");  filterChainDefinitionMap.put("/css/\*\*","anon");  filterChainDefinitionMap.put("/index","anon");  filterChainDefinitionMap.put("/user/login","anon");  // <!-- authc:所有url都必须认证通过才可以访问; anon:所有url都都可以匿名访问-->  filterChainDefinitionMap.put("/\*\*", "authc");  // 如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面  shiroFilterFactoryBean.setLoginUrl("/user/login");  shiroFilterFactoryBean.setLoginUrl("/user/loginAction");  // 未授权界面;  shiroFilterFactoryBean.setUnauthorizedUrl("/403");  Map<String, Filter> filters=new HashMap<String,Filter>();  shiroFilterFactoryBean.setFilters(filters);  shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);  return shiroFilterFactoryBean;  }    @Bean  public EhCacheManager getEhCacheManager() {  EhCacheManager em = new EhCacheManager();  em.setCacheManagerConfigFile("classpath:ehcache-shiro.xml");  return em;  }  // 开启Controller中的shiro注解  @Bean  public DefaultAdvisorAutoProxyCreator getDefaultAdvisorAutoProxyCreator() {  DefaultAdvisorAutoProxyCreator daap = new DefaultAdvisorAutoProxyCreator();  daap.setProxyTargetClass(true);  return daap;  }  /\*\*  \* 配置org.apache.shiro.web.session.mgt.DefaultWebSessionManager  \* @return  \*/  @Bean  public DefaultWebSessionManager getDefaultWebSessionManager(){  DefaultWebSessionManager defaultWebSessionManager=new DefaultWebSessionManager();  defaultWebSessionManager.setSessionDAO(getMemorySessionDAO());  defaultWebSessionManager.setGlobalSessionTimeout(4200000);  defaultWebSessionManager.setSessionValidationSchedulerEnabled(true);  defaultWebSessionManager.setSessionIdCookieEnabled(true);  defaultWebSessionManager.setSessionIdCookie(getSimpleCookie());  return defaultWebSessionManager;  }  /\*\*  \* 配置org.apache.shiro.session.mgt.eis.MemorySessionDAO  \* @return  \*/  @Bean  public MemorySessionDAO getMemorySessionDAO(){  MemorySessionDAO memorySessionDAO=new MemorySessionDAO();  memorySessionDAO.setSessionIdGenerator(javaUuidSessionIdGenerator());  return memorySessionDAO;  }  @Bean  public JavaUuidSessionIdGenerator javaUuidSessionIdGenerator(){  return new JavaUuidSessionIdGenerator();  }  /\*\*  \* session自定义cookie名  \* @return  \*/  @Bean  public SimpleCookie getSimpleCookie(){  SimpleCookie simpleCookie=new SimpleCookie();  simpleCookie.setName("security.session.id");  simpleCookie.setPath("/");  return simpleCookie;  }  @Bean  public LifecycleBeanPostProcessor getLifecycleBeanPostProcessor(){  return new LifecycleBeanPostProcessor();  }  @Bean(name = "securityManager")  public DefaultWebSecurityManager getDefaultWebSecurityManager(UserRealm userRealm) {  DefaultWebSecurityManager dwsm = new DefaultWebSecurityManager();  dwsm.setRealm(userRealm);  // <!-- 用户授权/认证信息Cache, 采用EhCache 缓存 -->  dwsm.setCacheManager(getEhCacheManager());  dwsm.setSessionManager(getDefaultWebSessionManager());  return dwsm;  }  @Bean  public UserRealm userRealm(EhCacheManager cacheManager) {  UserRealm userRealm = new UserRealm();  userRealm.setCacheManager(cacheManager);  return userRealm;  }  /\*\*  \* 开启shrio注解支持  \* @param userRealm  \* @return  \*/  @Bean  public AuthorizationAttributeSourceAdvisor getAuthorizationAttributeSourceAdvisor(UserRealm userRealm){  AuthorizationAttributeSourceAdvisor aasa=new AuthorizationAttributeSourceAdvisor();  aasa.setSecurityManager(getDefaultWebSecurityManager(userRealm));  return aasa;  }  } |

