1. **前言**

**1.最近计划**

**2.最终计划**

* + - 1. 以项目为线索，以单个知识点为驱动，以效率为准则，以All为目标；
      2. 保证本项目最多中断1天，不管当时遇到了怎样的打击，心情怎样低落，拿出高三那年的状态对待此事；
      3. 时间不等人，不能到30岁还在小企业混，不稳定，摆脱复制粘贴的code状态；
      4. 工作学习的核心时间禁止看视频，直播，游戏，手机等等，不想与java无关的事，要成功自律最重要，抵制诱惑；

**3.项目目标**

**4.项目配置**

* + - 1. jdk1.8.0\_131，mysql5.5， hosts更改，Redis开启，Gitbook开启，，

**5.编码规约**

**6.快捷按键**

* + - 1. ;，，，，，

**7.CSS属性**

* + - 1. ，，，，，，，，，，，，，，，，，，

**8.特别感谢**

**9.技术选型**

1. **V1.0开发**
   * + 1. 由于之前电脑太卡，重装了系统，是大地系统ghost的win10版本；先制作了启动盘，然后把系统镜像放在U盘上，之后重启电脑，F12用u盘启动，选择好系统，即可安装；注意安装过程只与C盘有关，其他盘的数据不受影响，但之前安装的软件都会失效了，因为注册表没有添加到新的系统中；系统一直存在激活的问题，最后在淘宝买了注册码完事。
       2. 按步骤搭建zheng的开发环境，首先安装jdk1.8.0\_131，就是安装后配置一下环境变量；安装MySQL5.5；安装maven，配置环境变量，settings.xml里加入本地仓库地址与阿里云的镜像；安装git与TortoiseGit这两个是一起使用的，前者是命令行后者是客户端；安装idea作为开发工具；安装Redis，是一种数据库支持高并发注意把启动方式改为手动；安装Zookeeper，是用来支持分布式架构的，安装好修改zoo.cfg文件名，zkServer.cmd启动服务；安装ActiveMQ，activemq.bat启动；安装nginx；
       3. 电脑移动文件时会复制，而不是单纯的移动，编写：新建文本文档.reg 文件，内容是：

Windows Registry Editor Version 5.00

[HKEY\_CLASSES\_ROOT\\*]

"DefaultDropEffect"=dword:00000002

[HKEY\_CLASSES\_ROOT\AllFilesystemObjects]

"DefaultDropEffect"=dword:00000002

然后双击运行文件，就可以了，这个动作就是向注册表中修改信息的过程。

* + - 1. 新建mtx工程，外壳是个空的Maven工程，上传GitHub，PRs Welcome的意思是欢迎其他人提交代码到我的项目，我来主导合并；搭好框架后从自动生成system代码开始做；项目有两个数据库，主库用来操作数据，从库用来查询数据；代码生成完毕配置springmvc准备开始搭建页面，包括spring；
      2. Maven控制台输出中文乱码解决： Setting->maven->runner VMoptions:-Dfile.encoding=GB2312；BigDecimal在商业计算中用到，用来计算小数的，更加精确；Preconditions类似断言，减少if语句，错了直接抛出异常；
      3. 报错：Caused by: org.springframework.beans.BeanInstantiationException: Failed to instantiate [org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter]: Constructor threw exception; nested exception is java.lang.NoSuchMethodError: com.fasterxml.jackson.annotation.JsonFormat$Value.empty()Lcom/fasterxml/jackson/annotation/JsonFormat$Value;

