218 words • 2~4 min read

我如何向HRMM介绍Microservice

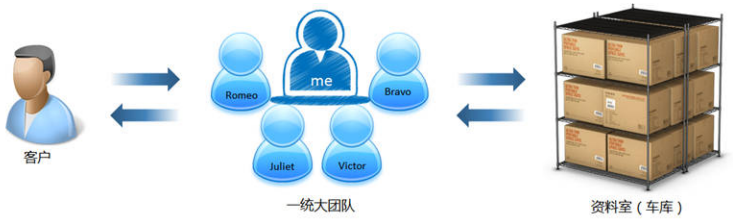
一天我司招才猫姐(HR 大人)问我，你给我解释一下 Microservice 是什么吧。故成此文。一切都是从一个创业公司开始的。

第一章：从集中到分权

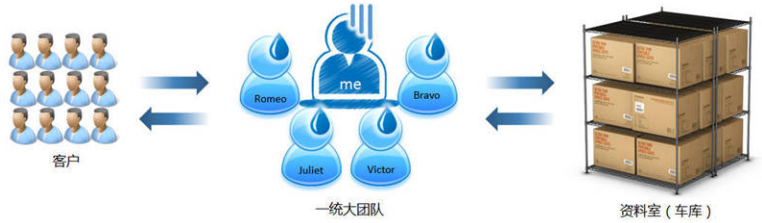
最近的创业潮非常火爆，我禁不住诱惑也掺和了进去，创建了一家公司。为了表达我的抱负，取千秋万代，一统江湖之意。我给公司定下了一个非常响亮的名字叫做——一统。

故事

虽说叫做一统但是凡是都要从头开始，公司成立之初有五个成员：罗密欧，朱丽叶，维克多，布拉伯还有老大——我。我们五个人都是工程师出身，自身具备了非常优秀的学习能力，各个都是从业务到代码的好手，五个人一起做策划，搞市场，写程序，做运维，面对客户；可谓你中有我，我中有你，努力拼搏，好不热闹。为了吉利，我们找了一个车库作为机房和资料室，上至合同，下至代码，全都放在里面。这就是一统最初的样子。

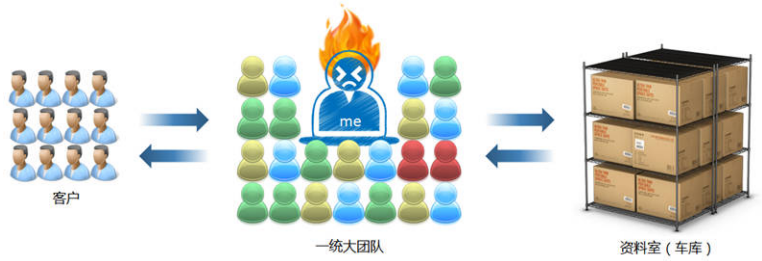


虽然创业艰难，大部分公司都在前两年倒下了，但是我大一统不但没有倒掉，还意外的得到了增长。客户从当初的一家猛增到一百家。这样即使我们的团队再出色也应付不了这么多的客户了。挣钱还是要命呢？

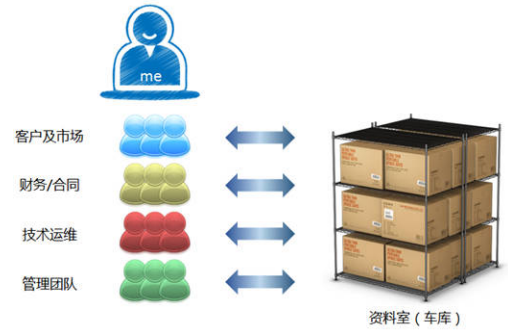


命得要，钱也得挣！怎么办呢？招人吧！我们绞尽脑汁搜罗（挖角）市场上的人才，组成了巨型一统团队。我们认为我们团队中的任何成员都应该跟初创时期的五个人一样，能攻能守，内外兼修，但是.....我们发现这是不可能的，有些人擅长写代码而非面对客户，有些人善于做市场但是不喜欢算财务。后来出现了更加让人挠头的事情，有一个财务流程，问了5-6个人竟然没有一个人能够完整的串起来。于是，我一个一个的问，最后才把整个流程从这些碎片知识里面串联起来。

