03 | 可扩展架构：如何打造一个善变的柔性系统？

2020-02-26 王庆友

架构实战案例解析

进入课程 >

讲述：王庆友

时长 20:00　大小 16.03M

你好，我是王庆友，今天我和你聊一聊如何打造可扩展的架构。

在实际工作中，业务需求总在不断变化，因此，你经常会面临以下这些问题：

如何快速地上线新业务？老板很可能明天就想看到效果。

对某个功能进行修改，如何不影响到系统其它的功能？

对于新的需求变化，我们一方面要快快搞定，另一方面要稳稳接住。但问题是软件虽然姓“软”，但也不是想变就能变，如果事先没有经过良好的设计，调整起来，往往牵一发动全身，导致系统到处出问题。

那如何设计一个具有良好扩展性的系统，能够快速支持业务变化落地呢？

接下来，我们围绕系统的可扩展，先来了解下什么是系统，什么样的系统才能具备良好的扩展能力。然后通过一个实际的例子，说明如何通过架构手段打造一个可扩展的系统。

系统的构成：模块 + 关系

我们天天和系统打交道，但你有没想过系统到底是什么？在我看来，系统内部是有明确结构的，它可以简化表达为：**系统 = 模块 + 关系。**

在这里，模块是系统的基本组成部分，它泛指子系统、应用、服务或功能模块。关系指模块之间的依赖关系，简单地讲，就是模块之间有调用，我们知道，调用区分发起方和服务方，因此，依赖关系是有方向性的。

这个模型虽然简单，但它给我们提供了一个深入分析系统的工具。接下来，我们就从业务扩展性出发，讨论什么样的模块是容易修改的，什么样的依赖关系是容易调整的。

模块

我们先看模块，模块定义系统都有哪些基本的“玩家”，分别承担什么职责。从业务的角度看，每个模块都代表了某个业务概念，或者说业务领域。

模块内部由数据和业务逻辑组成，其中数据是核心，业务逻辑围绕着数据，对数据做进一步加工，方便外部使用。

从扩展性的角度出发，首先，我们对模块的要求是：**定位明确，概念完整。**

每个模块要有明确的定位，模块有了定位，说明我们已经想清楚了它的核心职责是什么，这样，每个人对它的期望和理解就会一致。在实践中，我们经常会争论一个功能应该放到 A 模块还是 B 模块，表面上看，各有各的道理，谁也说不服谁，但如果对照模块的定位，回到模块设计的初心，我们往往很快就能有答案。

定位比较抽象，在具体划分模块职责的时候，要保证模块业务概念的完整性。数据上，模块需要覆盖对应业务领域的全部数据，比如一个订单模块，它要覆盖所有渠道的订单，包括三方平台的订单、自有商城的订单、线下门店的订单等，这些不同类型订单的数据模型和实际数据，都由订单模块负责。

功能上，模块要包含业务领域的全部功能，比如订单模块包含所有订单相关的功能，包括订单数据的增删改查、订单业务规则校验、订单的状态和生命周期管理等。

其次，模块还要：**自成体系，粒度适中。**

模块的业务逻辑**尽量围绕自身内部数据进行处理**，对外部依赖越小，模块的封装性越好，稳定性也越强，不会随着外部模块的调整而调整。

模块的粒度要保持适中，不能为了追求定位清晰，把粒度划分得很小，导致系统的碎片化。比如系统早期的时候，一般我们把积分功能放到用户模块里面，不单独构建积分模块，如果后续积分的概念越来越突出，承载的业务越来越复杂，到时候可以把积分功能分离出来，单独成模块。

这里，为帮助你更好的理解，我举一个模块划分的反面例子。在实际工作中，很多老系统都有体量很大的模块，我们称之为“肿瘤”，它的特点就是定位模糊，职责泛滥，功能无所不包，这样，模块的可维护性很差，没人敢轻易对它动刀子。

