# 任务拆分

前后端任务拆分是好事吗？

很多团队采用了前后端分离架构，用户故事也分成前端开发任务和后端开发任务。

按照前后端拆分开发任务的动因是什么？

这种拆分任务的方式有何利弊？

如何避免前后端开发工作量不均衡，“忙的忙死闲的闲死”的问题？

### 1）

前后端拆分在技术上是好事儿，每一块可以更关注自己的业务，而不需要混合在一起；在人员上拆分是坏事儿，导致人员相互等待，空耗时间。

我是全栈的坚决拥护者

Uncle Bob的Twitter

产品需要了就去学，没有刻意去学很多东西。

### **2）**

**前后端任务拆分是好事吗？**

在传统的软件开发模式中，难免会希望招聘full stack的全栈工程师，但是显然在技术发展如此迅速的今天，全栈工程师知识体系过于庞杂、不纯粹，反而不能专注的提高，所以前后端分离是职业细分的必然结果。

但前后端任务拆分往往看起来容易，但是怎么拆，怎么合，沟通是最大的问题，由于不同的知识体系和技术栈或经验的不同，往往在需求开发过程中，会耗费大量的精力进行沟通。

**很多团队采用了前后端分离架构，用户故事也分成前端开发任务和后端开发任务。按照前后端拆分开发任务的动因是什么？**

采用前后端分离的形式来拆分任务，可以把某个需求拆分拆分成子任务，进而让不同的开发者能够更好的关注自己应该关心的功能，并针对性的进行功能的开发。

**这种拆分任务的方式有何利弊？**

优点：便于不同的开发者领用与自己相关的功能，并及时的更新状态。

缺点：子任务往往让同一需求中的其他开发者更倾向于只完成自己的子任务，而不关注其他与其相关、但不会直接处理的任务。

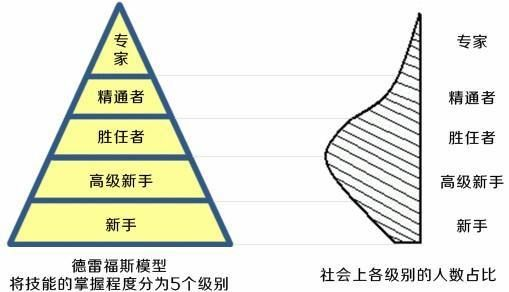
如何避免前后端开发工作量不均衡，“忙的忙死闲的闲死”的问题？

1、做好责任矩阵的分配，明确某个单一需求的直接负责人、执行人、咨询者、知情人。

2、前后端的衔接应该做好接口格式的统一化，该Mock就Mock,该文档就文档，尽量把沟通上可能存在的时间延迟提前消除。

客观来说，由于能力的不一致，忙的忙死、闲得闲死的情况太常见了。做好任务的预分配，做好在制品的工作任务分配。

德雷福斯能力模型



我个人觉得，学一个东西很容易，但精通很难，要付出很多时间的努力，例如加很多班。加班是快速学习技术的一个好办法，但不是沉淀技能的好办法。沉淀技能就只有认真写代码、用心写代码、然后去体会技术点的妙处，并通过代码实践，写博客来输出，然后告诉别人自己的理解，这样才能更好的促进技能的形成。而且技术=》技能，需要刻意的练习

喜欢提全栈的往往是小公司的技术leader居多

### **3）TW**

**前后端任务拆分是好事吗？很多团队采用了前后端分离架构，用户故事也分成前端开发任务和后端开发任务。**

前后端任务拆分有其优点，也有其缺点。技术架构的分离，并不意味着开发任务也要分离，或者开发任务就是全栈不分离。需要更具团队的实际技术能力、学习能力、时间来做balance。

