## MyBatis

### MyBatis原始使用（mybatis-1）

（1）准备：映射文件





（2）通过sqlsession使用

|  |
| --- |
| *//1.读取配置文件* InputStream in = Resources.getResourceAsStream(**"SqlMapConfig.xml"**); *//2.创建SqlSessionFactory工厂* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(in); *//3.使用工厂生产SqlSession对象* SqlSession session = sqlSessionFactory.openSession(); *//4.执行Sql语句* User user = session.selectOne(**"test.findUserById"**, 10); *//5. 打印结果* System.out.println(user); *//6.释放资源* session.close(); in.close(); |

### 基于原始DAO（mybatis-2）

通过引入dao层将SqlSession 以及相应的SQL操作进行封装而已。

1. 准备：mapper文件如上

Dao层封装如下：

|  |
| --- |
| **private** SqlSessionFactory **sqlSessionFactory**; *// 通过构造方法注入* **public** UserDaoImpl(SqlSessionFactory sqlSessionFactory){  **this**.**sqlSessionFactory** = sqlSessionFactory; } **public** User findUserById(Integer id) {  *//sqlSession是线程不安全的，所以它的最佳使用范围在方法体内* SqlSession sqlSession = **sqlSessionFactory**.openSession();  User user = sqlSession.selectOne(**"test.findUserById"**,id);  sqlSession.close();  **return** user; } |

1. 使用

|  |
| --- |
| **private** SqlSessionFactory **factory**; **private** UserDao **userdao**; **private** InputStream **in**; *// 作用：在测试方法前执行这个方法* @Before **public void** setUp() **throws** Exception{  *//1.读取配置文件* **in** = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);  *//2.创建SqlSessionFactory工厂* **factory** = **new** SqlSessionFactoryBuilder().build(**in**); }  **public void** testFindUserById() **throws** Exception{  *// 将初始化好的工厂注入到实现类中* UserDao userdao = **new** UserDaoImpl(**factory**);  User user = userdao.findUserById(34);  System.***out***.println(user);  *//User [id=34, username=小小, sex=1, birthday=Mon Jul 15 00:00:00 CST 2019, address=西安市]* } |

### 基于代理DAO（mybatis-3）

原始DAO需要定义DAOImpl类，并且在这个类中有大量的sqlsession重复代码，通过代理DAO可以解决此问题。

1. 准备：

A：mapper文件user.xml

B：User.xml文件中的mapper文件必须是全路径名：

|  |
| --- |
| **<mapper namespace="com.hust.lw.dao.UserDao">** |

C：去掉UserDaoImp.java文件

1. 使用

只需要有user.xml这个mapper文件，以及DAO接口文件，其中DAO接口文件中的接口与Mapper文件中的statement id是通过mapper文件中的namespace的全路径名关联的，然后通过sqlsession来使用。

|  |
| --- |
| **private** SqlSessionFactory **factory**; **private** UserDao **userdao**; **private** InputStream **in**; **private** SqlSession **session**;  *// 作用：在测试方法前执行这个方法* @Before **public void** setUp() **throws** Exception {  *//1.读取配置文件* **in** = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);  *//2.创建SqlSessionFactory工厂* SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  *//3.创建SqlSession工厂对象* SqlSessionFactory factory = builder.build(**in**);  *//4.使用工厂生产SqlSession对象* **session** = factory.openSession();  *//5.创建Dao接口的代理对象* **userdao** = **session**.getMapper(UserDao.**class**); }  @Test **public void** testFindUserById() {  *//5.使用代理对象执行方法* User user = **userdao**.findUserById(1);  System.***out***.println(user); } |

Mybatis的基本使用，以及来龙去脉通过这三个例子得到了一个整体的理解，随后就是一些复杂sql的mapper文件编写，以及在springboot中使用mybatis。

<https://www.cnblogs.com/benjieqiang/p/11183580.html>

## SpringMVC

### 1、环境搭建及controller与jsp互传数据（SpringMVC-1）

1. 新建SpringMVC工程



1. 部署Tomcat环境





1. 缺少Spring MVC相关类库引用





完成如上的3步后，则可以将这个空的SpringMVC工程跑起来了。

参考：

IDEA建立Spring MVC Hello World 详细入门教程

<https://www.cnblogs.com/wormday/p/8435617.html>

1. 创建controller类，找不到servlet

出现了问题： javax.servlet 包找不到

解决： 将本地 Tomcat 服务器的目录下【lib】文件夹下的 servlet-api.jar 包拷贝到工程【lib】文件夹下，添加依赖

这里面的“添加依赖”则出现了误会

参考：

cannot resolve symbol 'Servlet'

<https://blog.csdn.net/ycfn97/article/details/101114915>

1. 其他步骤，参考：

Spring MVC【入门】就这一篇！

<https://www.jianshu.com/p/91a2d0a1e45a>

说明：

A、 指定controller的方法，要么特定指定，要么泛指

|  |
| --- |
| *<bean id="simpleUrlHandlerMapping"  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  <property name="mappings">  <props>   <!-- /hello 路径的请求交给 id 为 helloController 的控制器处理-->    <prop key="/hello">helloController</prop>  </props>  </property> </bean> <bean id="helloController" class="com.hust.lw.controller.HelloController"></bean>* |

|  |
| --- |
| *<!-- 扫描ccom.hust.lw.controller下的组件 -->* <**context:component-scan base-package="com.hust.lw.controller"**/> |

如上只能选一个

B、在用模型传参时，模型中的set方法实现中，必须this.userName和this.password这样的格式

C、 jsp向后台controller传参一共有如下几种方法：

jnhs-SpringMVC jsp页面向controller传递参数的五种方式

<https://www.cnblogs.com/jnhs/p/9961589.html>

D、 Controller向前端jsp回传数据的方法：原生API完成、使用modelandview

、使用model、使用modelAttribute、map（也是一种model）

### 2、web工程转maven工程（SpringMVC-1-maven）

遇到了如下问题：

（1）pom.xml显示unknown

右键pom.xml点击maven->reimport

（2）调整目录结构

其中applicationContext.xml和\*-servlet.xml放在resources下

web.xml放在webapp/WEB-INF下

<https://blog.csdn.net/lch520baby/article/details/89374819>

（3）修改\*-servlet.xml路径

<https://blog.csdn.net/weixin_34194702/article/details/92248296>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:dispatcher-servlet.xml</param-value>

</init-param>

（4）jsp文件始终找不到，对比观察得知webapp文件夹上没有小蓝点

关于idea中新建web项目 webapp文件夹没有小蓝点 ，启动服务，访问不到解决方案

（5）获取各种路径

System.out.println(httpServletRequest.getContextPath());

System.out.println(httpServletRequest.getServletPath());

System.out.println(httpServletRequest.getRequestURI());

System.out.println(httpServletRequest.getRealPath("/"));

### 3、史上最全最强SpringMVC详细示例（SpringMVC-2）

[https://www.cnblogs.com/sunniest/p/4555801.html#commentform](https://www.cnblogs.com/sunniest/p/4555801.html" \l "commentform)

1. JSP到controller格式转换

由于前台传到controller里的值是String类型的，当往Model里Set这个值的时候，如果set的这个属性是个对象，Spring就会去找到对应的editor进行转换，然后再SET进去。

