15讲一起练习：手把手带你分解任务

前面在讨论 TDD 的时候，我们说任务分解是 TDD 的关键。但这种认识依然是一种感性上的认识。今天，我们就来用一个更加具体的例子，让你看看任务分解到底可以做到什么程度。

这个例子就是最简单的用户登录。需求很简单，用户通过用户名密码登录。

我相信，实现这个功能对大家来说并不困难，估计在我给出这个题目的时候，很多人脑子里已经开始写代码了。今天主要就是为了带着大家体验一下任务分解的过程，看看怎样将一个待实现的需求一步步拆细，变成一个个具体可执行的任务。

要完成这个需求，最基本的任务是用户通过输入用户名和密码登录。

> 用户通过输入用户名和密码登录。

用户名和密码登录这个任务很简单，但我们在第一部分讲过沙盘推演，只要推演一下便不难发现，这不是一个完整的需求。

用户名和密码是哪来的呢？它们可能是用户设置的，也可能是由系统管理员设置的。这里我们就把它们简单设定成由用户设定。另外，有用户登录，一般情况下，还会有一个退出的功能。好了，这才是一个简单而完整的需求。我们就不做进一步的需求扩展。

所以，我们要完成的需求列表是下面这样的。

用户注册用户名密码。

注册用户通过输入用户名和密码登录。

登录用户退出。

假设我们就是拿到这个需求列表的程序员，要进行开发。我们先要分析一下要做的事情有哪些，也就是任务分解。到这里，你可以先暂停一会，尝试自己分解任务，之后，再来对比我后面给出分解的结果，看看差异有多少。

好，我们继续。

我们先来决定一下技术方案，就用最简单的方式实现，在数据库里建一张表保存用户信息。一旦牵扯到数据库表，就会涉及到数据库迁移，所以，有了下面的任务。

设计用户信息表。

编写用户信息表的数据库迁移。

这时，需要确定这两个任务自己是否知道怎么做。设计表，一般熟悉 SQL 的人都知道怎么做。数据库迁移，可能要牵扯到技术选型，不同的数据库迁移工具，写法上略有差别，我们就把还不完全明确的内容加到任务清单里。

设计用户信息表。

数据库迁移工具选型。

编写用户信息表的数据库迁移。

数据库的内容准备好了，接下来，就轮到编写代码的准备上了。我们准备用常见的 REST 服务对外提供访问。这里就采用最常规的三层技术架构，所以，一般要编写下面几项内容。

- 领域对象，这里就是用户。
- 数据访问层，在不同的项目里面叫法不一，有人从 J2EE 年代继承下来叫 DAO（数据访问对象，Data Access Obejct），有人跟着 Mybatis 叫 mapper，我现在更倾向于使用领域驱动设计的术语，叫 repository。
- 服务层，提供对外的应用服务，完成业务处理。
- 资源层，提供 API 接口，包括外部请求的合法性检查。

根据这个结构，就可以进一步拆解我们的开发任务了。

编写 User 对象，包含用户名、密码和其他用户基本信息。

在 UserRepository 中，编写 save 方法，保存新创建的 User 对象。

在 UserService 中，编写 register 方法，实现用户注册。

在 UserResource 中，编写 register 方法，调用 UserService 的 register 方法。

在 UserRepository 中，编写 find 方法，根据用户名获取 User 对象。

在 UserService 中，编写 login 方法，实现用户登录。

在 UserResource 中，编写 login 方法，调用 UserService 的 login 方法。

在 UserService 中，编写 logout 方法，实现用户退出。

在 UserResouce 中，编写 logout 方法，调用 UserService 的 logout 方法。

不知道你有没有注意到，我的任务清单上列任务的顺序，是按照一个需求完整实现的过程。

比如，第一部分就是一个完整的用户注册过程，先写 User，然后是 UserRepository 的 save 方法，接着是 UserService 的 register 方法，最后是 UserResource 的 register 方法。等这个需求开发完了，才是 login 和 logout。

很多人可能更习惯一个类一个类的写，我要说，最好按照一个需求、一个需求的过程走，这样，任务是可以随时停下来的。

比如，同样是只有一半的时间，我至少交付了一个完整的注册过程，而按照类写的方法，结果是一个需求都没完成。这只是两种不同的安排任务的顺序，我更支持按照需求的方式。

我们继续任务分解的讨论。任务分解到这里，需要看一下这几个任务有哪个不好实现。register 只是一个在数据库中存储对象的过程，没问题，但 login 和 logout 呢?

