二

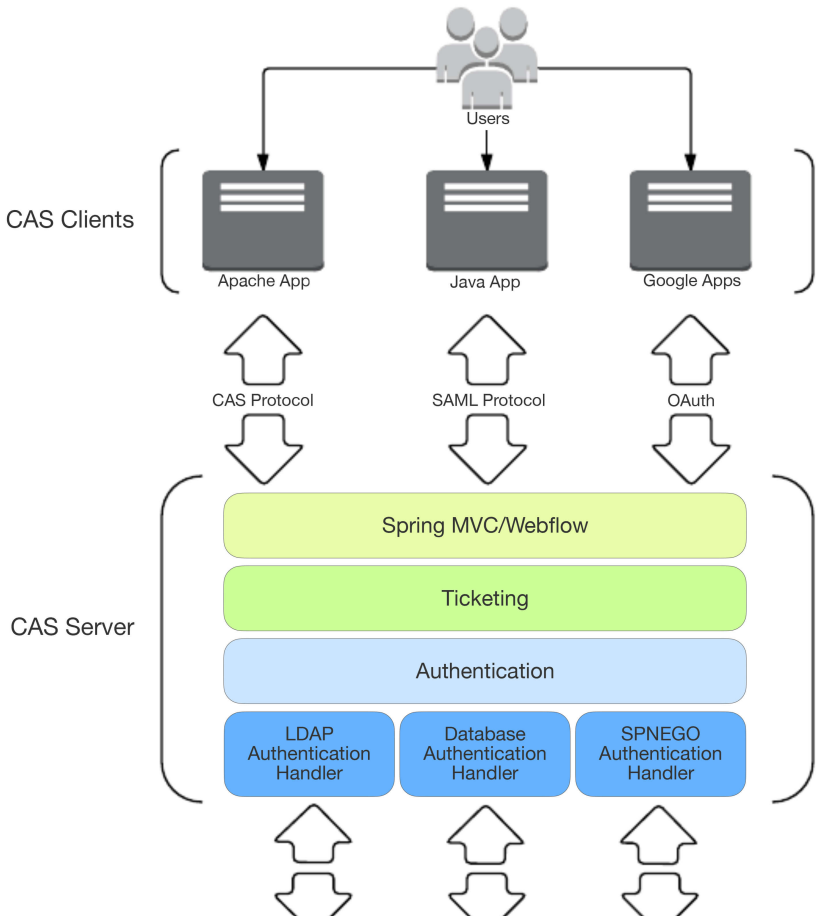
43 互联网架构模板：“用户层”和“业务层”技术

43 互联网架构模板：“用户层”和“业务层”技术上一期，我从计算机网络层的角度谈了应对“高性能”和“高可用”的整体架构设计。今天，我将从“用户层”和“业务层”的角度谈谈常见的应用场景和关键技术。

用户层技术

1. 用户管理

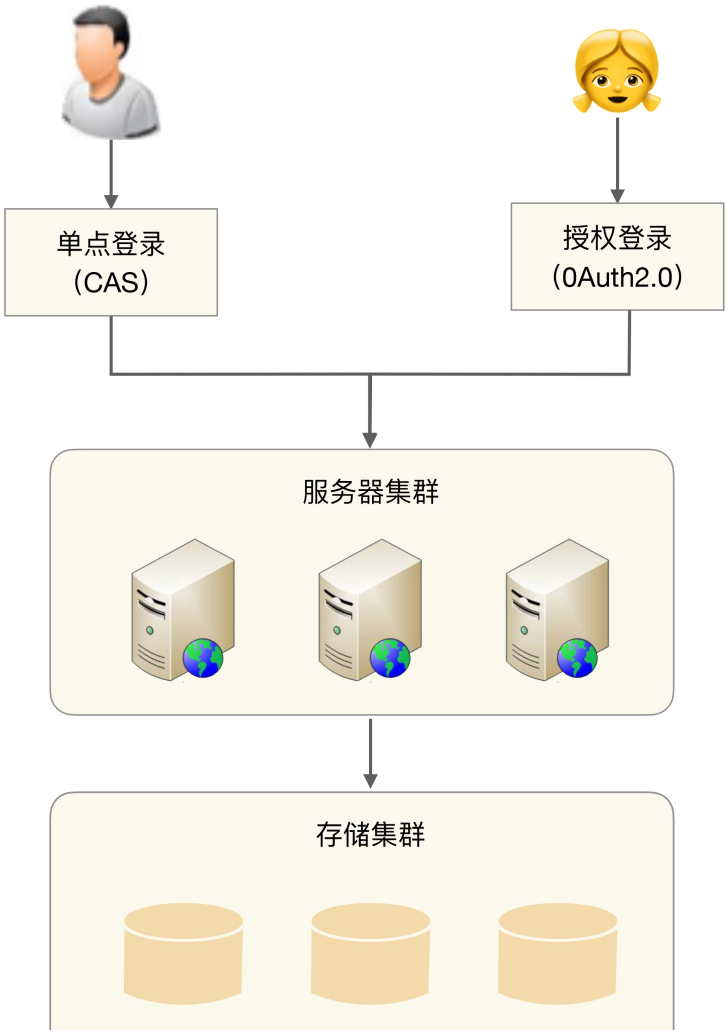
互联网业务的一个典型特征就是通过互联网将众多分散的用户连接起来，因此用户管理是互联网业务必不可少的一部分。