好了，说完了模块，我们再继续看下模块的依赖关系。

依赖关系

依赖关系定义了模块如何协作，一起完成业务流程，依赖关系实质上体现的是模块的组织结构。

如果不对模块的依赖关系做针对性设计的话，依赖关系就是一个多对多的网状结构，一个有 N 个模块的系统，理论上有 N×N 个依赖关系，如果考虑依赖具有方向性，这个数字还要加倍。

所以，要简化模块的依赖关系，我们就要同时简化依赖的方向和减少依赖的数量。

首先，我们希望模块之间的依赖是单向的，尽量避免相互调用，为什么单向更好呢？我们知道业务流程是有顺序的，如果模块依赖关系越直观地体现业务流程的顺序，越能帮助人理解，否则，我们会被双向的依赖箭头绕的晕头转向，很难通过模块之间的依赖关系还原实际业务的处理过程。

接下来，我们看下模块的组织结构。我们知道，网状结构是一种松散的结构，节点之间的依赖关系比较复杂，一般用于表示非正式的关系，比如人群的社交关系；而层次结构是一种更有序的结构，一般用于表示正式的关系，比如公司内部的人员关系。

在模块的组织结构设计上也是如此，我们要尽量把网状结构转化为层次结构，模块结构层次化是简化模块依赖关系的有力手段。

具体做法就是，我们按照模块定位的不同，把模块划分为不同层次，比如划分为上面的应用层和下面的资源层。这样，一个层通过把多个模块组织在一起，就形成了概念上更大粒度的模块。有了层以后，我们理解业务时，因为模块定位相同，往往关注这个更大粒度的层就可以，依赖关系只要指向这个层，而不是层里面的各个模块。这样，从人理解业务的角度，依赖的数量大幅度地减少了。