有两种：只在当前controller中生效的；在全局controller中生效的。

|  |
| --- |
| */\*\*  \* 使用InitBinder来处理Date类型的参数  \* the parameter was coverted in initBinder  \** ***@url:*** *http://localhost:8080/spring-mvc-action-tutorial/mvc/date?date=2016-05-17  \* http://localhost:8080/spring-mvc-action-tutorial/mvc/date?date=2016-05  \* http://localhost:8080/spring-mvc-action-tutorial/mvc/date?date=2016-05-17 15:00:00  \*   \** ***@param date*** *\** ***@return*** *\*/* @RequestMapping(**"/date"**) **public** String date(Date date) {  System.***out***.println(date);  **return "hello"**; }  */\*\*  \* At the time of initialization,* ***convert the type "String" to type "date"***  *\*   \** ***@param binder*** *\*/* @InitBinder **public void** initBinder(ServletRequestDataBinder binder) {  System.***out***.println(**"InitBinder of date : convert the type of String to type date."**);  binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(**new** SimpleDateFormat(**"yyyy-MM-dd"**), **true**)); } |

参考：

SpringMVC之自定义类型转换WebBindingInitializer@initBinder

<https://blog.csdn.net/zmx729618/article/details/54133830>

spring中@InitBinder和WebBindingInitializer的使用

<https://blog.csdn.net/yu870646595/article/details/51191188/>

1. 前端ajax调用到后台controller，然后返回数据到ajax

通过PrintWriter返回

|  |
| --- |
| */\*\*  \* 使用Ajax调用: pass the parameters to front-en using ajax  \* url: http://localhost:8080/spring-mvc-action-tutorial/mvc/page/ajax  \*   \** ***@param name*** *\** ***@param pw*** *\*/* @RequestMapping(value = **"/getPerson"**, method = RequestMethod.***POST***) **public void** getPerson(String name, PrintWriter pw) {  *// 也可以使用HttpServletResponse.getWriter()来获取PrintWriter* pw.write(**"hello, "** + name); } |

参考：

前后台数据交换，printwriter、jsonobject、jsonarray、ajax请求，数据交换

<https://www.hellojava.com/a/69745.html>

1. JSON在前后台之间传递

“客户端请求的数据是json字符串,springmvc不能直接解析，加@requestBody解析成对象，客户端请求的是key/value的拼接不需要加@requestBody，@responseBody返回json字符串，如果是ajax请求，我们想到得到json对象的话(为了方便根据key取value)”



对于request是json串，则必须要有@requestbody，如下图所示：



对于表单数据，则可以不需要@RequestBody：



Form-data、x-www-form-unlencoded、raw/json三者区别：







参考：

@RequestBody和@ResponseBody的用法以及Stringify()的作用

<https://blog.csdn.net/linlinlinfeng/article/details/82705772>

springmvc实现json交互 -requestBody和responseBody

<https://www.cnblogs.com/cnblog-long/p/6547380.html>

浅谈@RequestMapping @ResponseBody 和 @RequestBody 注解的用法与区别

<https://blog.csdn.net/ff906317011/article/details/78552426>

关于postman，参考如下两篇：

postman Params和Body的区别

<https://blog.csdn.net/Loren_SUN/article/details/86063374>

Postman使用详解

<https://www.cnblogs.com/xiaoxi-3-/p/7839278.html>

1. spring异常统一处理

Spring MVC异常统一处理的三种方式

<https://www.cnblogs.com/junzi2099/p/7840294.html>

1. SpringIOC和SpingMVC整合

主要就是一些配置文件的修改。



参考：（@RequestBody和@ModelAttribute两个同时使用时，以@ModelAttribute为主）

@RequestParam、@RequestBody和@ModelAttribute区别

<https://www.cnblogs.com/zeroingToOne/p/8992746.html>

@RequestParam、@RequestBody、@PathVariable区别

|  |
| --- |
| **(1)spring 中bean的作用域**  Application scope----web服务器启动后变量或实例就存在于内存中，除非重启服务，否则不会初始化。  Session Scope ----一次回话存在于内存中，就是你开一次新的网页就初始化一次。有20钟有效期！20分钟内都是有效的变量或实例，只要在20分钟内，session的变量可以随便调用（与Request Scope 的区别是，Request Scope是在请求结束后就失效了）[结合controller的session形参对象]  Request Scope ----是完成请求后，变量或实例就失效了。不能再次调用  Page Scope ----变量或实例只在当前页面有效  在web.xml中加入  <session-config>  <session-timeout>100</session-timeout>  </session-config>  以分为单位  **（2）SpringMVC模型model**  SpringMVC在调用方法前会创建一个隐含的数据模型，作为模型数据的存储容器，成为”隐含模型”。也就是说在每一次的前后台请求的时候会随带这一个背包，不管你用没有，这个背包确实是存在的，用来盛放我们请求交互传递的值。  有三种模型：**Model、ModelMap和ModelAndView**,前两者是模型，最后一个是模型+视图。  Spring中Model详解  <https://blog.csdn.net/yongwa123/article/details/85017551>   1. **使用模型Model**   @ModelAttribute最主要的作用是将数据添加到模型对象中，用于视图页面展示时使用。  @ModelAttribute等价于 model.addAttribute("attributeName", abc);  @ModelAttribute注解的使用总结  <https://blog.csdn.net/leo3070/article/details/81046383> |

1. 其他重要参考

4.1 吐血整理！14 个 Spring MVC 顶级技巧！

<https://blog.csdn.net/bntX2jSQfEHy7/article/details/104980512>

1. 多种控制类
2. 基于类的控制接口和基于方法的控制接口
3. 将对象放入模型中

4.2 SpringMVC-方法四种类型返回值总结，你用过几种？

<https://www.cnblogs.com/lenve/p/10817619.html>

1. 前后端未分离时ModelAndView用的比较多，即返回数据也返回视图
2. Void也就是各种控制跳转，String，JSON

4.3 Spring MVC教程（易佰）

## SSM

### 1、详解idea搭建SSM框架（SSM-1）

(1)完整搭建

SQLyog连接虚拟机MySQL连不上解决

<https://www.cnblogs.com/invban/p/5824891.html>

<https://www.cnblogs.com/linjiqin/p/5270938.html>

右侧没有mybatis-generator选项

<https://blog.csdn.net/yytwiligt/article/details/81010360>

详解intellij idea搭建SSM框架(spring+maven+mybatis+mysql+junit)(上)

<https://www.cnblogs.com/toutou/p/ssm_spring.html>

（2）session理解



Session：将user数据保存在服务端，通过一个jsessionid将前端和后台匹配起来，分布式后台存在session同步的问题

Token：所有的信息在token中，由前端保存，后端仅签名和验签

Postman使用jsessionid：只要是一个合法的接口，都会返回sessionid，但是这个sessionid不是合法的（不对应用户），只有在先调用login接口后返回的sessionid才是合法的，后续带着这个sessionid去访问，后台才认为合法， 否则500内部错误

参考：

SpringMVC HttpSession 与 Model 区别

<https://blog.csdn.net/Hello_ChenLiYan/article/details/106411452>

postman测试带sessionid的接口

<https://blog.csdn.net/maryshine/article/details/81014887>

### 2、前后端分离和分页查询（SSM-2）

（1）前后端分离，接收的数据是Json时需要@RequestBody作用在方法参数上.用于将POST请求数据转化为实体类.返回数据时使用@ResponseBody;

当然对于全局的也可以使用@RespController注解.前后端不分离一般使用@RequestMapping，请求方式不用管。

|  |
| --- |
| @Controller  Public class shopManagerController{  @RequestMapping(value="/getshopmanagementinfo",method=RequestMethod.GET) @ResponseBody  private Map<String,Object> getShopManagementInfo(@RequestBody User user){  } |

|  |
| --- |
| @RestController  Public class shopManagerController{  @RequestMapping(value="/getshopmanagementinfo",method=RequestMethod.GET)  private Map<String,Object> getShopManagementInfo(@RequestBody User user){  } |

（2）A:如果只是使用@RestController注解Controller，则Controller中的方法无法返回jsp页面，配置的视图解析器InternalResourceViewResolver不起作用，返回的内容就是Return 里的内容。

例如：本来应该到success.jsp页面的，则其显示success.