除此之外，当业务做大成为了平台后，开放成为了促进业务进一步发展的手段，需要允许第三方应用接入，由此引申出用户管理的第二个目标：**授权登录**。现在最流行的授权登录就是 OAuth 2.0 协议，基本上已经成为了事实上的标准，如果要做开放平台，则最好用这个协议，私有协议漏洞多，第三方接入也麻烦。

用户管理系统面临的主要问题是用户数巨大，一般至少千万级，QQ、微信、支付宝这种巨无霸应用都是亿级用户。不过也不要被这个数据给吓倒了，用户管理虽然数据量巨大，但实现起来并不难，原因是什么呢？ 因为用户数据量虽然大，但是不同用户之间没有太强的业务关联，A 用户登录和 B 用户登录基本没有关系。因此虽然数据量巨大，但我们用一个简单的负载均衡架构就能轻松应对。

用户管理的基本架构如下：



2. 消息推送

消息推送根据不同的途径，分为短信、邮件、站内信、App 推送。除了 App，不同的途径基本上调用不同的 API 即可完成，技术上没有什么难度。例如，短信需要依赖运营商的短信接口，邮件需要依赖邮件服务商的邮件接口，站内信是系统提供的消息通知功能。

App 目前主要分为 iOS 和 Android 推送，iOS 系统比较规范和封闭，基本上只能使用苹果的 APNS；但 Android 就不一样了，在国外，用 GCM 和 APNS 差别不大；但是在国内，情况就复杂多了：首先是 GCM 不能用；其次是各个手机厂商都有自己的定制的 Android，消息推送实现也不完全一样。因此 Android 的消息推送就五花八门了，大部分有实力的大厂，都会自己实现一套消息推送机制，例如阿里云移动推送、腾讯信鸽推送、百度云推送；也有第三方公司提供商业推送服务，例如友盟推送、极光推送等。

通常情况下，对于中小公司，如果不涉及敏感数据，Android 系统上推荐使用第三方推送服务，因为毕竟是专业做推送服务的，消息到达率是有一定保证的。

如果涉及敏感数据，需要自己实现消息推送，这时就有一定的技术挑战了。消息推送主要包含 3 个功能：设备管理（唯一标识、注册、注销）、连接管理和消息管理，技术上面临的主要挑战有：

海量设备和用户管理

消息推送的设备数量众多，存储和管理这些设备是比较复杂的；同时，为了针对不同用户进行不同的业务推广，还需要收集用户的一些信息，简单来说就是将用户和设备关联起来，需要提取用户特征对用户进行分类或者打标签等。

连接保活

要想推送消息必须有连接通道，但是应用又不可能一直在前台运行，大部分设备为了省电省流量等原因都会限制应用后台运行，限制应用后台运行后连接通道可能就被中断了，导致消息无法及时的送达。连接保活是整个消息推送设计中细节和黑科技最多的地方，例如应用互相拉起、找手机厂商开白名单等。