另外，我们知道，层与层之间的依赖关系都是层与层之间自上而下的依赖，相对于多对多的网状依赖，层次依赖的方向更清晰，特别符合人的理解习惯。

举个具体例子，作为开发，我们都比较了解 MVC 架构，系统模块按照定位，分为表示层、应用层、聚合服务层、基础服务层。

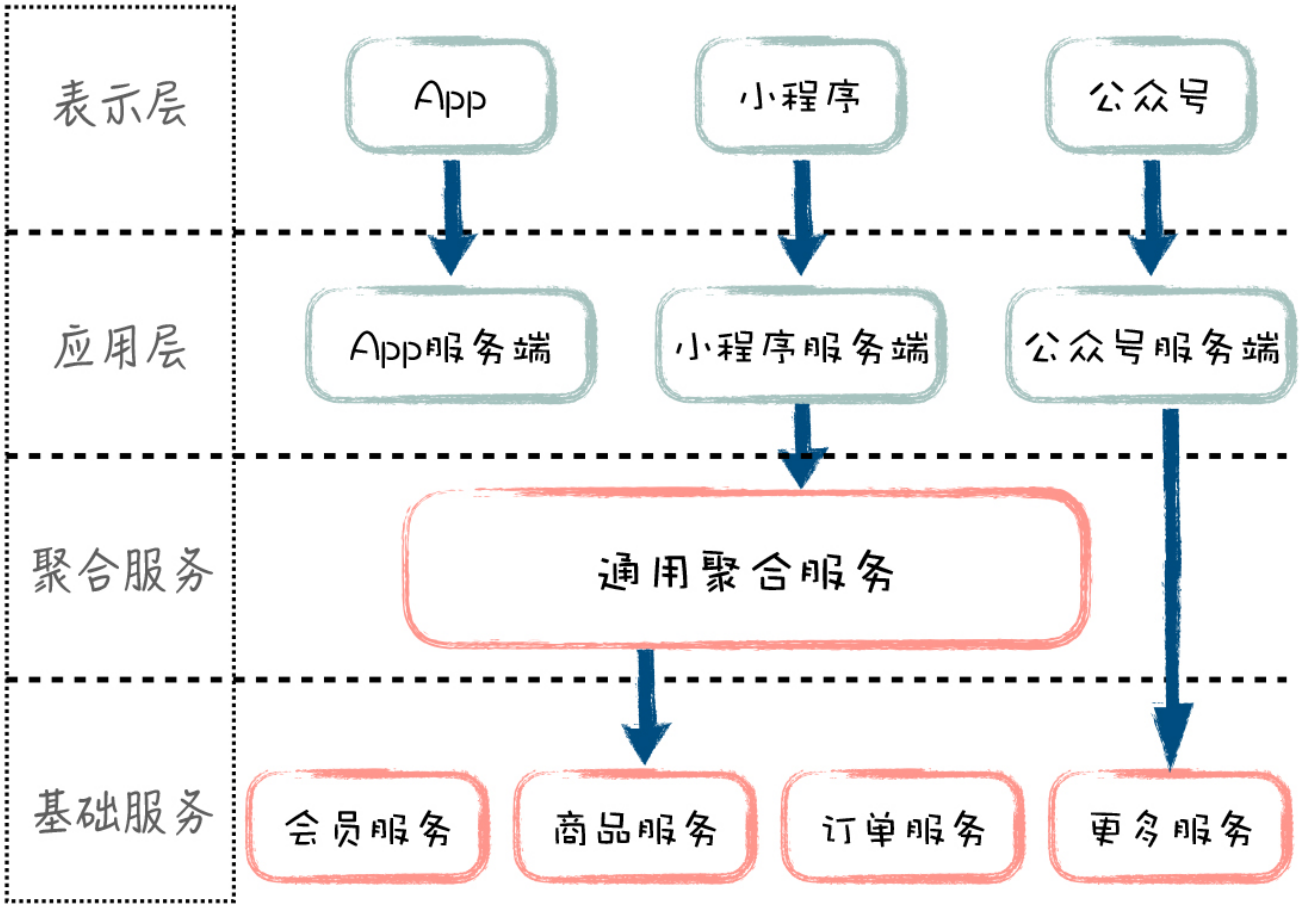
表示层，对应前端的模块，如 App、小程序、公众号等，属于 View 层。

应用层，对应和前端表示层直接关联的服务端，属于 Control 层。

聚合服务层，如果系统业务比较复杂，经常需要单独的聚合服务层负责业务流程的编排组合，这个属于 Model 层的加强。

基础服务层，代表最基础的业务模块管理，如订单、商品、用户等，属于实际的 Model 层。

我在这里贴了一张 MVC 分层结构图，你可以看到，模块总体上是非常清晰的层次结构。



现在，我们清楚了一个可扩展系统对模块和依赖关系的要求，接下来，我们再回到系统扩展性目标，做个深入总结。

扩展性的本质

在文章开头，我们说因为业务总在变化，所以需要架构设计给系统提供良好的扩展性。

这只是表象，深层的原因是，一个新的需求进来，系统不只是为它增加一个新功能这么简单，系统的调整会引起一系列的连锁反应，从而大面积地影响系统的现有功能。架构设计时，如果模块划分的不好，一个 N 个模块的系统，它的复杂度就是 N×N（这个在上一讲介绍的支付宝一代架构中，体现得很明显）。如果再加一个新的模块，复杂度就变成