## 表结构及数据







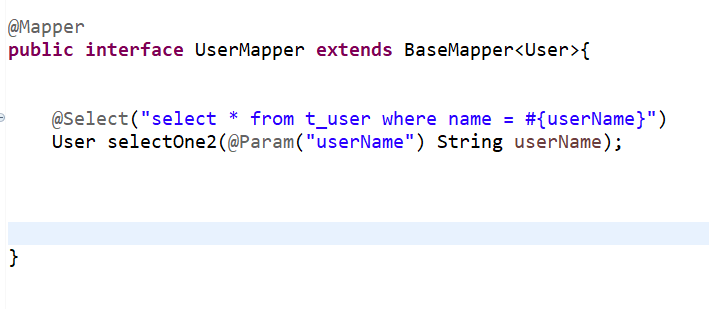
## Realm配置

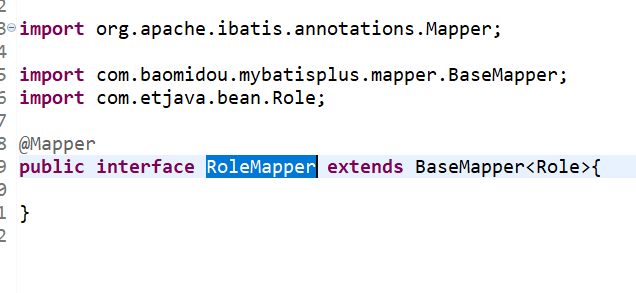
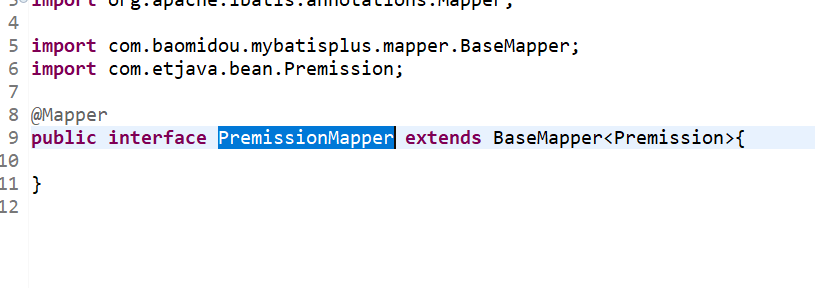
realm是用来进行身份验证及授权

**shiro安全框架的核心**

|  |
| --- |
| package com.etjava.realm;  import java.util.ArrayList;  import java.util.List;  import org.apache.shiro.authc.AuthenticationException;  import org.apache.shiro.authc.AuthenticationInfo;  import org.apache.shiro.authc.AuthenticationToken;  import org.apache.shiro.authc.SimpleAuthenticationInfo;  import org.apache.shiro.authz.AuthorizationInfo;  import org.apache.shiro.authz.SimpleAuthorizationInfo;  import org.apache.shiro.realm.AuthorizingRealm;  import org.apache.shiro.subject.PrincipalCollection;  import org.springframework.beans.factory.annotation.Autowired;  import com.baomidou.mybatisplus.mapper.EntityWrapper;  import com.baomidou.mybatisplus.mapper.Wrapper;  import com.etjava.bean.Premission;  import com.etjava.bean.Role;  import com.etjava.bean.User;  import com.etjava.mapper.PremissionMapper;  import com.etjava.mapper.RoleMapper;  import com.etjava.mapper.UserMapper;  public class UserRealm extends AuthorizingRealm {    @Autowired  private UserMapper userMapper;    @Autowired  private PremissionMapper premissionMapper;  @Autowired  private RoleMapper roleMapper;  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection pc) {  SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();    // 获取登录的用户名  String username = (String)pc.getPrimaryPrincipal();  // 获取到当前用户的角色和权限  User u = userMapper.selectOne2(username);  Integer roleId = u.getRoleId();  Role role = roleMapper.selectById(roleId);  // 授权角色  info.addRole(role.getRoleName());  // 获取当前用户的权限并进行授权  Wrapper<Premission> wrapper = new EntityWrapper<Premission>();  wrapper.eq("roleId", role.getId());  List<Premission> preList = premissionMapper.selectList(wrapper );  // 需要将权限转成String类型的数组  List<String> list = new ArrayList<>();  preList.forEach(p -> {  list.add(p.getPreName());  });  info.addStringPermissions(list);      return info;  }  // 这里做登录控制  @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {  String username = (String)token.getPrincipal();  User u = userMapper.selectOne2(username);  if(u!=null) {  AuthenticationInfo auth = new SimpleAuthenticationInfo(u.getName(),u.getPassword(),"others wirte");  return auth;  }  return null;  }  } |

