**总结文档**

主要技术选型：

1. **自创持久层小框架wzxSQL——减少持久层”切面”编写**
2. JDK动态代理——AOP面向切面编程，代码增强，减少交叉事务
3. 静态工厂模式——责任分离
4. ThreadLocal事务管理——提高安全性
5. 三层架构——职责分化，代码功能模块化，解耦
6. 数据库MySQL——持久化数据，满足项目动态数据变化
7. 面向接口+抽象编程——DIP依赖倒置

***技术实现思路：***

1. 小框架的创建围绕两个点，反射+字符串的拼凑。通过传入的封装类字节码文件可以创建该封装类对象，以及通过获取的可变参数可以为？赋值，并且通过字符集对象可以获取到项名，把项名通过字符串的拼凑可以将其化为pojo类对应属性的set方法名，然后有了方法名和字节码文件，很轻松就可以通过反射来赋值，有细节在于对于单项项名和双项列名的拼凑，以及对于DQL语句和DML语句的区别对待，详情可直接看源码
2. 围绕（增强代码+代理方法+增强代码）的架构，进行ThreadLocal的事务添加，这样子在调用该代理方法时会自动触发跳转到JDK动态代理的代理处理器中去进行方法的进行，并且执行的结果是经过增强后的代码，有个细节在于对于方法的四要素（执行对象、返回值、参数、具体方法）中有三个是在处理器的invoke方法中可以在JDK底层获取到，但是对于执行对象就需要靠外界传参，所以就得搞一个私有目标对象属性+带参初始化其属性值。其实采取这种JDK动态代理的一个典型就是Spring的AOP编程，功能就三儿：**增强、保护、连接**
3. 静态工厂模式的话在我看来主要是为了使对象的生产和消费分离开来，减少程序的耦合度，实现职责分化，原理在于将所有的具体产品角色抽离一个抽象产品角色作为父类，然后在工厂类中设置静态的方法来通过传入的参数来对应获取需要的具体产品对象，细节在于该方法的返回值是抽象父类，这里利用到的是抽象类多态的知识点。但是静态工厂模式的缺点也很明显，那就是该工厂类集中了所有产品的创造逻辑，形成一个无所不知的全能类（上帝类）。显然工厂类非常关键，不能出问题，一旦出问题，整个系统瘫痪，安全性不高，同时，不符合OCP开闭原则，在进行系统扩展时，需要修改工厂类。我们还可以进行一下优化，采用动态工厂模式，但是一个工厂对应一个具体产品就很麻烦，就偷下懒不做了哈哈哈哈
4. 对于ThreadLocal的话，其实它的底层就是采用了Map集合的方式，构建三大主要方法，get/set/remove，它的实现原理在于第一次从ThreadLocal中获取连接对象（获取原理是基于当前线程的线程名为Key的Map搜索），那么毫无疑问，它是一个空的对象，那么在调用后要添加一个是否为空的判断，为空就自己构建一个connection对象，然后通过**set**方法来将该对象与当前线程进行绑定，那么在这个线程执行的过程中通过**get**方法，依赖于同一线程的基础，就能得到唯一的一个coonection对象了，一个对象贯彻在一条线程，在初始开启事务，最后提交事务，就实现了该业务的事务管理，别忘了关闭资源时要将Map中的那条数据用**remove**移除，不然再次访问ThreadLocal时拿到的可能是上一次的线程对象，但是你上一次的连接对象已经归还/关闭了，就会导致报错，其原因在于**线程池的使用**
5. 三层架构的话其实不算一种技术，而是一种设计理念吧，基于**MVC开发模式**，为了程序设计的合理性和业务分配，围绕表现层+业务逻辑层+持久层，各层分工明确，表现层用来处理数据获取、操作和响应，业务逻辑层就是类似于拼积木，通过将持久层那些没有逻辑性的东西进行逻辑拼凑形成复合功能体，提高业务能力，增加了四大方法（CRUD）的复用性，最后持久层就是数据库的访客了，里面没啥逻辑可言，所以说对于这些没啥逻辑的”切面”类，**mybatis**和**Spring**中时可以动态生成的
6. 常规的CRUD，可以添加**连接池**来进行优化，提高效率
7. DIP依赖倒置的话，目的在与降低程序的耦合度，提高扩展力，大白话就是你死别害大家一起死，其实它的实现的话，目前我感觉还是得搭配着Spring的DI（依赖注入）功能才比较明显，当然，**面向接口**和**面向抽象**，**不面向具体**还是它的底层原理

**设计思路：**

1. 表现层——>业务层——>持久层，以此三大主要步骤进行大模块编写
2. 围绕管理员和普通用户两个情况设置两个表现层，代码功能分化
3. 对于每一层的方法细分为一个个方法块，通过键盘输入的选项进行选择性调用
4. 对于对象的创造问题围绕工厂模式来进行对象获取，减少程序耦合度，实现职责分离
5. 对于事务编写等”交叉事务”，利用JDK动态代理+ThreadLocal来进行代码增强，一步到位为业务层每一个方法增添事务管理，提高安全性
6. 围绕原生持久层代码的”样式单一，代码繁琐，复用性差”问题，编写框架来进行统一解决
7. 围绕业务层操作的每一个功能的进行多个逻辑可能代码实现，增强代码抗压性和安全性
8. 先进行逻辑安排，通过注解来写出每一步操作的逻辑步骤，然后再围绕注解进行代码编写
9. 代码编写阶段，站以客户端角度来思考项目的亲民性和逻辑性，完善项目的人性化功能
10. 对于复用性高的代码进行块处理，直接调用实现

心得体会：

1. 面对项目，要学会围绕功能来将每一个功能的实现化作一个个代码方法块，类似于拼积木。
2. 带有全局观，什么方法可以复用，什么功能要带来什么逻辑改变，一个方法块的实现要确保不会干扰到其他代码块的正常运行，都要考虑，如果说持久层是一个个小零件，那么业务层就是拼凑小零件为大零件的过程，表现层就是将大零件窜连的部分
3. 对于复用性高的代码可以将其独立化为一个方法来进行调用实现
4. 对于缺乏逻辑性，样式单调的代码，要考虑将其抽离出核心业务层，采用代理或者框架来简化编写，提高代码可读性和逻辑清晰
5. 先安排好逻辑过程，再写，不然delete的想哭
6. 每一个代码块的完结都得以Junit测试成功为准，不然到最后一堆bug哭死

**问题回答：**

1. 对考核作业的评价：

答：个人觉得考核的难度不算太高，但是对于项目的一些功能的逻辑处理和考核题目的思维理解需要一定的实践才能比较轻松的把握。我个人的预计完成耗时在2天左右，但是为了一些优化实现的想法，所以所花费的时间就多了一点点。我感觉这考核的内容更偏向于实践化和逻辑化，如果说我们学习的过程是了解汉字，那么这一次的考核就是让我们将汉字进行逻辑化构造作文。感觉帮助挺明显的，让我们对知识的掌握更为透彻，且对于体会代码的逻辑性有很大的帮助

1. 为什么想加入爪哇部落：

答：想要靠近更加优秀的人，了解好自己的定位也提高自己的学习动力，同时在一个优秀的学习氛围中促进自己的成长。

1. 对爪哇部落的印象：

答：很强，一开学就听好多老师推荐，就知道是个很nb的学习氛围很浓厚的组织，想加入~~

1. 想对部落说的话：

答：鑫哥哥来啦~