(N+1)×(N+1)，系统的复杂度随着功能的数量指数级地上升，这样一来，当系统的规模到一定程度，复杂度就会失控，导致系统彻底无序。

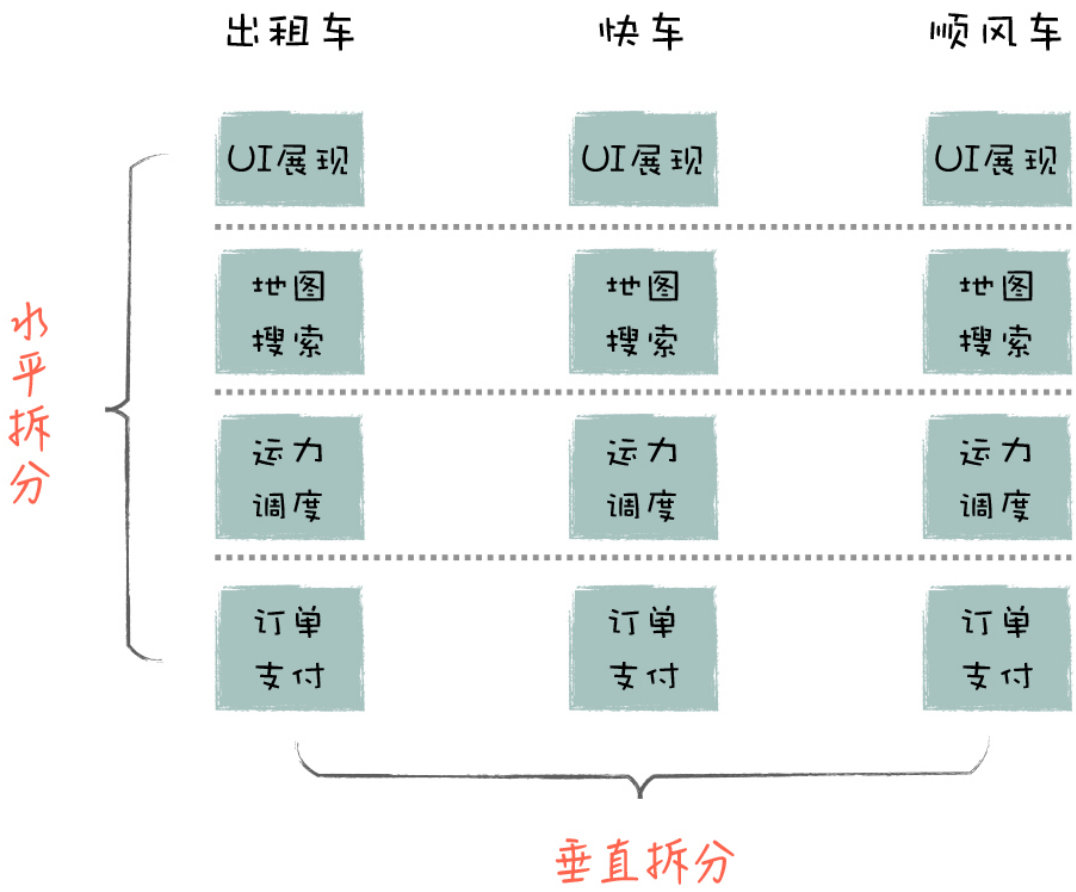
所以，要支持系统的扩展，架构设计上必须能够控制系统的复杂度，面对新需求，要让系统复杂度做加法而不是乘法，从而保证系统的调整是局部化和最小化的，所以，业务架构扩展性的本质是：**通过构建合理的模块体系，有效地控制系统复杂度，最小化业务变化引起的系统调整。**

那如何打造一个合理的模块体系呢？具体的架构手段就是按照业务对系统进行拆分和整合：**通过拆分，实现模块划分；通过整合，优化模块依赖关系。**

接下来，我们以一个在线出行公司为例，它有出租车、快车和顺风车 3 条业务线，来具体看下如何为它打造合理的模块体系。

打造可扩展的模块体系：模块拆分

我们先对系统进行模块化拆分，拆分有两种方式：**水平拆分和垂直拆分。**



水平方向拆分

水平拆分是指从上到下把系统分为多层，按照系统处理的先后顺序，把业务拆分为几个步骤。

比如，整个叫车过程，我们可以分为 UI 展现、地图搜索、运力调度和订单支付等几个环节，这是根据系统的处理过程进行划分的。

这样一来，我们就把一个复杂流程，分解为几个相对独立的环节，分别进行处理，这么做带来了很多好处。

首先，UI 展现部分独立成为一个模块，实现了前后端的分离。我们知道，前端的用户体验和界面样式会经常变化，而后端的数据和业务逻辑相对稳定，通过水平拆分，我们实现了稳定部分和不稳定部分的分开，避免相互影响。