这样的团队的服务质量可想而知，不久就接到了数十件客户投诉。竞争对手趁机抢占市场，一个欣欣向荣的公司瞬间就摇摇欲坠了。



为了留住客户，我们必须在扩张的同时能够保证服务质量。我发现目前最重要的问题就是职责不清晰，大家不知道自己应该干什么也不清楚怎么干，于是我抽调了各个业务部分的精干力量，总结流程，形成了客户及市场、财务/合同、技术运维、管理团队四个独立的业务部分。采取内部招聘的方式将人力分配到了这四个部门中。我期望将大家从纷繁的知识体系中解脱出来，每一个不需要了解那么多的知识，集中力量关注自己的问题以提升效率和服务质量。



在此次结构调整之后，大家的工作效率明显提升了。抱怨知识结构太复杂，无法短期适应工作的声音消失了。一个月之后，我们做了一个抽样调查，发现大家对自己的工作范围和内容都了如指掌。一些客户又重新和我们签订了合同。

正当我沾沾自喜的时候，发生了一个重大事件。由于我们的资料室是对全公司开放的，任何成员都可以查看或修订其中的信息。客户及市场部的一个员工平时非常好学，对财务方面的知识掌握的非常系统。有一天，公司急需草拟一份财务清单，但是这个任务非常耗时，财务部门的同时当时正在月末审核，无暇抽身。这位热心的同学就凭借自己出色的能力从资料室取来相应的材料完成了清单。三天后，财务部的罗密欧准备细化这个清单。但是清单的内容让他着实吃了一惊——清单上的填写内容和他们部门内约定格格不入。罗密欧只得自己重新完成了清单。一来一去让他的工作耽搁了一天，罗密欧向我抱怨道。无独有偶，其他部门也发生了同样的事情。这让我意识到只是把人员的职责进行划分并不能彻底解决问题。我们不能再继续混用一个资料室了，因为这样人人都可以任意修改各种资料，安全性不好不说还会造成填写格式混乱。于是我把财务部的资料放在了另一个独立的屋子里，并给罗密欧单独配了钥匙。这样，任何想填写财务清单的人只能找财务部的人所要单据，而当单据缴回的时候也必须经过财务部的审核。鉴于财务部每个月底都会很忙，在那时我会临时抽调人手去帮忙。

我希望对其他部门做相应的整改，但这种动作幅度毕竟很大，因此，一段时间内还会有多个部门共用资料室。经过一个月的努力，我们最终还是淘汰了公用资料室。为每一个部门都配备了独立资料室。虽然需要缴纳

更多的房租，但是各部门再也没有犯之前的错误。

从故事到项目

大多数项目也会像我们的创业公司一样，一开始大家一起干活，每一个人都是冲锋陷阵多面手。大家一起组成了应用程序的全部，而我们的车库就是数据库。这种组织结构代表了典型的 monolithic application。这种系统的逻辑架构是类似这样的。

在项目的早起，业务简单，吞吐不大，这种结构清晰，易于理解的架构非常实用。但是随着业务的增大，混在一起的代码不易理解。而最容易想到的解决办法就进行职责的划分。一统公司一开始在维持车库结构的情况下仅仅把人员上做了拆分。这种情形在实际项目中也存在。系统进入了多个服务共享一个数据库的阶段，而集成点在数据库上。

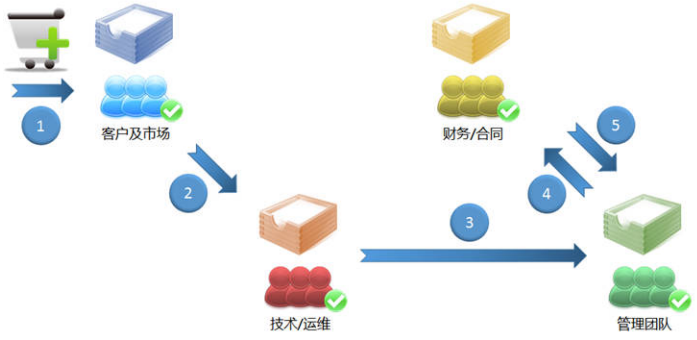
这种职责划分但又维持数据库集成的方式只应当作为过度阶段存在。程序永远都是逻辑+数据，而数据的混杂谈不上职责的独立。长期维持这种数据集成的状态容易出现业务下行，数据表达不一致等问题。从逻辑+数据的整体划分边界（称为模块，或者服务）势在必行。而在边界划定之后，就需要考虑独立的服务之间如何进行协作了。

