项目三总结：

思想转变

从实现功能到设计功能实现的转变

实现同一个需求，有多种不同的解决方式。完成同一个功能模块，也有不同的技术可以做到。在多种技术之间，选择用哪些技术完成一个同能模块，才是目前我需要注重的事情。

譬如，骡窝窝项目。其实不用分布式，完全可以完成可以的需求。这时候，就注意为什么我们现在学习用分布式的方式来实现。分布式这个概念对等的技术是什么。（在知识点储备中说）

同样，在具体的表的设计，方法的设计等。也是有很多种实现方式的。那项目三中表的设计理由又是什么，为什么这么设计。都是需要注意到的东西。

这种，从关注具体功能的实现流程 到 关注实现功能的技术的选择应用，是项目三学习，对思想上的改变。

初级程序员到核心程序员思想上的转变

老师提出过一个问题，初级程序员和核心程序员的区别是什么。学习项目三以后，我总结有其下几个方面：

技术面的不同：

核心程序员的技术面较广。譬如，对非关系型数据库的应用，对一些基础公共包的应用，对一些常用框架的熟悉程度。

开发效率的区别：

核心程序员，因为其对技术的接触面，掌握程度比较好。相对应，对同一需求，完成的效率会大大提高。也就是说，开发程序免不了要重复造车轮，核心程序员会少造几个，多用别人造好的。

代码的强壮性：

程序的设计，其落地就是对方法的设计，对表结构设计，对容器的设计上。核心程序员的方法设计上，多用适用性比较强，譬如多用泛型，合理的对数据的包装。这些都使方法更加的适用。

关注细节，到对技术的总体把控

在之前的项目中，大都在意的是对代码细节，对业务流程的把握。经过项目三的学习与总结，感觉到：如果不明确需求是什么，就不要着手开发代码；如果不明确一个技术出现，需要解决什么问题，以及之前解决方案是什么，与新技术对等的技术有哪些，新技术的优势在什么地方，那么，就不要学习这个新技术了。我们项目三的核心，在于对三个分关系型数据库的学习把握。这些技术对等的业务功能，是关系型数据库，即MySQL，Oracle等。

知识点储备

知识点是所有技术功能实现的落地。也是衡量核心技术员和初级技术员的最明显的区别。同时，也是我们是否能顺利就业，能尽快融入工作的重要内功。

在项目三的开发中，我总结学到的知识点如下：

1. **系统架构的演变：**

架构是有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。

而系统架构即：是对已确定的需求的技术实现构架、作好规划，运用成套、完整的工具，在规划的步骤下去完成任务。

说人话，就是：我们之前把我们项目文件放在tomcat猫里面运行，项目三则是运用Springboot，把tomcat猫放到了项目里面。这就是两种不同的系统架构。而之所以产生这些形形色色的系统架构，原因是在于互联发展中需求的不断变化。

- web1.0时代：

在web兴起的伊始，我们的需求仅仅是能把数据库，服务器里面的信息展示到网页上即可。解决这个需求的核心技术，就是tomcat。tomcat能够让只要符合他规定的结构的项目，通过打包成war包，部署到tomcat指定的文件夹中，开启tomcat，就能完成部署。然后外部就可以访问项目里面的资源了。

- 前后端分离时代：

web1.0时代，简单的说，就是一个猫，一个项目。在互联网发展起来以后，随着业务越来越复杂，遇到了瓶颈。即：1. service业务逻辑层越来越多，相互调用变得复杂，后期的开发成本增加。2. 前端搭建本地环境也复杂起来，统一前后端的环境难度增加。JSP代码的可维护性变差。 不是web1.0的问题，是互联网发展带来的成长的烦恼。

解决这个问题的最佳实践，即：springMVC。SpringMVC一类的框架，使得前后端相互分离，这也引出了互联网开发的一个趋势：追求低耦合。前后端分离，使得前端专注于关心页面展现的问题，后端专注于关注数据处理于业务调用的问题。这时候，后台的开发，也真的变成了接口开发，即，前台发过来一个请求，我们响应正确的数据回去即可。相伴随的数据交互格式，最佳实践，及时JSON字符串。