**按照前后端拆分开发任务的动因是什么？**

往往是因为技术原因导致了前后端在技术上不能实现前后端一起实施，开发任务上不做拆分短期会给团队带来开发速度上的挑战。

为了能够迭代内能够产出内容，因此选择了前后端任务拆分。

还有一些技术领域上的限制导致前后端的拆分。例如：游戏。各种终端和后端主程不在前后端开发上拆分，在学习成本和机会把控上都会面临很大的挑战。

**这种拆分任务的方式有何利弊？**

【优点】

 将用户故事分解为前端任务和后端任务，对于非全栈团队能够在地学习成本时就任务完成。

 对前端/后端技能的能力增长会更加快速。

 符合很多开发团队脑子中拆分方式。

 符合某些团队的特点。

【缺点】

增加了沟通成本；

估点时前后端评估标准不同，开发速度不易确定；

前端后端任务拆开了，同时增加了管理成本，需要持续跟踪依赖；

前后端任务拆分除了导致依赖block，还会出现空闲等待的

**如何避免前后端开发工作量不均衡，“忙的忙死闲的闲死”的问题？**

**短期解决：**

 如果出现空闲的问题，可以在迭代内加入一些并非立即要完成的内容。

 如果出现忙死的问题，可以根据当前团队的技术能力，做好tasking和技术点讲解后，让前端开发做后端任务，或让后端开发做前端任务

**长期解决：**

      在开发能力上能够前后端应该可以相互了解，每个人都去尝试既要做前端，也尝试做后端。

      团队能力达到后，可以跟根据实际情况，部分卡不拆分前后端任务。

通用的拆分方式会造成问题。需要根据团队的聚焦的内容来调整出当前团队适合的拆分方式。

### **4）HW**

**前后端任务拆分是好事吗？**

好事，几年前还在用jsp的时候就在公司倡导前后端，早期的UI工程师写好html，后端工程师还要去转化为jsp，当出现问题的时候还需要UI工程师去调整，然后再转化为jsp，效率低下。

很多人觉得沟通会加大成本，团队根据业务的进行组织，将前端工程师纳入到整体，就能解决沟通协作的问题。

**按照前后端拆分开发任务的动因是什么？**

动因，团队间专业化分工协作，提升研发效率。同时，前端可以先行，实现前端页面展示。

**这种拆分任务的方式有何利弊？**

有弊端，前端写得快，页面都写好了。后端接口可能还没有出来，会存在等接口的情况。

不能说完全避免出现忙得忙死，闲的闲死的情况，要求前端工程师注重交互体验，从数据的加载，到给用户的友好提示需要前端工程师去优化调整。

同时，前端工程师并不是只实现静态的页面，需要将内部的业务逻辑实现，等到后端接口实现后进行对接。

另外，要求前端工程师进行组件话的研发，避免同样的功能仍需要花费时间去研发调整。

### **5）**

关于今天的问题，我理解题目说的不是故事而是任务，也就是实现故事所需完成的任务按照前后端拆分是否是好事？谈谈我的理解：

**1. 前后端任务拆分是好事吗？**

从技术角度来说是好事，从管理角度来说是无奈。

**2. 按照前后端拆分开发任务的动因是什么？**

这是由团队面临的现实情况决定的，早期Java程序员都是默认会前端的，近年来出道的程序员则以90后为主，前后端分离成为主流，最主要原因还是在于前后端技术和理论的发展，一个人短时间内很难同时精通这两个领域的技术，导致市场上已经很难找到合格的全栈工程师。而对于团队来说如何更有利于工作展开才是最重要的，一般情况下在没有合格的全栈工程师能同时负责前后端工作时候，按前后端拆分任务几乎成了唯一选择。

**3. 这种拆分任务的方式有何利弊？**

**从技术角度：**

       首先从技术角度前后端分离肯定会带来一系列好处，前后端分离也是现在很多项目的标准做法。

**从分工和管理角度：**

       如果团队研发人员是分前后端的，没有全栈工程师，这时候按前后端拆分任务是必要的，也是相对较好的方法:

     a. 更利于在看板跟踪每个人的工作进度（但是容易忽略故事的整体进度）。

     b. 对全栈工程师依赖降低，降低团队组建和招聘难度。

     c. 分工和责任更清晰，可以让更专业的人做更专业的事儿。

      缺点是

会导致沟通成本增加。造成相互等待，工作衔接不顺畅，如果资源比例不合理还可能形成资源瓶颈，导致在制品积压严重。

**3. 如何避免前后端开发工作量不均衡，“忙的忙死闲的闲死”的问题？**

a. 控制好前后端开发人员比例，鼓励前端学习后端技术，后端了解前端技术，不要求完全精通，但是当工作出现积压时能帮上忙（这个目前我们实践有一定效果，但是还不理想）。

b. Story Owner制度，虽然任务分了前后端，但是针对故事还是有一个主责人，这个人负责推进这个故事开发，包括前后端测试等角色工作的协同，避免前后端工作不同步。

c. 利用好每日站会，规范前后端工作衔接流程，严控在制品，及时发现等待情况并予以干预，经过一段时间磨合后，前后端配合也可以达到很顺畅的程度。