B:如果需要返回到指定页面，则需要用 @Controller配合视图解析器InternalResourceViewResolver才行。

C:如果需要返回JSON，XML或自定义mediaType内容到页面，则需要在对应的方法上加上@ResponseBody注解

(3)

@RestController注解相当于@ResponseBody ＋ @Controller合在一起的作用。

@GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。

@PostMapping是一个组合注解，是@RequestMapping(method = RequestMethod.POST)的缩写。

（4）示例1

|  |
| --- |
| @RequestMapping(value="hello",method=RequestMethod.*POST*) @ResponseBody Map<String, Object> hello(@RequestBody User user){  Map<String, Object> hashmap = new HashMap<String, Object>();  hashmap.put("status","success");  hashmap.put("code",200);  hashmap.put("user",user);  return hashmap; } |



（5）示例2

|  |
| --- |
| @RequestMapping(value="hello2",method=RequestMethod.*POST*) @ResponseBody RestBean hello2(@RequestBody User user){  RestBean restBean = RestBean.*ok*("this is ok");  restBean.setObj((Object)user);  return restBean; } |



SSM前后端分离/不分离对比Demo

<https://www.cnblogs.com/liruilong/p/12244925.html>

SSM框架中的前后端分离【详细说明了前后端分离的意义及示例】

<https://www.cnblogs.com/Tu9oh0st/p/10147168.html>

Jackson和data注解

<http://www.luyixian.cn/news_show_6846.aspx>

<https://blog.csdn.net/zzzghj/article/details/88828984>

## JWT

### 1、JWT封装解封装加密解密（JWT-1）

(1) JWT原理









（2）测试项目

* 创建项目



* 将src文件夹mark as source root
* Run->edit configurations

参考：

JWT 超详细分析【三类token】

<https://www.cnblogs.com/DeadBoy/p/11481146.html>

基于Token的登录流程【比较详细的登录、注册和注销流程】

<https://cloud.tencent.com/developer/article/1444727>

API鉴权及JWT详细分析【详细的比较session和token优劣势】

<https://my.oschina.net/u/4034553/blog/3063980>

JWT实现token-based会话管理

<https://www.cnblogs.com/lyzg/p/6028341.html>

Jwt入门教程：实战（ 二） | Java/.Net/Python中的使用

<https://blog.csdn.net/qq_25112523/article/details/84330053>

JWT(二)：使用 Java 实现 JWT

<https://www.cnblogs.com/lhat/p/12056759.html>

## Shiro

### 基于ini进行认证和鉴权（shiro-1）

1. user、role、permission等信息存储在ini中
2. 认证：

UsernamePasswordToken token = new UsernamePasswordToken(user.getName(), user.getPassword());//用户输入的用户名和密码

subject.login(token)//交给shiro默认的realm来进行判断

1. 鉴权（只有在认证通过后，才会鉴权，也就是用户信息已经存在subject中了）：

方式一：通过角色来判断

subject.hasRole(role);//判断此用户是否是这个角色

方式二：通过权限来判断

subject.isPermitted(permit);//判断用户是有这个权限

### 基于SSM和注解的shiro（shiro-2）

1. user、role、permission等信息存储在DB中
2. 通过自定义realm来将用户、权限及角色等信息读取出来
3. 集成到ssm框架中
4. 认证直接在自定义的realm的重写认证函数中判断密码是否相等（认证）
5. 通过注解来指明权限（鉴权）
6. 基于session来进行会话管理
7. 此套代码基于how2j的第5节完成，主要是将eclipse的普通web工程转换为了idea的maven工程

A：右键项目->Add Frameworks Support->maven

B：File->project structure->modules/facets/artifacts调整，特别是module中的web目录要选对，直到webContent文件夹上有一个小蓝点

出现如下错误：No Spring WebApplicationInitializer types detected on classpath

<https://my.oschina.net/sprouting/blog/1137851>

More than one fragment with the name [spring\_web] was found.

<http://www.mamicode.com/info-detail-2968771.html>

C：解决在service中调用mapper出异常，还没有任何日志问题，对比ssm-2，发现是applicationContext中没有配对mapper.xml的路径

|  |
| --- |
| **public class** UserServiceImpl **implements** UserService{   @Autowired UserMapper **userMapper**;    @Override  **public** String getPassword(String name) {  *//* ***TODO Auto-generated method stub*** User u = **userMapper**.getByName(name);///此行出错 |

解决办法，applicationContext.xml中修改：

|  |
| --- |
| <**bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean"**>  <**property name="typeAliasesPackage" value="com.how2java.pojo"** />  <**property name="dataSource" ref="dataSource"**/>  <**property name="mapperLocations" value="classpath:com/how2java/mapper/\*.xml"**/> </**bean**> |

D：web的根目录没有蓝色小点，需要进行如下配置：



如果基于ini，则所有的解析以及使用均由shiro框架完成了，用默认的AuthorizingRealm即可，但现在ini中的所有数据都存放到MySQL中了，则需要重写AuthorizingRealm，如下：

|  |
| --- |
| **public class** DatabaseRealm **extends** AuthorizingRealm {   @Autowired  **private** UserService **userService**;  @Autowired  **private** RoleService **roleService**;  @Autowired  **private** PermissionService **permissionService**;    @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {  *//能进入到这里，表示账号已经通过验证了* String userName =(String) principalCollection.getPrimaryPrincipal();  *//通过service获取角色和权限* Set<String> permissions = **permissionService**.listPermissions(userName);  Set<String> roles = **roleService**.listRoles(userName);    *//授权对象* **SimpleAuthorizationInfo s = new SimpleAuthorizationInfo();**  *//把通过service获取到的角色和权限放进去* s.setStringPermissions(permissions);  s.setRoles(roles);  **return** s;  }   @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  *//获取账号密码* UsernamePasswordToken t = (UsernamePasswordToken) token;  String userName= token.getPrincipal().toString();  String password= **new** String( t.getPassword());  *//获取数据库中的密码* String passwordInDB = **userService**.getPassword(userName);     *//如果为空就是账号不存在，如果不相同就是密码错误，但是都抛出AuthenticationException，而不是抛出具体错误原因，免得给破解者提供帮助信息* **if**(**null**==passwordInDB || !passwordInDB.equals(password))   **throw new** AuthenticationException();    *//认证信息里存放账号密码, getName() 是当前Realm的继承方法,通常返回当前类名 :databaseRealm* **SimpleAuthenticationInfo a = new SimpleAuthenticationInfo(userName,password,getName());**  **return** a;  } |