这里的后端包含三个模块，其中地图搜索负责路径规划，运力调度负责人车匹配，订单支付负责交易管理。

可以看到，通过水平拆分，可以使每一块职责都比较明确，功能内聚，每个模块管理自己内部的复杂性。同时，模块之间相互松耦合，一个模块的修改不影响另一个模块，比如地图搜索模块中改变了优先路径的推荐，不会影响运力调度模块中的人车匹配算法。

水平分层可以很好地满足现有业务做深度扩展，当业务有变化时，系统在特定层做调整，对其他层影响有限，这样把变化局限在一个小范围。

垂直方向拆分

垂直拆分指的是按照不同的业务线拆分，比如，将整个出行业务分为出租车业务、快车业务和顺风车业务，按照不同的业务场景，自上而下进行竖切，让每个业务都自成体系，形成自己的业务闭环。

通过垂直拆分，一个复杂的出行场景就拆分为几个具体的场景，我们可以根据各个业务线的特点去设计系统，从而降低了整个系统的复杂性。

垂直拆分可以很好地满足业务广度上的扩展，比如说增加一条新的业务线，可以按照这个思路落地系统。

一般做业务架构时，我们先考虑垂直拆分，从大方向上，把不同业务给区分清楚，然后再针对具体业务，按照业务处理流程进行水平拆分。

如果同时进行垂直拆分和水平拆分，一个大系统被拆分为了一个二维的模块矩阵，每个模块既属于某个业务线，也属于业务流程的某个环节。这样一来，每个模块的职责都很清晰，当业务变化了，我们可以清楚地知道，这个变化涉及哪些模块，然后，对这些模块进行相应的调整就可以。

为了帮你更好地理解这两种拆分方式的好处，我这里举个搭积木的例子。经过拆分，每个业务模块都成为一个积木，然后，我们以搭积木的方式来构造系统。当业务发生变化，我们就调整对应的积木，如果系统拆分得合理，拆分后的模块就具有良好的封装性，也就意味着我们主要是调整积木的内部，而它的外观基本不变。这样一来，相邻的积木不会受到影响，系统整体也不需要大的调整。结果是，系统的变化是局部和可控的，保证了灵活的应对变化能力。