最终发现是jar包冲突导致的，以后遇到类似问题应该早点察觉。

* + - 1. 之后开始调试单元测试的内容；决定使用领域驱动，DO为与DB交互模型，VO是页面展示模型，DTO是保存修改模型，使用Dozer进行模型间的类型转换；lang3是Apache Commons 团队发布的工具包，要求jdk版本在1.5以上，lang是老版本的；break;结束整个循环，continue;结束此次循环；
      2. 登录时提交用户信息到服务器，服务器通过HTTP请求头返回给浏览器，浏览器保存cookie信息，之后再次请求服务器，请求头中会包含cookie信息，服务器依此判断；cookie是有时效的；
      3. session存放在服务器上，发送到浏览器的只有sessionId，浏览器再请求时会将这个id带上，服务器根据id找到Session，session也是有时效的，可以在web.xml设置时间，JSESSIONID就是sessionId，浏览器打开新窗口会产生两个session；
      4. jsp页面上的错误没有被spring捕获，会因为web.xml的配置转到错误页面；controller层捕获的错误会到统一异常处理，最后也是跳转到错误页面；
      5. Filter过滤器需要在web.xml中配置，依赖于Servlet；Interceptor拦截器需要在SpringMVC中配置，依赖于框架类似AOP编程；Filter的执行顺序在Interceptor之前；Filter主要是针对URL地址做一个编码的事情、过滤掉没用的参数等；太细的话，还是建议用interceptor；
      6. 由于项目调试过于麻烦，暂时取消rpc方式，并使用idea的JRebel插件热部署开发；首先hotswap 开启热更新，server项目热部署打钩，将jetty的热部署关闭；jetty:run-war先把项目打成war包再运行，启动时间太久；
      7. 由于父项目修改后不能热部署，因为子项目是以jar包的形式调用的，决定使用jrebel插件并关闭jetty的热更新，jrebel所有项目都要勾选；bootstrapValidator使用要把<div class="form-group"></div> 放在输入框外面包住，不然无法生效；在ajax的beforeSend方法上加一个load，用来防止重复提交数据；
      8. Dic信息由于不容易变，放入ehcache缓存中；mybatis-plus与pagehelper存在冲突，同时只允许一个存在，我选择plus，注意-jdbc文件要做出对应的配置；jrebel插件有点优秀，但每过180天会要求重新激活，注意一下；bean注入到spring，不能通过new取出来，注意着点；缓存就是Map类型，存放在系统temp文件夹里，需要的时候取出来即可；
      9. 开发环境切换配置有两处1：maven，2：spring的web.xml；
      10. 使用Redis存储数据，Redis Desktop Manager为其可视化工具，注意开启此DB需要开启一个bat窗口；因为现在要做的是一个项目簇，所以使用SSO单点登录，共享sessionId，所以用redis保存sessionId；
      11. <setting name="cacheEnabled" value="false"/><!-- 开启缓存 -->开启的是mybatis的缓存,测试开发中就暂时关闭了

1. **代码记忆**

**public abstract class** BaseServiceImpl<Mapper, Record, Example>//向类中传递泛型，利用反射简化代码

*//ThreadLocal用于保证：数据源名称 这个变量不被其它线程干扰，不可能两个线程同时在修改数据源***private static final** ThreadLocal<String> ***CONTEXT\_HOLDER*** = **new** ThreadLocal<>(); 线程局部变量

*// 主库****MASTER***(**"masterDataSource"**, **true**),  
*// 从库****SLAVE***(**"slaveDataSource"**, **false**),; 先写好枚举类的构造方法即可

**for** (DataSourceEnum dataSourceEnum : DataSourceEnum.*values*()) { //通过循环遍历枚举类

DataSourceEnum.***MASTER***.getDefault() // DataSourceEnum.MASTER相当于 对象 ，这样可以调用枚举类里的方法

Method countByExample = **mapper**.getClass().getDeclaredMethod(**"countByExample"**, example.getClass());  
Object result = countByExample.invoke(**mapper**, example); //先获取类的方法example.getClass()是传入参数类型，再执行这个方法example是传入的参数，result是结果

**public static** <T> T getBean(Class<T> clazz) { //<T> T 可以看做固定写法，表示返回该类型的 对象，这个就是泛型方法

SpringContextUtil.*getBean*(**mapper**.getClass()); //方法的调用

*// 打印的结果是：泛型参数String 是第一个泛型参数***class** AnonymousArrayListA **extends** HashMap<String,Integer> {  
  