前后端分离，在课程上，反应在第二个项目。这时候带来了一系列的问题，跨域吖等等。

- 集群的出现：

互联网新的需求是不断涌入的。随着互联网应用的普及，一个网站，同时访问的人越来越多。也就是：即使前后端分离，前端可以独立部署在一个猫上了。对于互联网上，比较火的网站，其某时刻的并发量越来越大。那么，对于前端来说，一个猫不够用了。解决这个问题的现有最佳实践，即是集群的引入。

在项目中的应用，我们可以看到，在website，mgrsite这些模块，是由template的。其他的没有，也就是这两个业务模块就是前端的业务模块。并不是说，我们不是前端开发人员，就不开发与前端相关的东西了。后端的开发，就是接口的开发，前端开发，不仅是页面的开发设计，还有业务请求中的那些controller。

sebsite，如果并发太多，就可以用集群的方式解决问题。

- 分布式 + 集群

集群，其实已经解决了绝大多数的问题。而分布式，只是在大流量的情况下解决成本问题。因为实践发现，对于前端的访问，不同接口接受到的流量是严重不均等的。譬如，双十一的电商项目，在当天处理订单的接口，其流量是远远高出与评论的流量。

如果不解决流量分配处理的问题，我们势必需要把所有的模块处理流量的能力都提升到最高的哪一个。这个成本不言而喻。

解决不同接口流量，并发不均等的最佳实践就是分布式。

分布式的优势，在于，大大降低了后端开发的耦合性；同时也就达到了对于并发比较大的功能模块，运用集群的技术，给其更大的并发处理功能。

分布式，是相对于业务层面来说的，集群是解决高并发的问题。

架构的演变发展没有停止，正如技术也在不断进步。譬如，现在新出现的开发模式：Node带来的全栈式开发模式...

2. 项目拆分

这个其实是项目经理需要做的，这里我也不妨说说我的观点：

1. 项目拆分原则：根据访问的并发拆分。
2. 学习项目拆分规则：根据知识点的拆分。譬如，同关系型数据库交互的模块，就是老师的Article模块；同redis交互的模块，老师放在的cache模块中；同elasticsearch交互的，老师都放在的search服务中；同mongoDB交互的，老师放在了comment包内。

疑问:这里老师拆分的依据，到底是模拟真实项目，还是方便学习。因为我发现，不同功能模块之间，其实相互引用了，譬如，article服务就被其他的项目反复引用。真实环境下，我们是专门建立一个模块，同数据库交互，然后对这个功能模块集群，还是，在不同模块，分别配置控制sql数据库的环境。

3. 注册功能设计：

对功能的设计，最初都应该从数据库的设计开始。注册是对用户信息的收集，这里选择用关系型数据库。其实用户的信息也可以使用mongoDB来实现。这里使用MySQL是因为MySQL技术普及率比较高，所有的程序员都会用。而且用户的注册，登陆这些业务流量比较少，不会存在太高的并发。关系型数据库能应付过来。同时，关系型数据库的事务功能，也可以给用户登陆做一定的安全性保证。

MySQL中对数据的存储，是通过其中的表来实现的。存储数据格式的设计，即对表中列的设计。原则上需要包含所有用户信息。这里我们可以用一个主表，一个附表来存储。主表存储用户的主要数据，后期如果需要扩充，用一个附表关系到主表即可。

这里表的列，是用户登陆需要在页面显示的必须数据。

其中用户登陆必须：phone/password

用户在页面中显示常用数据：nickname/headImgUrl/info(用户个人签名)/gender/city/birthday/level/email

特殊列：state-->原则上每一张表都需要由state列。可以用来自由定义对数据筛选的条件。譬如在用户表中，可以定义哪些用户被拉黑名单，或者向哔哩哔哩中那种，进入小黑屋的用户。

注册信息的筛选：

这里我们使用的是用手机号注册。在注册中存在机器人刷数据的现象。避免无效的数据涌入数据库，通常我们有以下的技术来筛选。

- 前端页面筛选：

  在前端页面用js可以动态的对数据进行筛选。在前端js的验证并不安全，只是后台验证的前期筛选。是为了减轻后台压力存在的。同时前台验证可以更友好的同用户交互，更快的提醒用户哪些操作不正确。