消息管理

实际业务运营过程中，并不是每个消息都需要发送给每个用户，而是可能根据用户的特征，选择一些用户进行消息推送。由于用户特征变化很大，各种排列组合都有可能，将消息推送给哪些用户这部分的逻辑要设计得非常灵活，才能支撑花样繁多的业务需求，具体的设计方案可以采取规则引擎之类的微内核架构技术。

3. 存储云、图片云

互联网业务场景中，用户会上传多种类型的文件数据，例如微信用户发朋友圈时上传图片，微博用户发微博时上传图片、视频，优酷用户上传视频，淘宝卖家上传商品图片等，这些文件具备几个典型特点：

文件体积小：大部分图片是几百 KB 到几 MB，短视频播放时间也是在几分钟内。

访问有时效性：大部分文件是刚上传的时候访问最多，随着时间的推移访问量越来越小。

为了满足用户的文件上传和存储需求，需要对用户提供文件存储和访问功能，这里就需要用到前面我在[专栏第 40 期]介绍“存储层”技术时提到的“小文件存储”技术。简单来说，存储云和图片云通常的实现都是“CDN + 小文件存储”，现在有了“云”之后，除非 BAT 级别，一般不建议自己再重复造轮子了，直接买云服务可能是最快也是最经济的方式。

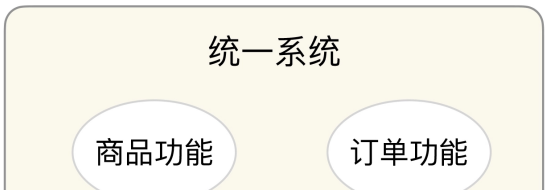
既然存储云和图片云都是基于“CDN + 小文件存储”的技术，为何不统一一套系统，而将其拆分为两个系统呢？这是因为“图片”业务的复杂性导致的，普通的文件基本上提供存储和访问就够了，而图片涉及的业务会更多，包括裁剪、压缩、美化、审核、水印等处理，因此通常情况下图片云会拆分为独立的系统对用户提供服务。

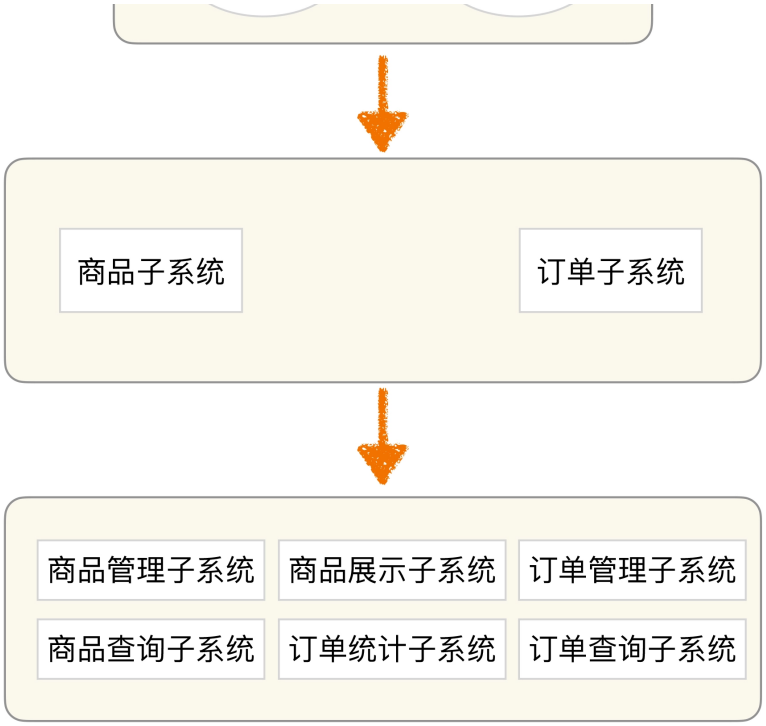
业务层技术

互联网的业务千差万别，不同的业务分解下来有不同的系统，所以业务层没有办法提炼一些公共的系统或者组件。抛开业务上的差异，各个互联网业务发展最终面临的问题都是类似的：业务复杂度越来越高。也就是说，业务层面对的主要技术挑战是“复杂度”。