## 各Mapper接口



## 实体类略

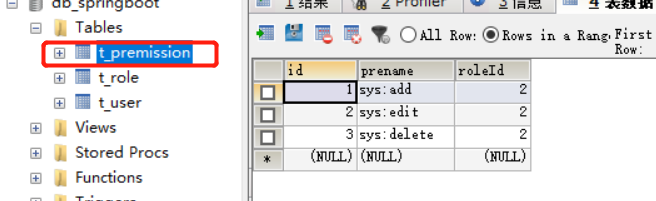
## 测试 Controller

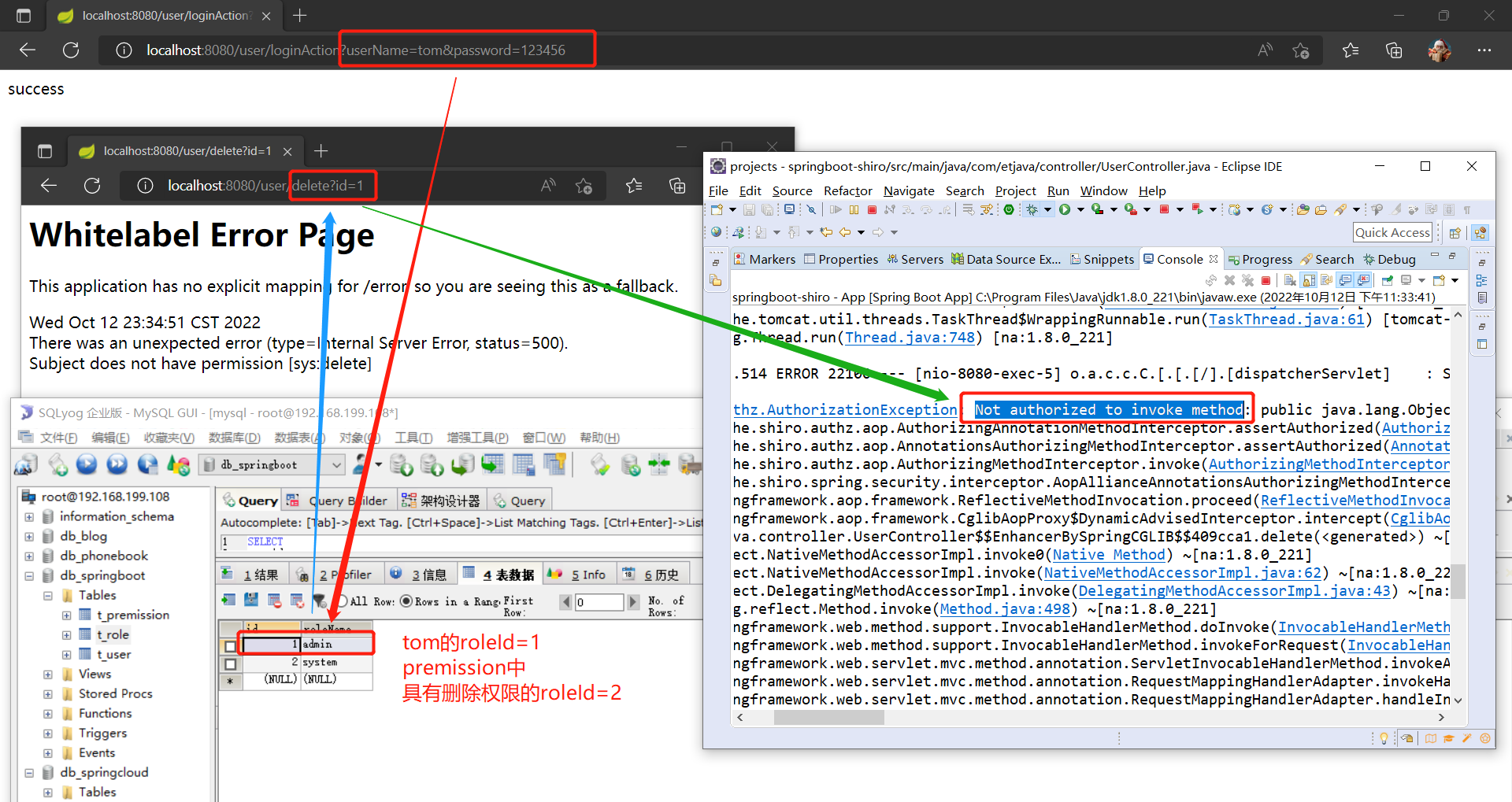
|  |
| --- |
| package com.etjava.controller;  import org.apache.shiro.SecurityUtils;  import org.apache.shiro.authc.AuthenticationToken;  import org.apache.shiro.authc.UsernamePasswordToken;  import org.apache.shiro.authz.annotation.RequiresPermissions;  import org.apache.shiro.authz.annotation.RequiresRoles;  import org.apache.shiro.subject.Subject;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.etjava.bean.User;  import com.etjava.mapper.UserMapper;  import com.etjava.util.CryptographyUtil;  import lombok.extern.slf4j.Slf4j;  @Slf4j  @RestController  @RequestMapping("/user")  public class UserController {  @Autowired  private UserMapper userMapper;    @RequestMapping("/login")  public Object login() {  return "login page";  }    @RequestMapping("/loginAction")  public Object loginAction(String userName,String password) {  Subject subject = SecurityUtils.getSubject();  String pass = CryptographyUtil.md5(password, "etjava");  AuthenticationToken token = new UsernamePasswordToken(userName,pass);  try {  // 执行无异常则登录成功 如果账号密码错误则抛出异常  subject.login(token );  return "success";  } catch (Exception e) {  log.error("登录异常 ",e);  return "error";  }  }  // 只有admin角色才可以访问这个方法  @RequiresRoles("admin")  @RequestMapping("/findById")  public Object findById(Integer id) {  User selectById = userMapper.selectById(id);  return selectById;  }  // 只有访问权限为sys:delete才可以访问该方法  @RequiresPermissions("sys:delete")  @RequestMapping("/delete")  public Object delete(Integer id) {  return userMapper.selectById(id);  }  } |