第二章：协作

现在我们的资料室都独立了，再也没有办法像以前一样有需要就去公共资料室里取资料了。那么我们应该如何进行协作呢？

故事

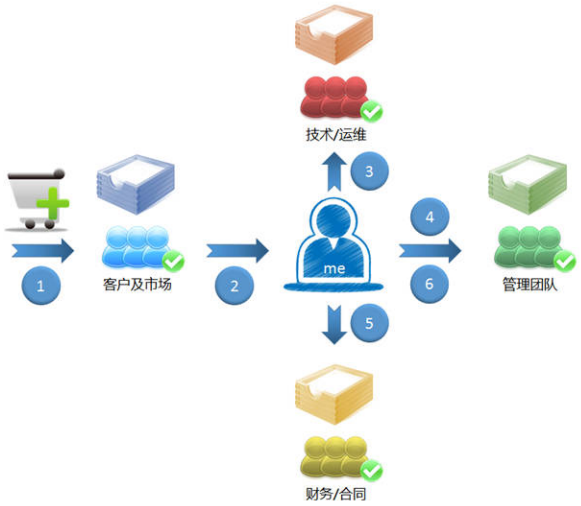
一个非常自然的想法是把各部门用业务流程穿起来。例如，如果洽谈一个订单，先由客户及市场部进行调查和谈判，然后由技术部制定解决方案，管理部审批之后由财务合同部拟定合同，最终递交管理部签署。这种协作方式我们不妨叫他 串联协作。



为了将这一方案执行下去，我们对各个部门的人员进行了培训。例如，对于对于技术部，如果是洽谈一个订单，那么技术部需要给出解决方案，完成之后需要将方案递交管理部审批。类似的业务有很多，每一个业务各个部门任务都不同，而下游接收的部门也不一样。但是我的同事们还是克服了困难，将它们烂熟于胸。

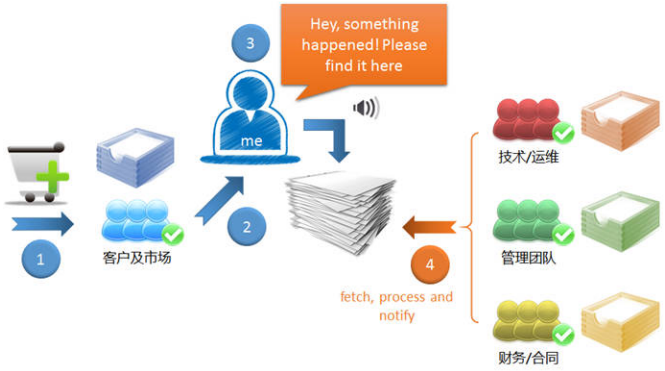
业务变化是最平常的，一变就是一大把。现在对于洽谈订单的业务，我们需要在技术方案给出之后先给财务部进行预算审核再递交管理部。这需要重新培训三个部门的人，洗脑一样的把他们之前的流程抹掉。一个业务变化还好，但是一下二三十个业务发生变化让大家抓狂。甚至有人说这和之前一大坨人一起做所有事情的时候没有什么差别。看来这种业务串联的方式是行不通了。

所谓我不入地狱谁入地狱，这种情况下我挺身而出，入住市场部——因为这个部门是直接为客户服务的。我对所有的流程了如指掌，当业务来了的时候，有我去对业务进行协调。例如，当一个洽谈订单的业务到来的时候，我会先将他交给客户市场部进行谈判；结束后市场部将需求和意向书交给我，我再把需求交给技术部去制定解决方案；方案制定完毕之后技术部会把方案返回给我，我再把结果交给财务部审核...如此进行。这样，每一个部门就不会由于业务变化重新接受培训了，也不用记住他们的下游应该是谁。大家觉得，职责明确多了，工作轻松多了。