也就是只需要实现认证接口doGetAuthorizationInfo以及鉴权接口doGetAuthorizationInfo

注意shiro-2还是基于注解来指明权限和角色，如下：

|  |
| --- |
| **public class** PageController {    @RequestMapping(**"index"**)  **public** String index(){  **return "index"**;  }  @RequiresPermissions(**"deleteOrder"**)  @RequestMapping(**"deleteOrder"**)  **public** String deleteOrder(){  **return "deleteOrder"**;  }  @RequiresRoles(**"productManager"**)  @RequestMapping(**"deleteProduct"**)  **public** String deleteProduct(){  **return "deleteProduct"**;  }  @RequestMapping(**"listProduct"**)  **public** String listProduct(){  **return "listProduct"**;  }  @RequestMapping(value=**"/login"**,method=RequestMethod.***GET***)   **public** String login(){  **return "login"**;  }  @RequestMapping(**"unauthorized"**)  **public** String noPerms(){  **return "unauthorized"**;  }  } |

|  |
| --- |
| Apache Shiro 核心通过 Filter 来实现，就好像SpringMvc 通过DispachServlet 来主控制一样;  传统结构项目中，shiro从cookie中读取sessionId以此来维持会话，在前后端分离的项目中（也可在移动APP项目使用），我们选择在ajax的请求头中传递sessionId，因此需要重写shiro获取sessionId的方式;  Shiro提供了完整的企业级会话管理功能，不依赖底层容器（如web容器的tomcat），不管是JavaSE还是JavaEE环境都可以使用，提供了会话管理，会话监听，会话存储/持久化，容器无关的集群，失效/过期支持。对Web的透明支持，SSO单点登录的支持等特性。即使用Shiro的会话管理可以直接替换如Web容器的会话管理。 |

### 基于SSM和URL的shiro（shiro-3）

1. user、role、permission等信息存储在DB中
2. 自定义realm获取用户、角色和权限信息
3. 通过在xml中配置的密码匹配器（org.apache.shiro.authc.credential.HashedCredentialsMatcher）进行认证（认证），具体执行也是在自定义的realm的重写的认证函数中自动判断

|  |
| --- |
| @Override **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  *// 获取账号密码* UsernamePasswordToken t = (UsernamePasswordToken) token;  String userName = token.getPrincipal().toString();  *// 获取数据库中的密码* User user = **userService**.getByName(userName);  String passwordInDB = user.getPassword();  String salt = user.getSalt();  *// 认证信息里存放账号密码, getName() 是当前Realm的继承方法,通常返回当前类名 :databaseRealm  // 盐也放进去  // 这样通过applicationContext-shiro.xml里配置的 HashedCredentialsMatcher 进行自动校验* SimpleAuthenticationInfo a = **new** SimpleAuthenticationInfo(userName, passwordInDB, ByteSource.Util.*bytes*(salt),  getName());  **return** a; } |

（3）集成到ssm框架中

（4）通过url动态指明权限，其中逻辑完全由自己在URLPathMatchingFilter中实现（鉴权）

|  |
| --- |
| *<!-- url过滤器 -->* <**bean id="urlPathMatchingFilter" class="com.how2java.filter.URLPathMatchingFilter"**/>    *<!-- 配置shiro的过滤器工厂类，id- shiroFilter要和我们在web.xml中配置的过滤器一致 -->* <**bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean"**>  *<!-- 调用我们配置的权限管理器 -->* <**property name="securityManager" ref="securityManager"** />  *<!-- 配置我们的登录请求地址 -->* <**property name="loginUrl" value="/login"** />  *<!-- 如果您请求的资源不再您的权限范围，则跳转到/403请求地址 -->* <**property name="unauthorizedUrl" value="/unauthorized"** />  *<!-- 退出 -->* <**property name="filters"**>  <**util:map**>  <**entry key="logout" value-ref="logoutFilter"** />  <**entry key="url" value-ref="urlPathMatchingFilter"** />  </**util:map**>  </**property**> |

（5）基于session来进行会话管理

Shiro前3个例子总结：

认证方法：

（1）默认realm，则是login是否产生exception

（2）在自定义realm中的认证重写函数中，自己逻辑比较密码

（3）指定其他的密码比较器，由shiro自己完成比较

鉴权方法：

1. 默认realm，接口上注释标明权限
2. 自定义realm，指定url filter用自己逻辑来鉴权，鉴权逻辑之前subject.isAuthd()

A：在出错的语句上，加上try catch，将会有详细的堆栈信息

B：如下错误，至少整了3个小时

|  |
| --- |
| java.lang.IllegalArgumentException: Mapped Statements collection does not contain value for com.how2java.mapper.UserMapper.selectByExample |

一般情况下，我们用到的资源文件（各种xml，properites，xsd文件等）都放在src/main/resources下面，利用maven打包时，maven能把这些资源文件打包到相应的jar或者war里。

有时候，比如mybatis的mapper.xml文件，我们习惯把它和Mapper.java放一起，都在src/main/java下面，这样利用maven打包时，就需要修改pom.xml文件，来把mapper.xml文件一起打包进jar或者war里了，否则，这些文件不会被打包的。（maven认为src/main/java只是java的源代码路径）。

就是打完包后，在运行时环境中找不到，那就是对于xml等资源文件打包的过程中出了问题。

在pom.xml文件中，加入如下：

|  |
| --- |
| <**build**>  <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  <**include**>\*\*/\*.xml</**include**>  <**include**>\*\*/\*.properties</**include**>  <**include**>\*\*/\*.tld</**include**>  </**includes**>  <**filtering**>true</**filtering**>  </**resource**>  </**resources**> </**build**> |

参考：

Mapped Statements collection does not contain value for真正的解决办法

<https://blog.csdn.net/qq_29820901/article/details/78165938>

利用MAVEN打包时，如何包含更多的资源文件(src/main/java,src/main/resources)

<https://blog.csdn.net/jsflzhong/article/details/52077075>

### Shiro和jwt整合（shiro-4）

（1）user、role、permission等信息存储在DB中

（2）集成到ssm框架中

（3）通过url动态指明权限

（4）基于session来进行会话管理

（5）自定义了一个filter：

class JWTFilter extends BasicHttpAuthenticationFilter



其中在shiro3中用的是PathMatchingFilter

preHandle->isAccessAllowed->isLoginAttempt->executeLogin

不管哪个filter都会先执行preHandle，其中shiro3中的filter此函数就是用于鉴权，shiro-4中此函数用于加一些header的options用于跨域设置

是否允许访问函数，实现逻辑是此request中是否有token，没有token则代表允许访问：return ture，否则进行excuteLogin

在excuteLogin的逻辑中，则将token转换成shiro中的AuthenticationToken，并将其交给shiro进行login判断：

|  |
| --- |
| JWTToken token = new JWTToken(authorization); *// 提交给realm进行登入，如果错误他会抛出异常并被捕获* getSubject(request, response).login(token); |

1. 自定义realm

紧接着上面的login，则交给realm来处理，也就是重写的realm函数：

AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken auth)

在这个里面通过调用jwt的verify来验证token的合法性

注意：

在session会话管理机制中，用户登录认证的过程是在realm的doGetAuthenticationInfo进行的，甚至可以指定其他的默认密码比较器，随后其他的业务接口，则不会再进行认证，只会进行鉴权

doGetAuthorizationInfo，也就是一次认证多次鉴权；

在token会话管理机制中，用户登录认证会返回token，直接在controller层做了，随后的业务接口实际上每次都必须进行认证以及鉴权（session机制中，业务接口每次也都会认证，只是框架自动做了），所以每个业务接口都会在doGetAuthenticationInfo进行认证，也就是通过token的verify实现，而鉴权则直接在urlpathmatching的prehandler自己逻辑实现

1. 去配置化

没有配置文件，通过configuration代码类实现

|  |
| --- |
| @Configuration public class ShiroConfig {   @Bean("securityManager")  public DefaultWebSecurityManager getManager(MyRealm realm) |

参考：

Shiro + JWT + Spring Boot Restful 简易教程

<https://www.inlighting.org/archives/spring-boot-shiro-jwt>

Shiro整合JWT+Token过期刷新，全都帮你整好了（工程比较大，另外还有其他项目）

<https://www.sundayfine.com/shiro-jwt/>

JWT Token刷新方案（只有一个token自动刷新）

<https://blog.csdn.net/m0_37809141/article/details/86572697?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