## 测试结果

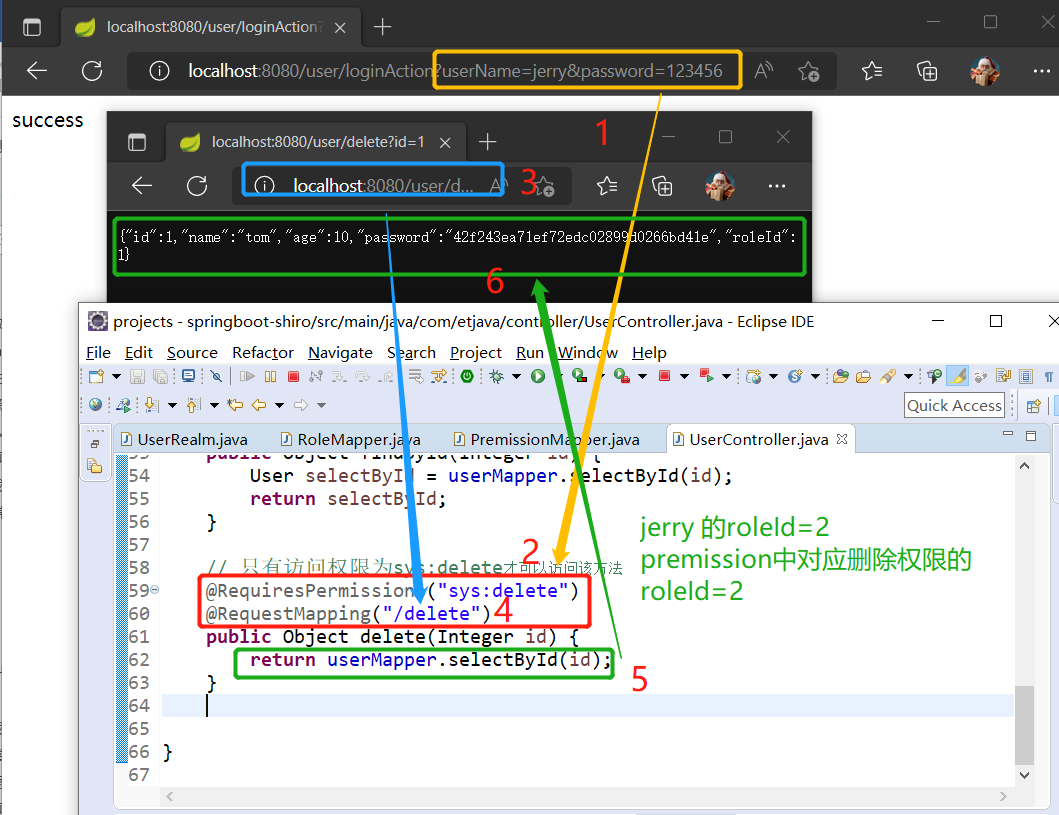
admin角色只可以访问findById方法

system角色可以访问delete方法





换个账号登录



# AOP注入全局日志

## 导入依赖

|  |
| --- |
| <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency>  <!-- aop切面 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency> |

## Log4j.properties

|  |
| --- |
| log4j.rootLogger=DEBUG,console,dailyFile,im  log4j.additivity.org.apache=true  # console 控制台输出  log4j.appender.console=org.apache.log4j.ConsoleAppender  log4j.appender.console.Threshold=DEBUG  log4j.appender.console.ImmediateFlush=true  log4j.appender.console.Target=System.err  log4j.appender.console.layout=org.apache.log4j.PatternLayout  log4j.appender.console.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # write file  log4j.appender.logFile=org.apache.log4j.FileAppender  log4j.appender.logFile.Threshold=DEBUG  log4j.appender.logFile.ImmediateFlush=true  log4j.appender.logFile.Append=true  log4j.appender.logFile.File=D:/phonebook/log/log.log4j  log4j.appender.logFile.layout=org.apache.log4j.PatternLayout  log4j.appender.logFile.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # 回滚文件(rollingFile)  log4j.appender.rollingFile=org.apache.log4j.RollingFileAppender  log4j.appender.rollingFile.Threshold=DEBUG  log4j.appender.rollingFile.ImmediateFlush=true  log4j.appender.rollingFile.Append=true  log4j.appender.rollingFile.File=D:/phonebook/log/log.log4j  log4j.appender.rollingFile.MaxFileSize=200KB  log4j.appender.rollingFile.MaxBackupIndex=50  log4j.appender.rollingFile.layout=org.apache.log4j.PatternLayout  log4j.appender.rollingFile.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # \u5B9A\u671F\u56DE\u6EDA\u65E5\u5FD7\u6587\u4EF6(dailyFile)  log4j.appender.dailyFile=org.apache.log4j.DailyRollingFileAppender  log4j.appender.dailyFile.Threshold=DEBUG  log4j.appender.dailyFile.ImmediateFlush=true  log4j.appender.dailyFile.Append=true  log4j.appender.dailyFile.File=D:/phonebook/log/log.log4j  log4j.appender.dailyFile.DatePattern='.'yyyy-MM-dd  log4j.appender.dailyFile.layout=org.apache.log4j.PatternLayout  log4j.appender.dailyFile.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # \u5E94\u7528\u4E8Esocket  log4j.appender.socket=org.apache.log4j.RollingFileAppender  log4j.appender.socket.RemoteHost=localhost  log4j.appender.socket.Port=5001  log4j.appender.socket.LocationInfo=true  # Set up for Log Factor 5  log4j.appender.socket.layout=org.apache.log4j.PatternLayout  log4j.appender.socket.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # Log Factor 5 Appender  log4j.appender.LF5\_APPENDER=org.apache.log4j.lf5.LF5Appender  log4j.appender.LF5\_APPENDER.MaxNumberOfRecords=2000  # \u53D1\u9001\u65E5\u5FD7\u5230\u6307\u5B9A\u90AE\u4EF6 appender  log4j.appender.mail=org.apache.log4j.net.SMTPAppender  log4j.appender.mail.Threshold=FATAL  log4j.appender.mail.BufferSize=10  log4j.appender.mail.From = etjava@hotmail.com  log4j.appender.mail.SMTPHost=mail.com  log4j.appender.mail.Subject=Log4J Message  log4j.appender.mail.To= etjava@hotmail.com  log4j.appender.mail.layout=org.apache.log4j.PatternLayout  log4j.appender.mail.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # \u5E94\u7528\u4E8E\u6570\u636E\u5E93  log4j.appender.database=org.apache.log4j.jdbc.JDBCAppender  log4j.appender.database.URL=jdbc:mysql://localhost:3306/test  log4j.appender.database.driver=com.mysql.jdbc.Driver  log4j.appender.database.user=root  log4j.appender.database.password=  log4j.appender.database.sql=INSERT INTO LOG4J (Message) VALUES('=[%-5p] %d(%r) --> [%t] %l: %m %x %n')  log4j.appender.database.layout=org.apache.log4j.PatternLayout  log4j.appender.database.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n  # \u81EA\u5B9A\u4E49Appender  log4j.appender.im = net.cybercorlin.util.logger.appender.IMAppender  log4j.appender.im.host = mail.cybercorlin.net  log4j.appender.im.username = username  log4j.appender.im.password = password  log4j.appender.im.recipient = corlin@cybercorlin.net  log4j.appender.im.layout=org.apache.log4j.PatternLayout  log4j.appender.im.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n |