这种协作方式我们不妨称之为 业务调度员 式的协作。

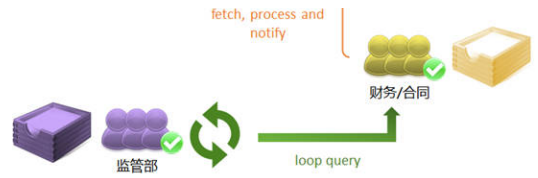
现在我俨然成为了流程的中心，其他业务部门不需要关注业务流程的变化。这给我们增加了很多灵活性，因为我一个人的变化速度要比一群人的变化速度快得多（你是独裁者吗喂）。但是我的大脑总是有限的，一年之后，经历了三四轮的业务变化，我已经开始不能准确的回忆某些业务细节。我不得不开始频繁的查询我的业务笔记来确定我的下一步操作；此外，不停的往各部门送信也令我不堪重负。这时候，我希望其他人能够为我分担。我决定化被动为主动，不由我主动联络各个部门，而由各个部门主动接受任务。我在公司安装了一个 大喇叭，话筒就在我的办公室里。当一个订单洽谈业务到来的时候我就朝着喇叭嚎：新订单来啦。客户和市场部专门关注新订单，因为按照流程，订单到来之后市场部要尽快投入谈判。于是他们会主动开始行动。当市场部工作完毕之后，他们会将谈判的结果、需求以及意向书拿到我的办公室里。我会再嚎：需求来啦。此时，技术部会从我这里拿走这些资料资料并开始工作。如此往复，直到项目完成。



这种协作方式令我不必再操心信息在各部门之间的流转。各部门知道他们应该何时介入，我只需要对着大喇叭喊自然会有相关的部门将活干完。这样，我作为业务的中转中心，工作量不会随着业务的增长显著的增大（只要喊话就行了，至多就是增加喊话种类），而各部门也不用关心自己的下游到底是谁，只需要关心我喊的话就行了。这种协作方式不妨称之为 公司广播。看起来这是一种非常方便的形式。但是这真的就又快又省吗？

有一天，来了一个新的订单。在我接到意向书和需求之后，我照例喊话：需求来啦！接着，我就出去吃饭了（真是资本家啊你）。等我回到了办公室，发现文档已经被拿走了，而结果还没有送过来。时间一天一天过去了，我手头的新订单越积越多，需求文档和意向书源源不断的送过来。到了第三天我实在是受不了了，想去查找文档的去向和任务的完成状况。这时候我才发现我根本无从下手，因为我不知道文档是谁拿走的，于是我便一个部门一个部门的去询问。可是由于业务的发展，现在我们已经有20多个部门了，这种非常规的询问不仅让我跑断了腿，而且为了查证，需要翻阅各部门成吨的业务日志，各部门的部长对此也颇有微辞。我终于意识到——这种协作方式在令结构松散灵活的同时也极大的增加了监管的难度。必须采取额外的投入来弥补这一短板。

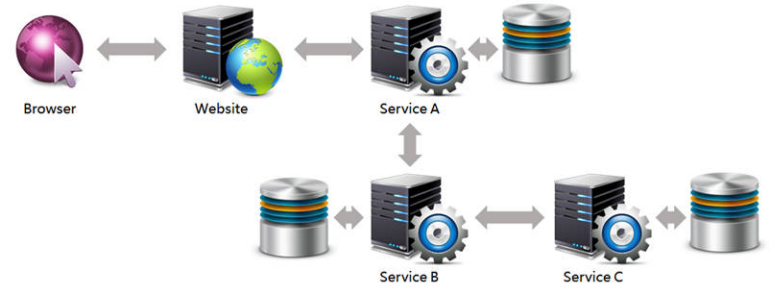
于是，专门监视各部门动向的“纪检委”：监管部出现了。一开始，我只是想确认一下各部门运作是否正常，有没有由于天灾人祸而出现团灭的现象。出于不对各部门添加新的压力的愿望，监管部会定期去确定各部门走一圈看看是否都在好好干活（真是讨人嫌的部门啊），就像这样。