APP使用token和refreshToken实现接口身份认证，保持登录状态

<https://blog.csdn.net/Paulangsky/article/details/95048410?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

参考：

Shiro学习

<https://how2j.cn/k/shiro/shiro-plan/1732.html>

SpringBoot:集成Shiro之INI认证篇【几张比较清晰的图，以及全面的系列文章】

<https://www.jianshu.com/p/766c8662a8b6>

shiro（6）- SessionMananger（操作session）【将shiro session和http session的区别讲得比较透彻，以及非常全面的系列文章】

<https://www.jianshu.com/p/a8ab2d1fb61a>

shiro（12）-session与token（有状态认证PK无状态认证）【有一张图，将shiro中关于session认证和鉴权的过程详细描述】

<https://www.jianshu.com/p/07dfa6c899bc>

<https://www.jianshu.com/p/484c039930e2>

Mysql

## user微服务（ssm-2-projectname）

1. interface和class区别

interface不用指定成员变量public，也一样能访问;

Interface中定义的是常量，而class中可以定义static的变量，也就是全局变量。

2、@EqualsAndHashCode和@JsonProperty

@EqualsAndHashCode(callSuper = true)

equals和HashCode深入理解以及Hash算法原理

<https://blog.csdn.net/qq_21688757/article/details/53067814?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

@JsonProperty("loginName")-----进一步做下实验，看是否与父类CheckParaResult有关

Jackson has the @JsonProperty("name") annotation, which can be applied to methods - the return value of the method will be assigned to the "name" parameter in the JSON.

<https://stackoverflow.com/questions/31381439/is-jsonproperty-annotation-required-on-accessor-methods?r=SearchResults>

（有点明白了，相当于告诉jackson，这个check函数是某个property的）,以及JsonIgnore

3、mapper中user和userexample区别

<https://blog.csdn.net/biandous/article/details/65630783>

4、分页

import com.github.pagehelper.PageHelper;  
import com.github.pagehelper.PageInfo;

浅析pagehelper分页原理【单步调试到最终的SQL语句】

<https://blog.csdn.net/qq_21996541/article/details/79796117>

Github PageHelper 原理解析

<https://www.cnblogs.com/gkmeteor/p/11900984.html>

MyBatis 分页插件 - PageHelper

<https://github.com/pagehelper/Mybatis-PageHelper/blob/master/wikis/zh/HowToUse.md>

5、事务

import org.springframework.transaction.annotation.Transactional;  
import org.springframework.transaction.interceptor.TransactionAspectSupport;

@Transactional注解应该只被应用到 public 方法上，这是由Spring AOP的本质决定的。

@Transactional注解内部实现依赖于Spring AOP编程。而AOP在默认情况下，只有来自外部的方法调用才会被AOP代理捕获，也就是，类内部方法调用本类内部的其他方法并不会引起事务行为。

Spring 事务 -- @Transactional的使用【特别是后面函数间调用以及事务注释的关系，以及事务注释的各参数含义】

<https://www.jianshu.com/p/befc2d73e487>

6、通过header来区分手机还是web用户

7、合法性校验

dto的合法性校验中，即使方法体为空，也可能导致收到的为空

必须将这个合法性校验的方法去掉才行，而不仅仅是置空方法体。

8、WEB中的Filter和SpringMVC中的拦截器intercept区别

Filter中autoware service

<https://blog.csdn.net/jsloveyou/article/details/79459401>

Java中三大器 | 监听器、过滤器、拦截器

<https://www.jianshu.com/p/1ce89f9c6f8d>

9、Filter

Filter及FilterChain的使用详解

https://blog.csdn.net/zhaozheng7758/article/details/6105749?utm\_medium=distribute.pc\_relevant\_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth\_1-utm\_source=distribute.pc\_relevant\_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase

重写Request，修改Request中的Body内容

https://www.jianshu.com/p/47de9e15b2b7?utm\_campaign=hugo

Servlet之Filter详细讲解

https://www.cnblogs.com/zlbx/p/4888312.html

解决HttpServletRequest的流只能读取一次的问题

https://blog.csdn.net/dsfllx/article/details/99081384?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase

10、@RefreshScope

SpringCloud配置热更新@RefreshScope,以及没有出现/refresh的动态刷新地址,访问404的解决办法

<https://blog.csdn.net/zzzgd_666/article/details/84322947>

@RefreshScope 配置文件自动刷新

<https://blog.csdn.net/Anenan/article/details/85134208>

11、权限管理实现：注解+SpringAOP

spring AOP + 自定义注解实现权限控制小例子

<https://blog.csdn.net/Minhellic/article/details/51919690>

【Spring-AOP-学习笔记-7】@Around增强处理简单示例

<https://www.cnblogs.com/ssslinppp/p/5845659.html>

20、用户微服务

20.1 接口描述

（1）获取验证码（/user/sms-code/no-token/get）

A：服务端生成、发送并保存验证码（以loginname为索引）

B：服务端生成RSA公私钥，保存私钥，公钥返回（以loginname为索引）

C：服务端返回签名：md5(公钥#动态码)

（2）注册（/user/register）

A：客户端处理密码：RSA(password, rsapublickey)

B：客户端生成sign：MD5(password#code)

C：服务端根据用户名从redis中取出rsaprivatekey和code，并验证签名

1. 登录（/user/login）

A：客户端生成password签名：md5(Nonce#Time#loginName#md5(password))

B：服务端根据用户名从MySQL取出密码，并验证password签名

C：服务端生成token和sid返回

1. 获取用户信息（/user/info）

A：客户端把token信息带上并请求用户信息

B：服务端在filter中对token进行验签，同时补充request中的tokenDataDto和uri（便于后续使用），通过则放行

<https://www.kancloud.cn/zldev/apidoc/76361>

<http://api.okayapi.com/docs.html>

接口参考：

20.2 一步步

（1）dpikey理解：一个dpikeyinfo，代表1个key，然后key#time生成动态码，以动态码作为后台RSA的key，dpikeypackage则是管理多个dpikey

（2）上述4个接口，涉及filter、md5和RSA加解密

（3）权限管理TDResult：注释+springAOP，并且利用到了验签的结果tokenDataDto

（4）token续签，基本上就是用在token有效期T\*2的时间内，还可以重新生成新的token，相当于客户端要记录token的状态（在快过期时重新申请token），服务端也有魔鬼处理，在T\*2内，收到/token/renew请求，则判断token是否在T\*2内，如果是则重新生成token；

而对于非/token/renew接口，则直接验签token看过期T

20.3关键点

（1）网关里面有一个filter用于token鉴权，有一个tokenUtil，也有一个tokenservice

（2）TdResult通过切面实现的注释，来表明执行哪些controller接口需要哪个权限，以及是否需要token。其实在这里面实现鉴权不合理，应该是在网关里面实现，既然会在网关里实现身份认证，其实也可以一起将鉴权一起做了（认证，鉴权，权限管理）

（3）Dpikey

（4）sid理解(结合token续签)

## oAuth2.0

SpringBoot项目的Edit Configuration是新建springboot，而不是local tomcat，当然也可以让springboot项目跑在Local Tomcat里面。