打造可扩展的模块体系：模块整合

系统拆完后，接下来就是模块整合的工作，整合也有两种好的手段：通用化和平台化。

通用化整合

通用化指的是通过抽象设计，让一个模块具备通用的能力，能够替代多个类似功能的模块。

回到刚才的出行平台，我们发现 3 条业务线都有地图搜索、运力调度、订单支付这些模块，不同的业务线之间，这些同名的模块逻辑高度类似，只是细节方面有差别。

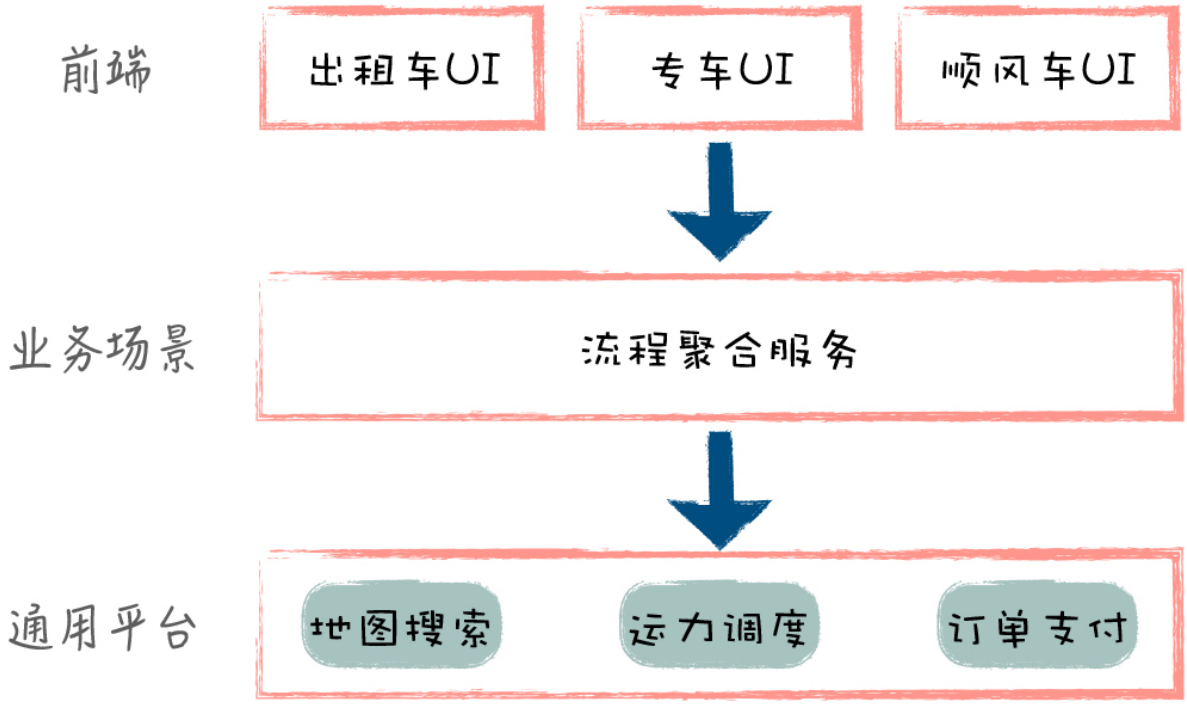
那么，我们能不能对这些类似的模块进行抽象化处理，整合成一个通用的模块呢？答案是可以的，我们可以在模块接口中，通过输入参数标识调用来自哪个业务，是出租车、快车还是顺风车，然后在模块内部，针对不同业务线的差异化部分做针对性处理。结果可能是这个通用模块增加 5% 的逻辑，但避免了 95% 的重复逻辑，这样，经过通用化整合，新的模块以很低的代价，就为多个业务线提供了复用。而且，当新的业务线进来，很可能这个通用化的模块，就已经提供了现成的支持。

通过模块通用化，模块的数量减少了，模块的定位更清晰，概念更完整，职责更聚焦。在实践中，当不同业务线对某个功能需求比较类似时，我们经常会使用这个手段。

平台化整合

平台化是把定位相同的模块组织在一起，以组团的方式对外提供服务。对于外部系统来说，我们可以把这些模块看成是一个整体，一起对业务场景提供全面的支撑。

如下图所示，我们可以看到，地图搜索、运力调度、订单支付，都是各个业务线都需要的基础和通用的业务能力，当我们增加新的业务线时，还是离不开这些基础能力。



所以，我们可以把这些基础模块放在同一层，构成一个基础业务平台。之前，它们是一个个离散的服务，独立地输出能力，现在变成一个大的业务平台，可以提供整体的能力输出。

通过打造业务平台，一方面，我们对多个业务模块进行包装，形成更大粒度的抽象，相当于减少了模块的数量；另一方面，作为平台，它的定位更明确，系统依赖关系也更清晰；而且，如果新的业务线进来，它可以基于业务平台快速落地。

业务平台化是模块依赖关系层次化的一个特例，只是它偏向于基础能力，在实践中，当业务线很多，业务规则很复杂时，我们经常把底层业务能力抽取出来，进行平台化处理。

总结

好了，下面我来总结一下今天所讲的内容。

首先，我们对系统进行建模，系统 = 模块 + 关系，这样会简化你对系统的认识。基于这个模型，我们对模块划分和关系定义提出具体的要求，你可以在实际设计时参考这些要求。

另外，我们深入地分析了扩展性的本质。系统的扩展能力来自于内部模块体系的有序，这样才能低成本地应对业务变化，认识到了这一点，有助于你从根本上理解和重视架构的扩展性设计。

然后，我提供了一个出行平台的例子，来帮助你理解，如何通过模块拆分和整合的手段，具体地设计一个可扩展的架构，希望你能在工作中灵活运用。

最后，给你留一道思考题：你所在公司里有没有类似的肿瘤系统，它包含了太多职责，导致系统内部结构混乱，大家都不敢对它进行调整？

欢迎在留言区和我互动，我会第一时间给你反馈。如果觉得有收获，也欢迎你把这篇文章分享给你的朋友。感谢收听，我们下期再见。

上一篇　02 | 业务架构：作为开发，你真的了解业务吗?