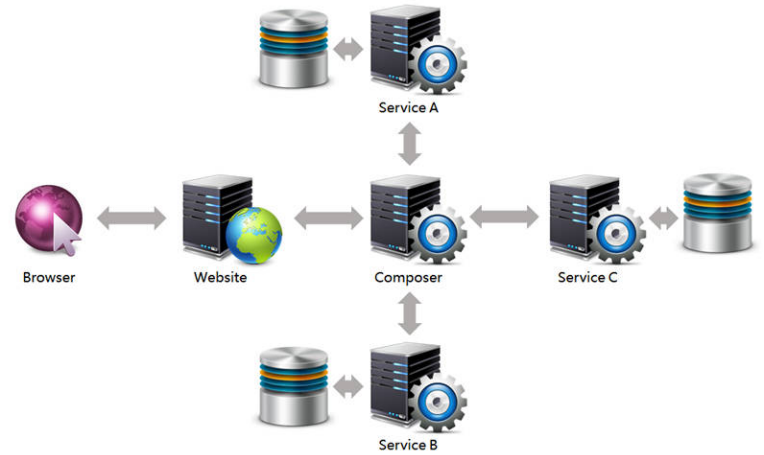
但是随着业务的发展，我希望得到各个部门更加详尽的信息，例如，各部门是否积压了大量任务而需要帮助，在处理过程中是否由于流程不合理而无法继续下去等等。需要收集的数据已经远远超过了监管部跑腿的速度。怎么办呢？如果我手头没有什么钱，我可能会降低监管部工作的周期，例如之前是半天一次，现在改成两天一次。但是我是土豪，于是我制定了监管汇报单，强制各部门在状况出现问题的时候主动向监管部汇报，就像这样。

从故事到项目

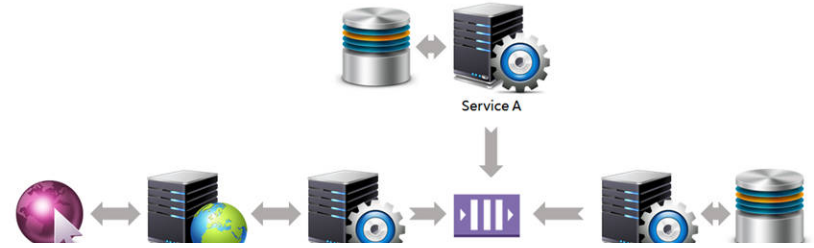
我们真正的开始考虑服务之间如何进行协作了。和一统公司的做法一样，最容易想到的就是业务串联。业务首先由第一个服务处理，处理之后第一个服务调用第二个服务继续处理，直至业务全部处理完毕。这样，我们就得到了这样的系统架构：

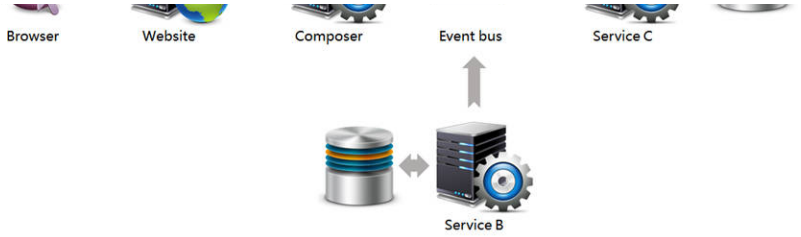


正如故事中说的一样，这种业务的串联难以应对业务流发生的变化。同时，由于服务之间互相直接耦合，集成点多，难以做到独立进行部署。业务调度员 的方式是一种很好的改进。如果你还记得那位勤劳的调度员（我），那么你就一定认识 Composer 这个特殊的服务：我们使用 它进行业务流的分发与控制。这样，服务仅仅和 composer 进行对接，不需要考虑业务流的问题。既明确了职责又可以进行独立部署。以下就是这种协作方式的架构：



而最复杂的 公司广播 则是变直接调用为事件触发。使用 事件 隔离了各自的模型。系统结构更加松散灵活。





隔离和协作的矛盾

从踌躇满志的创业开始到现在，我感慨于业务的扩大和公司规模的发展。也有一些会议例如 PCon 希望我们能够去介绍的结构划分和协作模式。但是每当我在工作之余静静思考的，却感到一些厌倦。我们的协作真的好吗？随着职责的划分，我们越来越专业化，互不影响的工作方式极大的增加了我们的效率，良好的隔离让我们的失误不至于扩散并能够横向扩展；而监控系统也告诉我们一切尽在掌握。但是隔离同时也是壁垒。为了协作，我们在这些壁垒上开了些小孔，为了严格控制进出，我们制定了越来越多的规条，例如，信件应该怎么写，单据应该怎么填。回头看去，成百上千的规约连我也不能尽数，当我需要对我的公司做出变化的时候，如果变化仅仅发生在各个部门内部，这种结构的优势就显露无遗。但是如果设计到部门之间的协作，甚至部门的拆分合并，就会变得异常艰难。由于部门和规约的耦合是存在于脑内的，并不显露在外，修改已经存在的规约和部门结构往往不现实，只能花更多的钱去组建新的部门，逐步介入公司业务后淘汰旧有的部门。而这种周期动辄是以月计算的。有的时候，可能宁可去接受新的方式从头开始，卖掉旧的公司，创建新的公司，然后继续轮回。