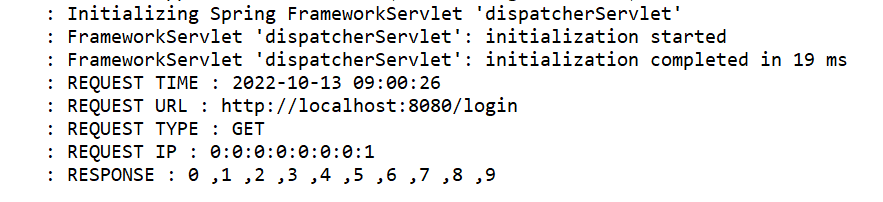
## Log的切面配置类

指定了需要在那些类中的方法进行添加开始之前和开始之后的日志记录

|  |
| --- |
| package com.etjava.log;  import java.text.SimpleDateFormat;  import java.util.Date;  import java.util.Enumeration;  import javax.servlet.http.HttpServletRequest;  import org.aspectj.lang.JoinPoint;  import org.aspectj.lang.annotation.AfterReturning;  import org.aspectj.lang.annotation.Aspect;  import org.aspectj.lang.annotation.Before;  import org.aspectj.lang.annotation.Pointcut;  import org.springframework.stereotype.Component;  import org.springframework.web.context.request.RequestContextHolder;  import org.springframework.web.context.request.ServletRequestAttributes;  import lombok.extern.slf4j.Slf4j;  @Aspect  @Component  @Slf4j  public class EtjavaLogAspect {    // 扫描所有controller包 给所有public修饰的请求方法添加统一的访问日志  @Pointcut("execution(public \* com.etjava.controller..\*.\*(..))")  public void webLog() {    }  @Before("webLog()")  public void doBefore(JoinPoint joinPoint) throws Throwable {  // 接收到请求，记录请求内容  ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  HttpServletRequest request = attributes.getRequest();  // 记录下请求内容  log.info("REQUEST TIME : "+new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()));  log.info("REQUEST URL : " + request.getRequestURL().toString());  log.info("REQUEST TYPE : " + request.getMethod());  log.info("REQUEST IP : " + request.getRemoteAddr());  Enumeration<String> enu = request.getParameterNames();  while (enu.hasMoreElements()) {  String name = (String) enu.nextElement();    log.info("name:{},value:{}", name, request.getParameter(name));  }  }      @AfterReturning(returning = "ret", pointcut = "webLog()")  public void doAfterReturning(Object ret) throws Throwable {    // 处理完请求，返回内容  log.info("RESPONSE : " + ret);  }  } |

## 测试controller

|  |
| --- |
| **package** com.etjava.controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public** **class** TestController {    @RequestMapping("/login")  **public** String hello(String username,String password) {    StringBuffer buf = **new** StringBuffer();  **for**(**int** i = 0; i< 10;i ++) {  buf.append(i+" ,");  }    //@SuppressWarnings("unused")  //int i = 1/0 ;  **return** buf.substring(0, buf.length()-2).toString();  }  } |