}  
AnonymousArrayListA aa = **new** AnonymousArrayListA();  
Class typeA = (Class) ((ParameterizedType) aa.getClass().getGenericSuperclass()).getActualTypeArguments()[0];  
System.***out***.println(typeA);

*///F:/All/1workspace/mtx/mtx-common/target/test-classes/ 获取类的真实路径****log***.info(MybatisGeneratorUtil.**class**.getResource(**"/"**).getPath());  
*//F:/All/1workspace/mtx/mtx-common/target/classes/template/generatorConfig.vm****log***.info(MybatisGeneratorUtil.**class**.getResource(**"/template/generatorConfig.vm"**).getPath().replaceFirst(**"/"**, **""**));

@Target({ElementType.***TYPE***})  
@Retention(RetentionPolicy.***RUNTIME***)

@Inherited*//允许子类继承父类的注解*  
@Documented  
**public** @**interface** BaseService {  
} 先定义一个注解

@Transactional(rollbackFor = Exception.**class**)*//当你的方法中抛出异常时，它会将事务回滚，数据库中的数据将不会改变*

一般都要加上这个

<**bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"**>  
 <**property name="location" value="classpath:common.properties"** />  
</**bean**>*<!-- xml引用 properties -->*

<**dubbo:registry file="${dubbo.path}/mtx-system-server-dubbo.cache" address="zookeeper://zkserver:2181"**/>

*//使用AOP注意在mvc的配置文件里开启注解,面向的是controller的切面，非service*@Before(**"execution(\* \*..controller..\*.\*(..))"**)*//前置方法***public void** doBeforeInServiceLayer(JoinPoint joinPoint) {  
 ***log***.debug(**"--doBeforeInServiceLayer--"**);  
 **startTime** = System.*currentTimeMillis*();  
}  
  
@After(**"execution(\* \*..controller..\*.\*(..))"**)*//不管是抛出异常或者正常退出都会执行***public void** doAfterInServiceLayer(JoinPoint joinPoint) {  
 ***log***.debug(**"--doAfterInServiceLayer--"**);  
}

*//这个AOP就是将一个请求信息写入DB的过程*@Around(**"execution(\* \*..controller..\*.\*(..))"**)*//环绕增强,相当于执行该方法，result是返回值***public** Object doAround(ProceedingJoinPoint pjp) **throws** Throwable {  
 *// 获取request* RequestAttributes requestAttributes = RequestContextHolder.*getRequestAttributes*();  
 ServletRequestAttributes servletRequestAttributes = (ServletRequestAttributes) requestAttributes;  
 HttpServletRequest request = servletRequestAttributes.getRequest();  
  
 SystemLog systemLog = **new** SystemLog();  
 *// 从注解中获取操作名称、获取响应结果* Object result = pjp.proceed();*//执行后台方法,result是执行结果* Signature signature = pjp.getSignature();*//获取被增强的方法相关信息* MethodSignature methodSignature = (MethodSignature) signature;  
 Method method = methodSignature.getMethod();  
 **if** (method.isAnnotationPresent(ApiOperation.**class**)) {*//通过注解获取value的值* ApiOperation log = method.getAnnotation(ApiOperation.**class**);  
 systemLog.setDescription(log.value());  
 }

***log***.debug(**"doAround>>>result={},耗时：{} ms"**, result, **endTime** - **startTime**);*//log动态写法*

*//任意数组类型转换***public** <V,E> List<V> convertList(List<E> list, Class<V> clazz){  
 List<V> voList=**new** ArrayList<>();  
 **if**(ToolUtil.*isEmpty*(list)){  
 **return** voList;  
 }  
 **for**(E e:list){  
 V v = convertModel(e,clazz);  
 voList.add(v);  
 }  
 **return** voList;  
}

*//抽象方法，为了在不同子类里实现自定义的 内容***protected abstract** <V,E> V convertAttribute(E e,V v);