隔离既创造了灵活的单点变化也扩展造就了整体的僵化。细小的部门划分不能解决这个问题，因为部门越多，规约越多，实施成本越高（在例子中我们并没有考虑资料室的房租，监控的支出，但是现实中这往往是一个决定性因素）。单个部门结构的简化造就了整体的复杂。

因此，分部门就一定好吗？业务调度员就没有公司广播好吗？答案应该来源于你的现实，而不是理论。如果让我重来一次，我可能会慢一点，再慢一点。未来无法预计，针对当下的痛点做出反应可能不是最优的，但是却是一个容易理解的思路。

实际项目中也是一样。就像武侠小说中的一样，威力越大的招数其使用限制也就越多。一方面，系统结构越来松散越灵活，变化变得容易；另一方面，系统越来越复杂，维护越来越困难。当维护困难到一定程度不得不将其提上议事日程的时候专事监督与维护的工作就出现了，这种维护不同以往。维护者既要有维护知识又必须了解这个复杂系统的运作方式。别忘了，虚拟化、云、Devops 除了概念上的光鲜又复杂又花钱啊！

慢慢进化，慎重的考虑并决定是否选用 Microservice，然后做好维护吧。

第三章：Microservice

在前面的内容中，哪种方式可以称为 microservice 架构呢？故事里面公司组织结构演化分成了几个部分，并不是每一个阶段代表的结构都可以称之为 microservice：

- 第一个阶段是多个部门共用一个资料室，这种情况往往并没有摆脱数据库集成的事实（有些项目使用 Schema 对访问进行隔离，不在此列），不能称为 microservice；
- 第二个阶段是各个部门有自己的资料室，这是一个非常重要的改变，直到此时，各个部门才实现了完全的隔离，但是还足以称为 microservice；
- 第三个阶段分三类：
  - 首先是串联协作。在这种方式下，一个部门的变化往往影响了其他部门，因为每一个部门都需要知道业务的上游部门和下游部门，因此有比较紧密的耦合关系，无法独立变化。因此不足以称为 microservice；
  - 其次是业务调度员。这种方式下部门自身的变化不会扩散到其他部门，因此是可以独立决定和操作，只要部门足够小，这种结构可以称之为 microservice 了；
  - 第三是公司广播。这种方式下的部门耦合更加松散，可以独立进行变化，在部门足够小的前提下可以称之为 microservice。

我们的初衷是什么呢？构造一个系统能横向扩展以满足业务伸缩性，提供灵活变化的能力，又希望变化的影响不要扩散。因此，如果一个架构可以称之为 microservice 架构，那么意味着：

- 每一个 Service 可以进行独立部署；
- 每一个 Service 都足够小，完成完整的定义清晰的职责；

业务调度员和公司广播两个例子的组织结构都可以成为 microservice 架构。但是他们之间并没有绝对的优劣之分。选用哪一种应当取决于实际要求。

写在最后

Microservice 不是一种科学，而仅仅是实践。就像是 OO 一样。最终应当是需求，是人，决定架构而非架构决定需求。Microservice 是为 Business Capability 而建，是根据实际（或者痛点）做出的抉择。他解决了一些问题，但是并不能肯定就是今后软件的发展方向。硬件的革新——不论是近期的电池技术的发展，还是近乎黑科技的量子态传输，量子计算都有可能影响甚至颠覆今天形成的软件开发手段。就像是当年大家为了追求极致的执行效率试图将代码塞进 64K 的代码段一样，我们今天奉为经典的东西未来可能只是茶余饭后的谈资。我们能做的只有保持开放的心态，坚持辩证的观点，坚持从实际出发（我当年政治一定得高分了），寻找出最合适的解决方案。这也就是我们不称自己为码农的理由之一吧。

新兴技术 | 刘夏 | 2015年9月16日 · pm2:42