下一篇　04 | 可扩展架构案例（一）：电商平台架构是如何演变的?

精选留言 (13)

写留言

2020-02-26

老师讲的很好，有几个问题请老师指点一下，1. 如果服务拆分的很细了，而且还有大中台提供服务，一般的做架构的话服务是不是可以不用分层和解耦了，如果按业务来看，影响分层和解藕都有哪些因素需要考量。

2. 关于系统重构，业务梳理并划分清楚后，在技术角度需要考量的因素还有那些。

展开

作者回复: 如果有大中台的话，主要的分层和解耦工作已经帮你做了，剩下的是偏应用上层的业务，不用太多考虑再深入拆分。拆分不是越细越好，在人容易理解的前提下，是越粗越好。
系统重构时，除了设计到位，保证系统能够平滑过渡也很重要。技术角度多考虑数据如何迁移，如何实现系统的灰度改造，分阶段上线，减少风险。出问题时，要有B计划兜底。

2

川杰

2020-02-26

有。领导觉得没有必要调整，或者，要你利用业余时间提出方案，甚至最好是能改好，还不能有什么BUG，最关键的是领导还是程序员出身；只能等他的短视自食其果了。

作者回复: 能理解，这个要通过显式立项方式解决。

2　2

孙同学

2020-02-27

https://www.processon.com/view/link/5e51378ce4b0c037b5f9d1e3 今天更新了mindmap。可能我参与的系统比较小吧，都是web项目，按照mvc结构基本就能满足，这几天空闲时间就在梳理项目的结构，大概理清了三十多张表之间的逻辑关系，想根据所学试着提出些理解，但在看代码过程中发现另一个问题，不敢细究代码逻辑，真是纷纷扰扰理不

清头绪，之后只能通过功能界面理解需求，然后看涉及到的数据表的语句，忽略了各种...

展开

1

tt

2020-02-26

总结一下。

可扩展化的过程就是一个抽象的过程。

1、首先是从具体的业务场景中抽象出一个个的业务节点（接口），把类似的接口按照业...

展开

1 1

小伟

2020-02-29

业务和基础模块划分是门艺术，在应用体量不同时的划分可能截然不同，只有多做项目设计才能做成较科学的划分。

展开

小伟

2020-02-29

我司原来是单体应用，现在在做模块化拆分，从单体应用里把模块功能剥离到各自的项目中。但整体还未完成，遗留的单体应用就是个肿瘤，鱼龙混杂，啥功能都有。最终目标就是消灭这个肿瘤。

Geek_c00e0c

2020-02-28

肿瘤系统的形成也有一定因素，比如确实是功能内聚的一个模块，拆分为多个系统的话，可能会有功能不清晰，而且可能会增加了分布式事务的处理。

展开

Kǎfkā²⁰²⁰

2020-02-27

现在就在动手对一个历经好多年有各种补丁的系统做调整，也许最好的方法就是重新设计，重新构建

zeor
2020-02-27

老师您好，请问一个查块的拆分，组合，要什么时候去做

作者回复: 查块?什么意思呢

cricket1981
2020-02-27

业务中台概念是不是就对应文中的业务平台

展开

作者回复: 不一样，这里的业务平台是泛称，业务中台是特指，有一定的要求，后面文章有专门部分讲解业务中台

马哲富
2020-02-26

高屋建瓴，醍醐灌顶；系统=模块+依赖关系

展开

李博
2020-02-26

很棒！

展开

Alex
2020-02-26

系统的扩展能力来自于内部模块体系的有序，这样才能低成本地应对业务变化。这也是架构师的职责所在。