使用Springboot搭建OAuth2.0 Server的方法示例【基于postman将流程跑通了，但postman封装了内部逻辑，无法知道细节】

<https://www.jb51.net/article/145859.htm>

springboot2整合OAuth2.0认证实例【详细的步骤，但基于上面的demo，取token时提示400 bad request】

<https://blog.csdn.net/haoxiaoyong1014/article/details/80795574>

理解OAuth 2.0【这个最基础】

<http://www.ruanyifeng.com/blog/2014/05/oauth_2_0.html>

## SpringBoot

### springboot-1【helloworld】

1. 通过idea的spring initializr来创建springboot项目，全部为空
2. 在pom.xml中增加

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

1. add junit to class path
2. Import三个特殊类

|  |  |
| --- | --- |
| |  | | --- | | **import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*; **import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*; **import static** org.hamcrest.Matchers.*equalTo*; | |

1. 单元mvc为null，debug发现是@before注释下的setup()没有执行，原因是junit版本不对，@before应该改为@beforeeach

|  |
| --- |
| Junit测试中@Before不执行  错误原因：  Junit的版本原因  在Junit4下使用@Before和@After  而在Junit5下使用@BeforeEach和@AfterEach |

1. /user接口中，request是user的json，返回的也是user的json
2. /users接口中，将数据存储在static变量中（并且是线程安全的），模拟数据库的增删改查

参考：

Spring Boot 2.x基础教程：快速入门

<http://blog.didispace.com/spring-boot-learning-21-1-1/>

Spring Boot 2.x基础教程：构建RESTful API与单元测试

<http://blog.didispace.com/spring-boot-learning-21-2-1/>

### springboot-2【配置读取】

1. 多环境运行：spring.profiles.active=dev
2. 配置参数注入：@Value
3. 在Spring Boot 2.0中对配置属性加载的时候会除了像1.x版本时候那样移除特殊字符外，还会将配置均以全小写的方式进行匹配和加载。
4. 使用全小写配合-分隔符的方式来配置
5. 读取配置文件中的List和Map，见“UserConfig.java”、hello2和hello3
6. 环境变量和系统属性不考虑
7. 将配置文件中的List和Map独立取出来，而不是通过一个类来映射（区别于UserConfig.java），将context和binder定义为全局变量，这样可以在各个controller接口处使用，具体见hello3、hello4和hello5

参考：

springboot读取.properties配置文件中的map和list类型配置参数

<https://blog.csdn.net/u014338530/article/details/86136986>

Spring Boot属性配置文件详解

<http://blog.didispace.com/springbootproperties/>

Spring Boot 2.x基础教程：配置文件详解

<http://blog.didispace.com/spring-boot-learning-21-1-3/>

### springboot-3【请求参数校验和Swagger】

1. 基于swagger-spring-boot-starter更方便集成swagger，在配置文件application.properties中指定扫描路径swagger.base-package
2. 主要的swagger标签：@API, @APIModel 、@APIModelProperty,@APIOperation
3. 在pom中引入校验的依赖包

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

（4）对于非NotNULL的字段，则在Request报文中，可有可无，但是对于NotNULL的字段，则必须在Request中填上

（5）合法性校验遗留：六：user微服务，第7点

参考：

Spring Boot 2.x基础教程：JSR-303实现请求参数校验

<http://blog.didispace.com/spring-boot-learning-21-2-3/>

Spring Boot 2.x基础教程：使用Swagger2构建强大的API文档

<http://blog.didispace.com/spring-boot-learning-21-2-2/>

Springboot中的一些注释的详解 Swagger（@ApiOperation和@ApiModelProperty）

<https://blog.csdn.net/qq_35069673/article/details/88825824>

利用Swagger自动生成漂亮的静态开发文档说明页

<https://blog.csdn.net/yw8886484/article/details/100622506?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase>

### springboot-4【序列化和反序列化】

1. 定义： 序列化代表一致和统一，则是二进制流或者是json；反序列化则是很混乱，则是各种对象。

将JSON解析为Java对象也称为从JSON反序列化Java对象；

从Java对象生成JSON也称为将Java对象序列化为JSON。

1. Jackson 最常用的 API 就是基于”对象绑定” 的 ObjectMapper，之所以称为ObjectMapper是因为它将JSON映射到Java对象（反序列化），或者将Java对象映射到JSON（序列化）。
2. 反序列化（Request的过程）：默认情况下，Jackson通过将JSON字段的名称与Java对象中的getter和setter方法进行匹配，将JSON对象的字段映射到Java对象中的属性。 Jackson删除了getter和setter方法名称的“ get”和“ set”部分，并将其余名称的第一个字符转换为小写。
3. 序列化（Response的过程）：@JsonIgnore表示在序列化过程中忽略此字段
4. 属性重命名：

jackson中@JsonProperty("trueName")，fastjson中@JSONField(name="MessageType")；

如果对象属性首字母大写，则在反序列化时（Request），无法匹配上，因为找不到小写的；

在序列化时，对象的属性名首字母由大写自动改为小写。

具体见User对象的属性Address。

1. 时间字段LocalDateTime

|  |
| --- |
| **private** LocalDateTime **dateTime**;  @JsonProperty(**"dateTime"**) **private** String getDateTimeStr() {  DateTimeFormatter df = DateTimeFormatter.*ofPattern*(**"yyyy-MM-dd HH:mm:ss"**);  **return** df.format(**dateTime**); }  @JsonProperty(**"dateTime"**) **private void** setDateTimeStr(String dateTimeStr) {  DateTimeFormatter df = DateTimeFormatter.*ofPattern*(**"yyyy-MM-dd HH:mm:ss"**);  **dateTime** = LocalDateTime.*parse*(dateTimeStr,df); } |

重点理解：在反序列化时，JSON字符串中字段与对象属性字段匹配的过程，是通过set/get方法，也就是说JSON字段中的dateTime字段对应着User对象中的dateTime属性（通过@JsonProperty指定，如果在SET/GET函数上没有此注释，则通过上述默认的匹配方法，去掉set/get，然后首字段转小写就是对应的User对象的属性名：dateTimeStr），然后通过指定的这两个SET/GET函数，来将JSON中dateTime的值SET给User对象，以及GET 此User对象此属性值然后给JSON字串。

参考：

Jackson使用详解

<https://www.lizenghai.com/archives/97079.html>

对象转json字符串，属性首字母大写自动变为小写

<https://blog.csdn.net/pb951028/article/details/104863866/>

### springboot-5【Controller的统一异常处理】

1. 对于Spring MVC项目，事务配置在Service层，当数据库操作失败时让Service层抛出运行时异常，Spring事务管理器就会进行回滚。

如此一来，我们的 Controller 层就不得不进行 try-catch Service 层的异常，否则会返回一些不友好的错误信息到客户端。但是，Controller 层每个方法体都写一些模板化的 try-catch 的代码，很难看也难维护，特别是还需要对 Service 层的不同异常进行不同处理的时候。

（2）@ControllerAdvice顾名思义就是controller的增强器，**@ControllerAdvice注解将作用在所有注解了@RequestMapping的控制器的方法上**。(如果全部异常处理返回json，那么可以使用 @RestControllerAdvice 代替 @ControllerAdvice ，这样在方法上就可以不需要添加 @ResponseBody。)

（3）优点：将 Controller 层的异常和数据校验的异常进行统一处理，减少模板代码，减少编码量，提升扩展性和可维护性。

缺点：只能处理 Controller 层未捕获（往外抛）的异常，对于 Interceptor（拦截器）层的异常，Spring 框架层的异常，就无能为力了。

（4）CommonEnum实际上就是错误码code和错误消息msg的映射列表（继承接口BaseErrorInfoInterface），在自定义异常类BizException以及统一的response类ResultBody中（相当于TDResponse），都有用到。