考虑到我们在做的是一个 REST 服务，这个服务可能是分布到多台机器上，请求到任何一台都能提供同样的服务，我们需要把登录信息共享出去。

这里我们就采用最常见的解决方案：用 Redis 共享数据。登录成功的话，就需要把用户的 Session 信息放到 Redis 里面，退出的话，就是删除 Session 信息。在我们的任务列表里，并没有出现 Session，所以，需要引入 Session 的概念。任务调整如下。

在构建脚本依赖中添加 Redis 的依赖。

编写 UserSession，包含用户名、登录时间和用户 Token 等。

在 UserService 中，编写 login 方法，将 UserSession 放入 Redis。

在 UserService中，编写 logout 方法，将 UserSession 从 Redis 中删除。

如果采用 Redis，我们还需要决定一下在 Redis 里存储对象的方式，我们可以用原生的Java序列化，但一般在开发中，我们会

选择一个文本化的方式，这样维护起来更容易。这里选择常见的 JSON，所以，任务就又增加了两项。

在构建脚本依赖中添加 JSON 的依赖。

在存储到 Redis 时，将对象序列化成 JSON，取出时将 JSON 反序列化成对象。

至此，最基本的登录退出功能已经实现了，但我们需要问一个问题，这就够了吗？之所以要登录，通常是要限定用户访问一些资源，所以，我们还需要一些访问控制的能力。

简单的做法就是加入一个 filter，在请求到达真正的资源代码之前先做一层过滤，在这个 filter 里面，如果待访问的地址是需要登录访问的，我们就看看用户是否已经登录，现在一般的做法是用一个 Token，这个 Token 一般会从 HTTP 头里取出来。但这个 Token 是什么时候放进去的呢？答案显然是登录的时候。所以，我们继续调整任务列表。

在 UserService 中，编写 login 方法，将 UserSession 放入 Redis，返回 UserSession。

在 UserResource中，编写 login 方法，调用 UserService 的 login 方法，并根据返回的 UserSession 设置 HTTP 头。

编写 AccessFilter，根据待访问地址，确定是否进行用户登录信息校验。

至此，我们已经比较完整地实现了一个用户登录功能。当然，要在真实项目中应用，需求还是可以继续扩展的。比如：用户 Session 过期、用户名密码格式校验、密码加密保存以及刷新用户 Token等等。

这里主要还是为了说明任务分解，相信如果需求继续扩展，根据上面的讨论，你是有能力进行后续分解的。

来看一下分解好的任务清单，你也可以拿出来自己的任务清单对比一下，看看差别有多大。

设计用户信息表。

数据库迁移选型。

编写用户信息表的数据库迁移。

编写 User 对象，包含用户名密码。

在 UserRepository 中，编写 save 方法，保存新创建的 User 对象。

在 UserService 中，编写 register 方法，实现用户注册。

在 UserResource 中，编写 register 方法，调用 UserService 的 register 方法。

在 UserRepository 中，编写 find 方法，根据用户名获取 User 对象。

在构建脚本依赖中添加 Redis 的依赖。

编写 UserSession，包含用户名、登录时间和用户 Token 等。

在 UserService 中，编写 login 方法，将 UserSession 放入 Redis，返回 UserSession。

在 UserResource中，编写 login 方法，调用 UserService 的 login 方法，并根据返回的 UserSession 设置 HTTP 头。