# 定时器

springboot中实现思路

首先创建一个component 然后让springboot当作一个组件去加载 使用@Component注解即可

然后 在component组件中定义要执行的方法 然后通过@Scheduled(fixedRate = 毫秒数) 注解配置要执行的间隔时间

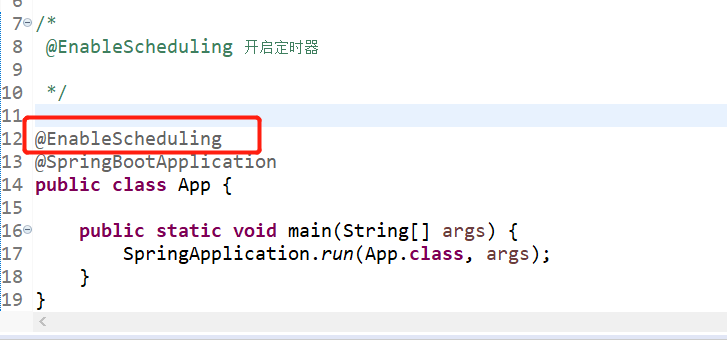
最后在Springboot的启动类中开启定时器即可

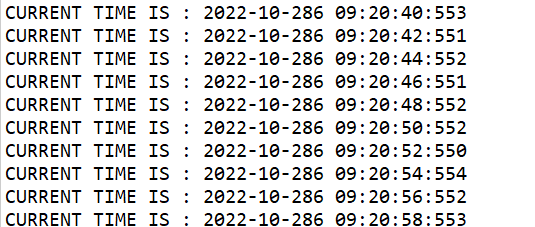
* springboot中开启任何组件 都是使用@Enable后面跟上需要开启的名称 例如缓存 @Enablecaching

## 定义Component组件

|  |
| --- |
| **package** com.etjava.component;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **import** org.springframework.scheduling.annotation.Scheduled;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** MyTask {  // fixedRate 单位是毫秒  @Scheduled(fixedRate = 2000)  **public** **void** showTime() {  System.***out***.println("CURRENT TIME IS : "+**new** SimpleDateFormat("YYYY-MM-DD HH:mm:ss:SSS").format(**new** Date()));  }  } |

## 启动类添加开启定时器注解

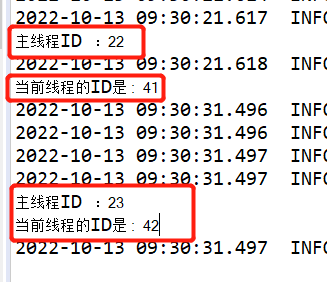




# 异步方法

程序执行过程中可以动态的去处理些其它相关但密度不紧的事情 此时就可以借助异步方式进行处理

判断一个方法是否为异步的 只需要与当前正在执行的线程ID进行比对就可以了 相同 则表示是同一个线程(非异步) 不同则表示为异步



实现思路：

首先新建一个组件并注入到spring中 通过@Component注解

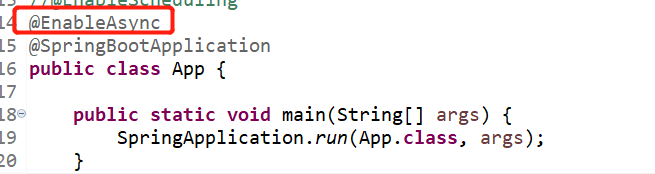
在组件中通过@Async标记一个异步方法

最后在启动类中开启异步调用 ，接下来哪里需要用到 直接通过@Autowired注入进来 然后调用里面的标记为异步的方法就可以

## 定义Component组件

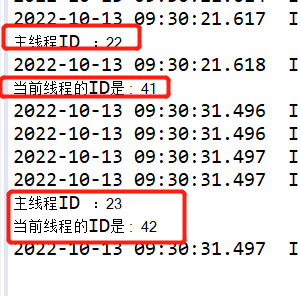
|  |
| --- |
| **package** com.etjava.component;  **import** org.springframework.scheduling.annotation.Async;  **import** org.springframework.stereotype.Component;  /\*\*  \* 测试异步方法  \* **@author** etjav  \*  \*/  @Component // 注入到spring中  **public** **class** MyAsync {  @Async // 该注解表示此方法为异步调用  **public** **void** async() {  System.***out***.println("当前线程的ID是 : "+Thread.*currentThread*().getId());  }    } |