@ControllerAdvice + @ExceptionHandler 实现controller层面的统一的异常捕获以及处理返回。

参考：

Spring Boot中Web应用的统一异常处理

<http://blog.didispace.com/springbootexception/>

SpringBoot优雅的全局异常处理

<https://www.cnblogs.com/xuwujing/p/10933082.html>

SpringMVC 中 @ControllerAdvice 注解的三种使用场景！

<https://www.cnblogs.com/lenve/p/10748453.html>

### springboot-6【Jdbctemplate及多数据源】

1. @Qualifier用于指定自动注入的bean（当有多个bean，会让编译器产生困惑时，则需要此注解）
2. 单数据源和多数据源的配置如下：

|  |
| --- |
| *#spring.datasource.url=jdbc:mysql://127.0.0.1:3308/mybatis #spring.datasource.username=root #spring.datasource.password=123456 #spring.datasource.driver-class-name=com.mysql.jdbc.Driver* **spring.datasource.primary.jdbc-url**=**jdbc:mysql://127.0.0.1:3308/mybatis spring.datasource.primary.username**=**root spring.datasource.primary.password**=**123456 spring.datasource.primary.driver-class-name**=**com.mysql.jdbc.Driver  spring.datasource.secondary.jdbc-url**=**jdbc:mysql://127.0.0.1:3308/mybatis2 spring.datasource.secondary.username**=**root spring.datasource.secondary.password**=**123456 spring.datasource.secondary.driver-class-name**=**com.mysql.jdbc.Driver** |

（3）关键的配置工作，则是在配置类DataSourceConfiguration中实现，然后在service实现类中，自动注入primary或secondary的jdbctemplate（名称与配置类中的bean名称一致即可）

参考：

Spring Boot 2.x基础教程：JdbcTemplate的多数据源配置

<http://blog.didispace.com/spring-boot-learning-21-3-7/>

Spring Boot 2.x基础教程：使用JdbcTemplate访问MySQL数据库

<http://blog.didispace.com/spring-boot-learning-21-3-1/>

### springboot-7【Druid数据源及其监控】

1. 数据源实际上就是数据库连接池，SpringBoot 在JDBC模块中自动化配置使用的默认数据库连接池是当前性能最佳的HikariCP。本节演示的阿里巴巴开源的国产的开源数据库连接池Druid，其也是一个高性能的数据库连接池，最关键的是自带了监控。
2. <http://localhost:8080/druid/sql.html>



参考：

Spring Boot 2.x基础教程：使用国产数据库连接池Druid

<http://blog.didispace.com/spring-boot-learning-21-3-3/>

### springboot-8【Mybatis XML方式，多数据源不考虑】

（1）配置文件中指定mapper.xml位置

mybatis.mapper-locations=classpath:mapper/\*.xml

（2）主程序中指定mapper.java目录位置

@MapperScan("com.hust.lw.mapper")

参考：

Spring Boot 2.x基础教程：使用MyBatis的XML配置方式

<http://blog.didispace.com/spring-boot-learning-21-3-6/>

Spring Boot 2.x基础教程：MyBatis的多数据源配置

<http://blog.didispace.com/spring-boot-learning-21-3-9/>

### springboot-9【定时任务以及异步任务】

1. @EnableScheduling和@Scheduled(fixedRate = 5000)：定时执行任务见ScheduledTasks类
2. 同步任务不需要特定的注解，只需要@Component，将其注解为bean，便于@Autowared自动注入，见SyncTasks类和LwApplicationTests#testSyncTasks方法
3. 异步任务，需要在主函数@EnableAsync，然后在类中的各方法函数上@Async，同时将异步类@Component，以便@Autowared自动注入 ，可以看到三个任务调用的顺序都是不可控的，也不知道调用的结果，必须Sleep足够时长，以保证三个任务都跑完了
4. 异步任务的异步回调，当异步任务执行完时，会回调一个函数将结果返回，异步回调的函数返回值类型：Future<String>

其成员函数：isDone()判断异步任务是否已经执行完成，isCanceld()判断异步任务是否被取消，

Get()获取异步任务的返回值，具体见见FutureAsyncTasks类和LwApplicationTests#testFutureAsyncTasks方法

（5）上面的异步任务都是在框架默认的线程池中执行的，其实可以自定义线程池，有两种方法：ThreadPoolTaskScheduler和ThreadPoolTaskExecutor，两者差不多，只是前者比后者多继承一个父类TaskSchedulered，可以做定时，以及优雅的关闭线程池。通过打印线程名，确认确实是已经在自建的线程池中运行各个异步任务。具体见见SelfPoolAsyncTasks类和LwApplicationTests#testFutureAsyncTasks方法

### springboot-10【日志管理】

1. slf4j+logback，应用->日志抽象层->日志实现，springboot底层也是采用slf4j+logback实现
2. 写在《file》tag里面的表示当前日志的名字，不接受日期pattern的。只有当时间roll了以后才会出现带时间的归档日志文件。
3. @Slf4j注解在每个类上，可以省掉：

static final Logger logger = LoggerFactory.getLogger(LwApplication.class);

然后直接用log.debug、log.info、log.error来输出日志

1. AOP统一处理WEB请求日志

A：切入点定义方法

|  |
| --- |
| 1）execution(\* \*(..))  //表示匹配所有方法  2）execution(public \* com. savage.service.UserService.\*(..))  //表示匹配com.savage.server.UserService中所有的公有方法  3）execution(\* com.savage.server..\*.\*(..))  //表示匹配com.savage.server包及其子包下的所有方法 |

B：由于controller中的方法未定义为public，导致AOP始终不生效

1. 动态修改日志级别

A：pom.xml文件中添加actuator依赖

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

B：打开loggers端点以及关闭访问actuator的权限验证

|  |
| --- |
| **management.security.enabled**=**false management.endpoints.web.exposure.include**=**loggers** |

C：获取和动态设置日志级别

|  |
| --- |
| GET [http://127.0.0.1:8080/actuator/loggers](http://127.0.0.1:8080/actuator/loggers/com.hust.lw)  GET <http://127.0.0.1:8080/actuator/loggers/com.hust.lw>  POST <http://127.0.0.1:8080/actuator/loggers/com.hust.lw>  {      "configuredLevel":"DEBUG",      "effectiveLevel":"DEBUG"  } |

1. Actuator监控端点，打开所有的端点（/autoconfig重命名为/conditions）

|  |
| --- |
| **management.endpoints.web.exposure.include**=**\*** |

使用：

|  |
| --- |
| <http://127.0.0.1:8080/actuator/metrics/>  <http://127.0.0.1:8080/actuator/metrics/process.uptime>  <http://127.0.0.1:8080/actuator/conditions> |