Assert.*assertNotNull*(map);*//断言null则抛出异常 ，顺我者昌逆我者亡 原则*

**for** (Iterator iterator = sourceList.iterator(); iterator.hasNext();){*//迭代器* Object sourceObject = iterator.next();

**public static** <T> List<T> union(**final** Collection<T> a, **final** Collection<T> b) {

无论参数是基本数据类型，还是引用数据类型，只要加了final，不好意思，该参数不可以再赋值

*/\*\*  
 \* json数据转化为对象(TypeReference)  
 \* List<String> list = JsonUtil.parseJson(jsonValue,new TypeReference<List<String>>(){}); 调用  
 \*/*@SuppressWarnings(**"unchecked"**)*//这个放在方法上就不会报黄色错误了***public static** <T> T parseJson(String jsonValue, TypeReference<T> valueTypeRef) **throws** IOException {  
 Preconditions.*checkArgument*(StringUtils.*isNotEmpty*(jsonValue), **"字符串不能为空!"**);  
 **return** (T) *defaultMapper*.readValue(jsonValue, valueTypeRef);  
}

@Component *//相当于spring配置文件中的bean***public class** NotDisplaySqlAspect {  
 **public static final** String ***DISPLAY\_SQL*** = **"DISPLAY\_SQL"**;  
  
 @Pointcut(**"@annotation(com.mtx.common.annotation.NotDisplaySql)"**)*//表示对什么样的方法使用切面* **private void** myPointCut() {  
 }

<**input onfocus="this**.blur();**" 禁止输入框聚焦**

**function** *removeField*(event) {*//通过event对象操作页面元素* **var** obj = *eventParseObject*(event); *//获取点击事件的源对象*  
 **if**(event.**target**.**nodeName** == **'SPAN'**){*//判断标签类型* obj.parent().parent().parent().remove();  
}**else** {  
 obj.parent().parent().remove();  
}  
}

**onClick="***removeField*(***event***)**"** >

**public abstract class** BaseCacheFactory **implements** ICache{  
 @Override  
 **public** <T> T get(String cacheName, Object key, ILoader iLoader) {  
 Object data = get(cacheName, key);*//抽象类里使用了 抽象方法，实际上方法会在子类的 实现方法中执行* **if** (data == **null**) {  
 data = iLoader.load();*//调用了其它接口的 抽象方法，可以在子类中重写实现* put(cacheName, key, data);  
 }  
 **return** (T) data;  
 }

CacheKit.*get*(**"1"**, **"2"**, **new** ILoader() {*//子类中重写* @Override  
 **public** Object load() {  
 **return null**;  
 }  
});

**private static** ICache *defaultCacheFactory* = **new** EhcacheFactory();*//多态 ICache是个接口*

*defaultCacheFactory*.put(cacheName, key, value); //调用接口的方法

**private static** Pattern *XSS\_1* = Pattern.*compile*(**"<script>(.\*?)</script>"**, Pattern.***CASE\_INSENSITIVE***);

String rlt = WafKit.*XSS\_1*.matcher(rlt).replaceAll(**""**); //将正则的内容置空

*//异步操作记录日志的线程池 指定延时后执行任务,把定时器放入其中***private** ScheduledThreadPoolExecutor **executor** = **new** ScheduledThreadPoolExecutor(10);

**executor**.schedule(task, **OPERATE\_DELAY\_TIME**, TimeUnit.***MILLISECONDS***);

*//执行定时器***public** TimerTask exceptionLog(**final** Integer userId, **final** Exception exception) {  
 **return new** TimerTask() {  
 @Override  
 **public void** run() {  
 SystemError systemError = SystemLogFactory.*createErrorLog*(userId,exception);  
 **try** {  
 **systemErrorService**.insertSelective(systemError);  
 } **catch** (Exception e) {  
 ***log***.error(**"创建异常日志异常!"**, e);  
 }  
 }  
 };  
}

@PostConstruct  
**private void** init() {*//构造方法执行后，用来初始化的* 也可以理解为在spring容器初始化的时候执行该方法**this**.**configMap** = **new** HashMap<String, String>();

file.transferTo(destFile);*// 复制dest文件*