## 启动类中开启异步调用



## controller中测试异步调用



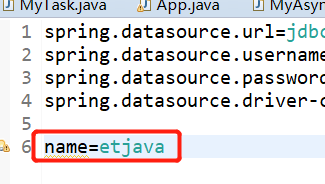


# 自定义参数

自定义参数就是将参数通过KV模式写在springboot的配置文件中的参数

获取自定义参数 通过@Value(“参数的Key”)方式即可获取到配置文件中的自定义参数

## 配置文件中添加自定义参数

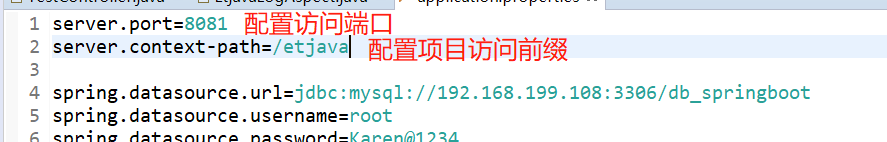


## 注入自定义参数

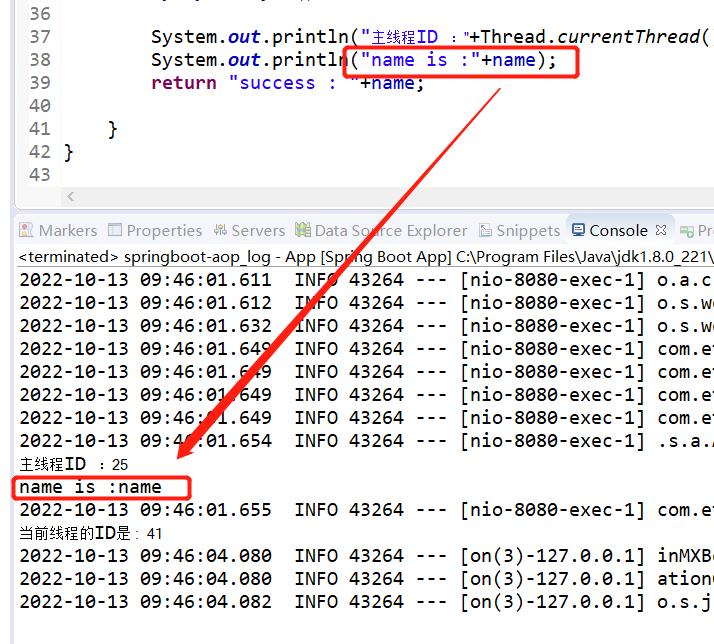


# 自定义访问端口和路径

在application.properties中通过server.port指定当前项目的访问端口即可



## 测试

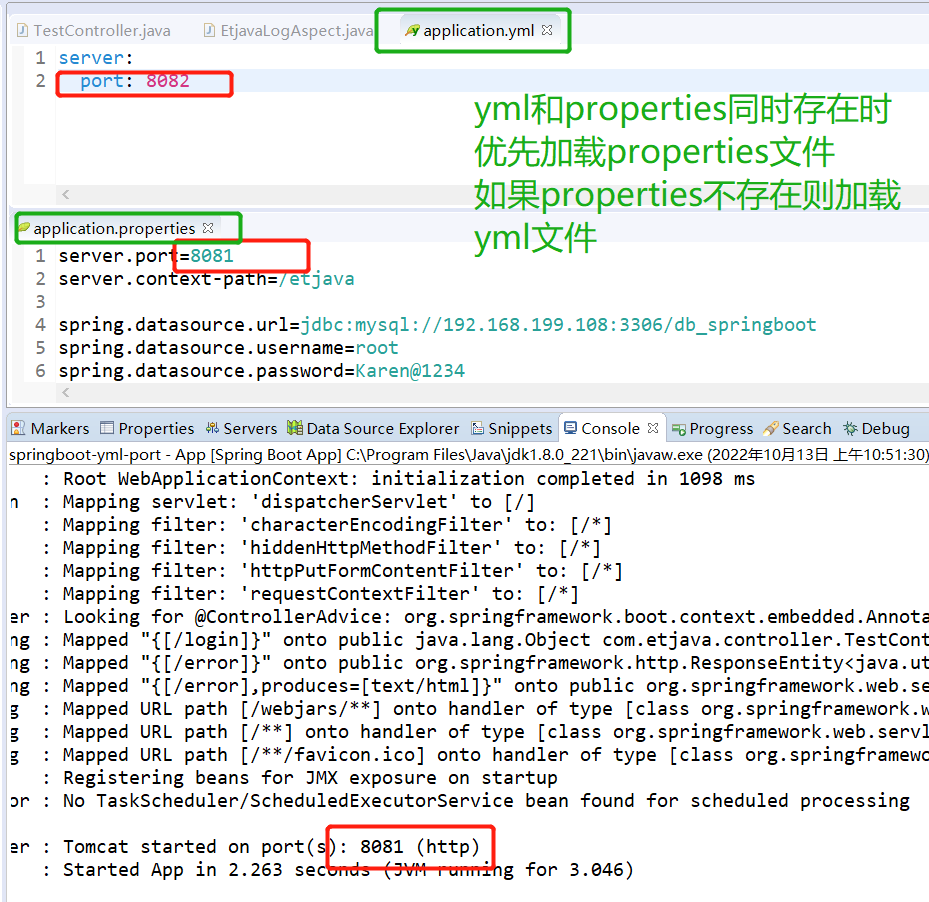


# yml文件配置

yml格式的文件功能与application.properties完全一致的

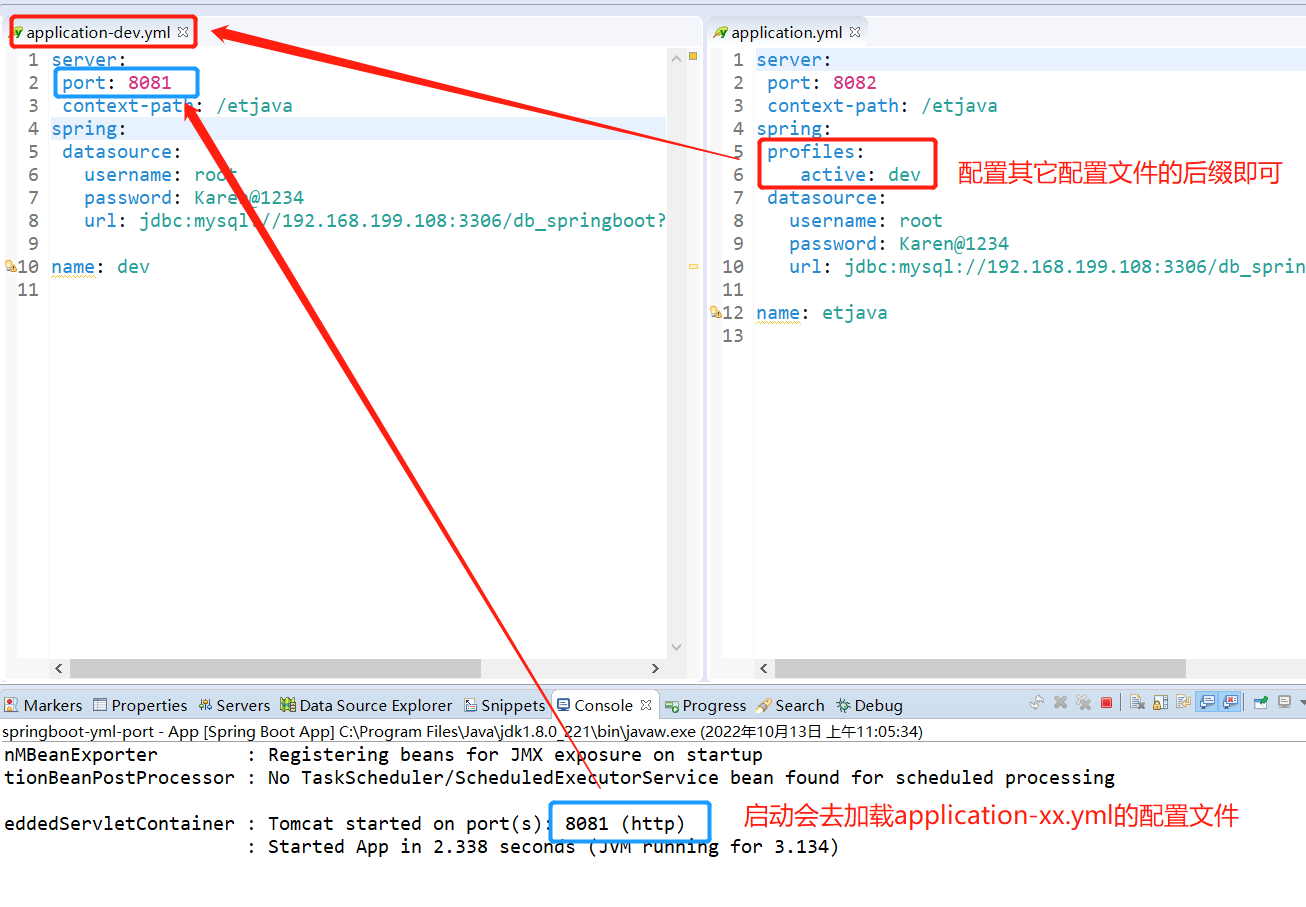