1. 点击右侧Maven Projects

选择lifecircle中clean->compile->package

参考：

maven项目中使用logback+slf4j（详细的logback.xml配置）

<https://www.jianshu.com/p/43c31c28e766>

spring boot集成日志（slf4j+logback）以及其他日志替换spring boot默认log框架（日志抽象、日志实现、以及各种组合）

<https://blog.csdn.net/ckyblack/article/details/102564281>

Spring Boot Web中的AOP不生效

<https://www.imooc.com/wenda/detail/541556>

Spring Boot中使用AOP统一处理Web请求日志

<http://blog.didispace.com/springbootaoplog/>

Spring Boot 2动态修改日志级别

<https://my.oschina.net/eacdy/blog/3040211>

Spring Boot Actuator监控端点小结

<http://blog.didispace.com/spring-boot-actuator-1/>

Springboot如何打包部署项目【详细的打包过程以及后台运行的nohup命令详解】

<https://blog.csdn.net/qq_34491508/article/details/91490434>

Spring Boot应用的后台运行配置【windows和linux的后台运行方式】

<http://blog.didispace.com/spring-boot-run-backend/>

## SpringCloud

### springcloud-1

1. springcloud依赖包

A：springcloud的依赖

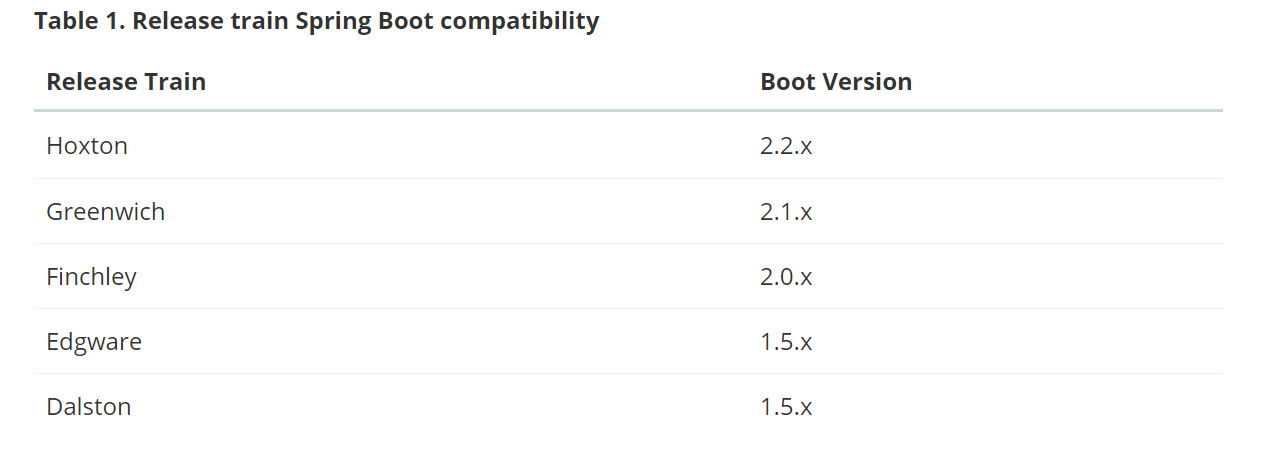
|  |
| --- |
| <**dependencyManagement**>  <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-dependencies</**artifactId**>  <**version**>Hoxton.RELEASE</**version**>  <**type**>pom</**type**>  <**scope**>import</**scope**>  </**dependency**>  </**dependencies**> </**dependencyManagement**> |

B：erueka server的依赖

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-netflix-eureka-server</**artifactId**> </**dependency**> |

C：springcloud和springboot的版本兼容关系

提示错误：“The following method did not exist: org.springframework.boot.actuate.health”



Springboot的版本配置为2.2.8.RELEASE

|  |
| --- |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>2.2.8.RELEASE</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**> |

1. 服务注册不上eruake

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency> |

不能是spring-cloud-netflix-eureka-client

另外：

是因为没有指定依赖的版本号（不知道为什么SpringCloud其他依赖都默认有版本号维护，但是这个没有，所以自己手动加一下）

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  <**version**>2.0.2.RELEASE</**version**> </**dependency**> |

Eurake client只需要上面这个依赖即可。

1. 消费者

A：始终启动后就退出，pom配置有问题

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-web</**artifactId**>  <**version**>RELEASE</**version**>  <**scope**>compile</**scope**> </**dependency**>  =============》  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**> </**dependency**>  和produce一样，只需要有上面这个dependency即可 |

B：HTTP接口调用方法如下：

HttpClient、Okhttp、Httpurlconnection、RestTemplate

C：Resttemplate和Feign区别（看下LFC采用的HTTP调用方式）

D：启动两个produce，然后consume不停调用http://127.0.0.1:3334/add，打断点判断调用到哪个produce了

java -jar -Dserver.port=2224 D:\springcloud-1\produce\target\produce-0.0.1-SNAPSHOT.jar

1. 消费者FeignConsume

|  |
| --- |
| @FeignClient(**"computer-service"**) **public interface** ComputerClient {  @RequestMapping(method = RequestMethod.***GET***, value = **"/add"**)  Integer add(@RequestParam(value = **"a"**) Integer a, @RequestParam(value = **"b"**) Integer b); } |
| @RestController **public class** ConsumeController {   @Autowired  ComputerClient **computerClient**; |

Autowired处为红色，不影响编译和运行

参考：

springcloud-eureka启动报错，提示The following method did not exist: org.springframework.boot.actuate.health

<https://blog.csdn.net/cmqwan/article/details/89410114>

使用Maven构建多模块层级项目

<https://blog.csdn.net/luoxiang183/article/details/76215635>

32.微服务之间的调用方式RestTemplate和FeignClient

<https://blog.csdn.net/u014590757/article/details/80233215>

1、整体过一遍

2、梳理需要形成demo的列表，及各个demo的说明

3、逐个去实现

1. swagger还是未理解，则需要对着lfc逐个去看下[@API, @APIModel 、@APIModelProperty,@APIOperation]，以及其他静态文件

[spring-boot-starter-swagger](https://github.com/SpringForAll/spring-boot-starter-swagger)到底做什么用的【比原生的swagger2使用方法要简单】

1. 入门：helloworld及单元测试，结合快速入门及编写单元测试
2. 配置文件，多个环境的，以及取不同位置的配置参数的方法，结合配置绑定2.0全解析
3. 参数校验，bean valicate，其中hiberate vallicate是其实现
4. Jdbctemplate实例
5. 数据源可以好好再理解下，实际上就是数据库连接池
6. Druid数据源连接池监控
7. Mybatis多数据源配置
8. 统一异常处理【control+exceptionhandler实现，对比lfc实现(没有实现全局的异常处理)，springaop advice增强】

<https://www.cnblogs.com/xuwujing/p/10933082.html>

SpringBoot优雅的全局异常处理

<https://www.cnblogs.com/lenve/p/10748453.html>

SpringMVC 中 @ControllerAdvice 注解的三种使用场景！

1. 时间日期序列化和反序列化，对比lfc实现

<https://www.cnblogs.com/carrychan/p/9883172.html>

LocalDateTime在spring boot中的格式化配置

|  |
| --- |
| import com.fasterxml.jackson.annotation.JsonProperty; import com.tdcloud.base.util.PublicUtil; import lombok.Data;  import java.time.LocalDateTime; @Data public class Log {  private Long id;   private Integer type;   private Long orderId;   private String content;   private Integer status;   private Long operator;   private LocalDateTime createTime;   @JsonProperty("createTime")  private String getCreateTimeStr() {  if (PublicUtil.isNotEmpty(createTime)) {  return String.valueOf(PublicUtil.getUTCTimeMillis(createTime));  }  return null;  }   @JsonProperty("createTime")  private void setCreateTimeStr(String createTime) {  this.createTime = PublicUtil.getUTCDate(Long.*parseLong*(createTime));  } } |

（11）序列化和反序列化 jackson：json字符串、json、对象之间相互转换

@JSONField

@JSONIgnore

（12）主键自动增长，在User bean中，以及Request中，如何处理这个唯一且自动增加的键

|  |
| --- |
| @Id  @GeneratedValue  private Long id; |

Idea破解

<https://www.w3h5.com/post/171.html>

<https://blog.csdn.net/qq_42262675/article/details/84934047?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>



API管理平台（第三方github登录）

[https://www.eolinker.com/](https://www.eolinker.com/" \l "/?status=link-jump)

Controller参数检验，必填、选填

<https://blog.csdn.net/dh554112075/article/details/80790464>

Header中鉴权key：Authorization ，以及各种类型