- 验证码防刷：

  防止机器人刷数据的第一道防线，由后台生成一个随机UUID，放在session中，通过比对台前的输入数据和后台存储的验证码是否一致来区分。

- 后台数据校验：

  后台数据的校验，才是真的校验。同时注意，应当使用异步请求的方式请求和响应。因为直接跳转会导致前台数据丢失，交互很不友好。

- 短信验证码，验证邮件的确认

最安全的校验方式。这里发送验证码，或者发送验证邮件，简单的方式是调用第三方的接口。

4. **调用第三方接口：**

首选明确，调用第三方接口是为什么，怎么做。

接口即是某一个功能，比如发短信。我们自己可以开发，但是开发周期长，效率低。与其这样浪费人力资源在接口开发，不如用第三方提供的公共接口。

怎么做：接口，关注的应当是怎么发送请求，怎么接受和处理响应数据。之前我们发送请求都是通过浏览器或者是通过postman这种第三方工具发送的。现在我们需要使用java代码，让我们自己的程序自己发送和接受请求。

java代码发送请求核心类：

URL，HttpUrlConnection。

我们只需要new 一个Url对象，通过openConnection方法得到连接对象。如果是HttpUrl，就强转。然后就可以通过HttpUrlConnection对象中的不同方法设置请求对象，请求体等内容。

注意：这里需要setDoOutput（true）来这是带请求参数。

最后使用：

connection.getOutputStream.write((String)msg.getByte(UTF-8)),

connection.connect();即可以通过流的方式发送请求了

同时一个对话可以接受响应。

```
String resp = StreamUtils.copyToString(connection.getInputStream(), Charset.forName("UTF-
8"));
Object response = JSON.parseObject(resp, Object.class)
```

通过对json对象指定内容的读取，就可以判断响应状态了。

5. **统一异常处理：**

需求：异常具有两个功能，1. 报错，即是框架定义的异常，希望提醒我们开发人员，系统的某一部分出错了。2. 开发人员自定义异常，用户统一传输数据。

比如。我定义了一个可见异常，VisibleException，就可以在用户登陆验证的时候，在某一个部分不符合要求，就直接中断后面的业务逻辑，直接通过异常把需要的信息抛出去。

这时候，就需要区分异常，做统一的异常处理。因为框架给的异常是给开发人员看的，我们自己定义的，才是需要显示在前台页面给用户看的。

解决方式：springboot的ControllerAdvice标签。

这是通过AOP实现的对当前项目的所有Controller的业务增强，是一个环绕增强。相当于把整个Controller给try起来了。标签内的Class即对某一种，或者多种Exception进行catch。

catch，即这个异常再次终止了，怎么才能把异常信息传输到页面上？使用HttpServletResponse，直接把前台页面需要的异常信息写出去即可。

注意：@ControllerAdvice是springboot系统识别的，作用域在当前项目的所有Controller中。

注意2：在不同模块之间的异常交流中，通过Dubbo接受的异常，Dubbo会对异常进行包装，此时统一异常处理就会识别不了是我们自定义的了。此时需要在生产者的接口中声明出抛出异常的类型，Dubbo才不会包装它。

方法学上的提升，如何学习

这个是之前学习redis中，得到的感悟。当时开发的效率极慢，而且在代码中绕不回来，同时完成需求的任务量比较大，一晚上完成不了。当时我抽了根烟，然后反省为什么会在代码中出不来了。发现自己太在意细节的处理而把目标给丢失了。

即：

必须明确，redis是数据库，同关系型数据库MySQL中一样，我需要关系只有两个地方，怎么把数据放进去，怎么把放进去的数据取出来处理。用这两个标准来做需求。后面对redis中key的设计，和value的设计只是设计存储方式而已。和redis的使用没有关系。

应该把业务的繁琐 和 技术的使用分开。需要明确，技术的使用不应该成为制约我们的部分，毕竟，只是使用。做到需求即可。

所以，当时突然想到，学习技术的时候，应该抓住主要点。

对于数据库，我们关心的是放进去，取出来。更详细的是CRUD，除此没有其他。

对于调用第三方接口，我们关心的应当是如何发送请求，如何获得和处理响应。

对于同一异常处理，我们应该关心的是，这个技术作用域，如何收集所有异常。然后记住主要的注解和核心的语法即可。