不同的是yml 以树形结构展示和配置springboot相关信息

两个文件同时存在时 默认加载properties文件



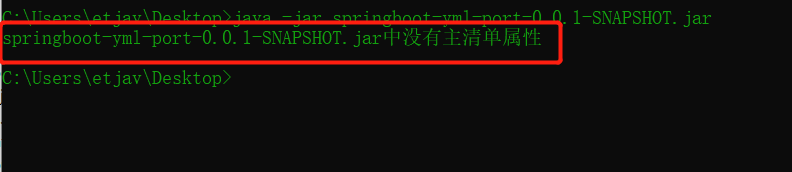
# 多环境配置

通过spring.profiles.active 指定需要采用的yml格式文件名称后缀(-dev 或hotfix等)



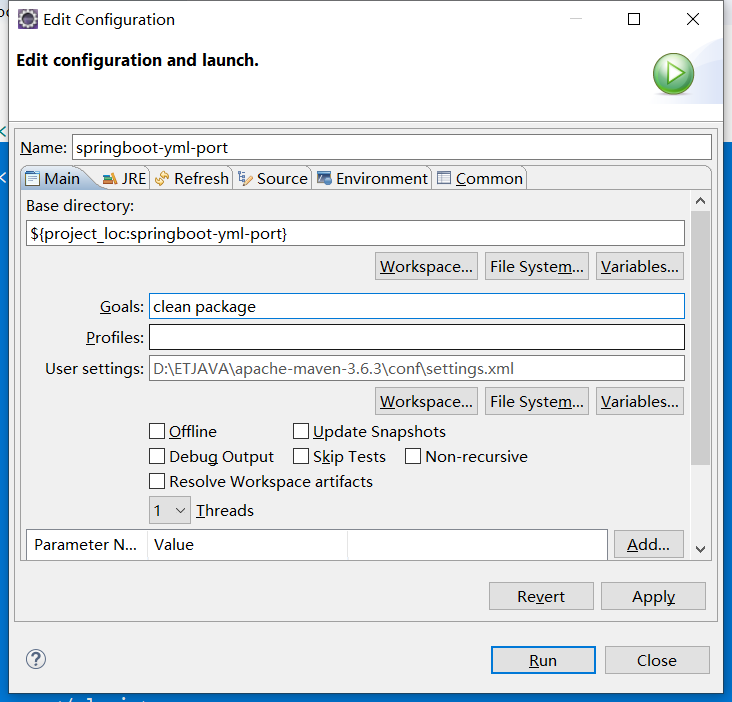
# 打包发布

在执行package或install之前需要在pom中添加如下内容 指定jar包中的启动类及其它相关清单 否则启动时会抛出缺少清单异常



|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <maimClass>com.etjava.App</maimClass>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |

然后选中项目右键 run as 执行package或install



启动时将打好的jar包放到需要的地方后 直接通过java -jar方式启动即可