在 UserService 中，编写 logout 方法，实现用户退出。

在 UserResouce 中，编写 logout 方法，调用 UserService 的 logout 方法。

在构建脚本依赖中添加 JSON 的依赖。

在存储到 Redis 时，将对象序列化成 JSON，取出时将 JSON 反序列化成对象。

编写 AccessFilter，根据待访问地址，确定是否进行用户登录信息校验。

首先要说明的是，任务分解没有一个绝对的标准答案，分解的结果根据个人技术能力的不同，差异也会很大。

检验每个任务项是否拆分到位，就是看你是否知道它应该怎么做了。不过，即便你技术能力已经很强了，我依然建议你把任务分解到很细，观其大略人人行，细致入微见本事。

也许你会问我，我在写代码的时候，也会这么一项一项地把所有任务都写下来吗？实话说，我不会。因为任务分解我在之前已经训练过无数次，已经习惯怎么一步一步地把事情做完。换句话说，任务清单虽然我没写下来，但已经在我脑子里了。

不过，我会把想到的，但容易忽略的细节写下来，因为任务清单的主要作用是备忘录。一般情况下，主流程我们不会遗漏，但

各种细节常常会遗漏，所以，想到了还是要记下来。

另外，对比我们在分解过程中的顺序，你会看到这个完整任务清单的顺序是调整过的，你可以按照这个列表中的内容一项一项地做，调整最基本的标准是，按照这些任务的依赖关系以及前面提到的“完整地实现一个需求”的原则。

最后，我要特别强调的一点，所有分解出来的任务，都是独立的。也就是说，**每做完一个任务，代码都是可以提交的。**只有这样，我们才可能做到真正意义上的小步提交。

如果今天的内容你只能记住一件事，那请记住：**按照完整实现一个需求的顺序去安排分解出来的任务。**

最后，我想请你分享一下，你的任务清单和我的任务清单有哪些差异呢？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言

desmond
这篇文章就值这套课程的价钱，赚了！
2019-02-01 09:25

作者回复
把文章转发给给朋友同事，赚更多！
2019-02-01 20:33

你的笑犹如初夏的阳光
没有把事情想透，就直接进去开发。发现自己会踩很多坑。而且在开发的过程中会遇到各种各样的麻烦。
2019-02-01 06:45

毅
老师的一条回复我觉得很有启发啊：“不忙的时候你知道该怎么做吗？”首先是意识问题，其次就是执行了。我自己的经历来看，之前在项目型软件公司，粗放式的工作惯了，后来经历了一家产品型公司的洗礼，刚开始格格不入，后来逐渐习惯了，现在想快也快不起来了。我经常跟我团队兄弟们说，想透了再动手，客户公司那些事我来搞定，我只要扎实的可交付的成果就行，拒绝返工！
2019-02-01 21:47

作者回复

忙，是做事不动脑的借口。

2019-02-02 10:23

pyhhou

任务分解走一遍，发现了好多的问题。像是我自己分的话就只会根据功能去划分 API 函数以及一些功能函数，脑子里面总是会有一种一个功能对应一个函数的思维，不知道是不是 JS 写多了；对于老师您的分解，像是功能分层，还有一些额外的考虑，类似 Redis 里面加 token，这些东西在我平时从来都没有接触过，很难想到，想请教一下老师，平时如何高效地学习写业务代码的的最佳实践，以及一些项目中必须考虑的东西，感觉能力经验不够，但是有一个明确的学习目标和方向，总是好的，谢谢老师，大年初一，祝老师在新的一年里心想事成，身体健康，工作生活一切顺利～

2019-02-05 17:40

作者回复

大年初一还在坚持学习和提问，赞一个！

行业中常见的实现方案，市面上大多数都可以搜到，是不是最佳的不一定。看别人的东西，多想想如果自己实现会怎么做，再到网上搜一下，对比一下别人的方案。这些东西就是慢慢积累的。