复杂度越来越高的一个主要原因就是系统越来越庞大，业务越来越多。幸运的是，面对业务层的技术挑战，我们有一把“屠龙宝刀”，不管什么业务难题，用上“屠龙宝刀”问题都能迎刃而解。这把“屠龙宝刀”就是“拆”，化整为零、分而治之，将整体复杂性分散到多个子业务或者子系统里面去。具体拆的方式你可以查看专栏前面可扩展架构模式部分的分层架构、微服务、微内核等。

我以一个简单的电商系统为例，如下图所示：





我这个模拟的电商系统经历了 3 个发展阶段：

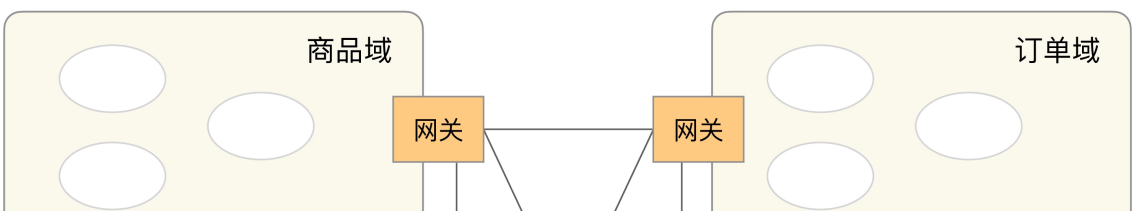
第一阶段：所有功能都在 1 个系统里面。

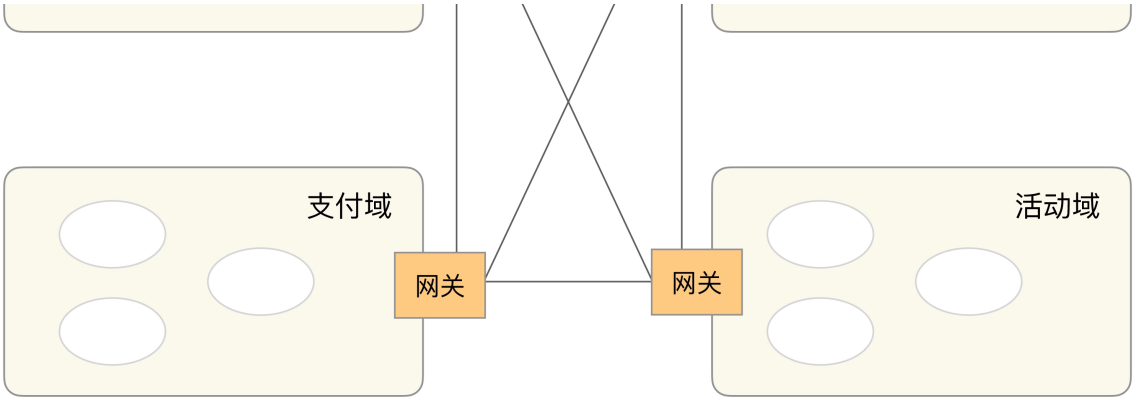
第二阶段：将商品和订单拆分到 2 个子系统里面。

第三阶段：商品子系统和订单子系统分别拆分成了更小的 6 个子系统。

上面只是个样例，实际上随着业务的发展，子系统会越来越多，据说淘宝内部大大小小的已经有成百上千的子系统了。

随着子系统数量越来越多，如果达到几百上千，另外一个复杂度问题又会凸显出来：子系统数量太多，已经没有人能够说清楚业务的调用流程了，出了问题排查也会特别复杂。此时应该怎么处理呢，总不可能又将子系统合成大系统吧？最终答案还是“合”，正所谓“合久必分、分久必合”，但合的方式不一样，此时采取的“合”的方式是按照“高内聚、低耦合”的原则，将职责关联比较强的子系统合成一个**虚拟业务域**，然后通过网关对外统一呈现，类似于设计模式中的 Facade 模式。同样以电商为样例，采用虚拟业务域后，其架构如下：





小结

今天我为你讲了互联网架构模板中的用户层技术和业务层技术，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，虚拟业务域划分的粒度需要粗一些还是要细一些？你建议虚拟业务域的数量大概是多少，理由是什么？

上一页 下一页