2019-02-05 22:27

shniu

想请问一下老师面对探索型的需求，调研型的需求如何做任务分解呢。大部分情况下，都会面临要做一些探索型的工作，一些新的架构，新的业务尝试，新的技术尝试，要面对很多自己并没有接触过的知识。举个实际的例子，最近在研究一些健值存储数据库，读了一些paper，一些设计实现，这可能都是要做的一些任务清单中的一些项，但是我想去实现一个kv存储的时候就会感觉有些无从下手，实际落地到代码层面和读paper读博客还是有很大差别的，落地代码更关注细节，关注写的每一行代码，在这方面老师能否给些建议

2019-02-02 18:11

作者回复

答疑见！

2019-02-12 16:45

西西弗与卡夫卡

郑老师的拆分更精细，非常值得学习。补充一点我自己的习惯，可以每做一个feature时就拉一个分支，feature进一步拆成更细的任务。由于多数编辑器都有本地历史，所以不常做git add。任务完成并且测试通过后再一把提交。整个feature完成再合并分支

2019-02-01 20:46

作者回复

前面说了，每一步都很小的话，用不着分支。

2019-02-02 10:23

nicklim

聽了老師的講解，開始反省自己在日常工作中沒有寫工作清單。如果我對技術不夠了解，不能拆解細分工作。再寫清單前需要做其他準備工作嗎?

2019-02-01 19:06

作者回复

不熟悉的技术，我在答疑中单独聊。

2019-02-01 20:32

WL

在开始写代码前事先推演一遍业务实现逻辑，就相当于在大脑中完成一次创造，写代码就是第二次创造。在大脑中推演创造可以整体把握实现逻辑， 形成清单后， 在后面真正写代码的时候就不至于陷入很多个细节中跳不出来或者遗漏，与同事交流的时候也有依据，而且可以始终保持从整个流程出发的宏观视角。老师讲的太棒了！

2019-02-01 15:51

划时代

看了一些同学的评论，有感而发想表达一下自己的观点，并非是口诛笔伐。之前也听过一些如何做好软件开发的大牛讲座；有一部分听众反馈，项目周期很紧催促上线，哪有机会按照标准流程开发，保证高质量文档，保证高质量代码？这确实是客观的事实，是极端情况，然而很多时候仅仅是挂在嘴边的借口用来搪塞的，却忘了工程师本身的职责。我不太相信，一位工程师在一年的工作中，每一个项目，每一天的工作，都如十万火急一般，无法保证项目的质量。一年中有忙的时候，也有不忙的时候

；一年中有急于上线的项目，也有井然有序的项目，往往不忙的时候多过忙的时候吧。在上升期很快的创业公司工作，也大概是这样子的。所以很多时候是考验个人执行力的问题，不断地实践和总结，掌握熟练之后得心应手。

2019-02-01 10:28

作者回复

其实只要回答一个问题，当你不忙时，你知道该怎么做吗?

2019-02-01 20:34

John

请教，UserService 和 UserResource 的职能划分？我做的项目只有 Repository 和 Service 两层，故有此疑问。谢谢！

2019-02-01 09:15

作者回复

这里的Resource相当于一些项目的Controller，因为这里用的是Rest，对应资源的概念，所以，用了Resource。

2019-02-01 10:04

Jxin

任务分解，想清楚再写。这些观点都认可。但公司实际工作，存在时间根本不够的情况。需求下来，确定上线日期，往前推几天提测，其他时间开发。

比较遗憾的事情是包括以终为始在内，大部分套路大家都懂，也认可哪怕时间不足也不能跳过这些步奏。然而真正面对时间不足，而且连续不足，而且势必的线上bug与需求冲突场景日益增多。真的就一边不认可一边就这么做了。对于稳定的系统，需求真的很难砍。而在人员调动频繁的团队中去接手稳定的系统，真的，每个需求的实际工作量double几倍都不是不可能的。

2019-02-01 08:53

作者回复

我在答疑中说了，我们得有目标，然后结合自己的工作实际，找到一条适合自己的改进路径，别想一步到位的完全改善。

2019-02-14 08:27