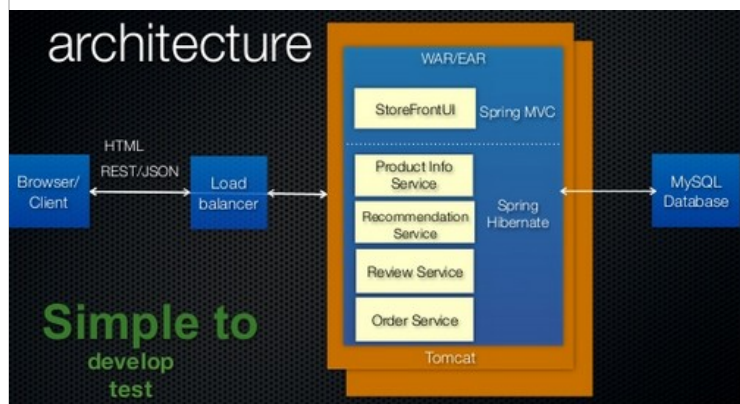




这是来自《POJOs In Action》作者和CloudFoundry原始创始人Chris Richardson的一篇谈论微服务PPT，结合**DDD**和**事件驱动**，比较全面和可落地。大意翻译如下：

以一个在线商店为案例，这是一个SpringMVC+Hibernate的简单应用，只有一个StoreFrontUI和四个服务，在这种简单情况下各方面都还好。

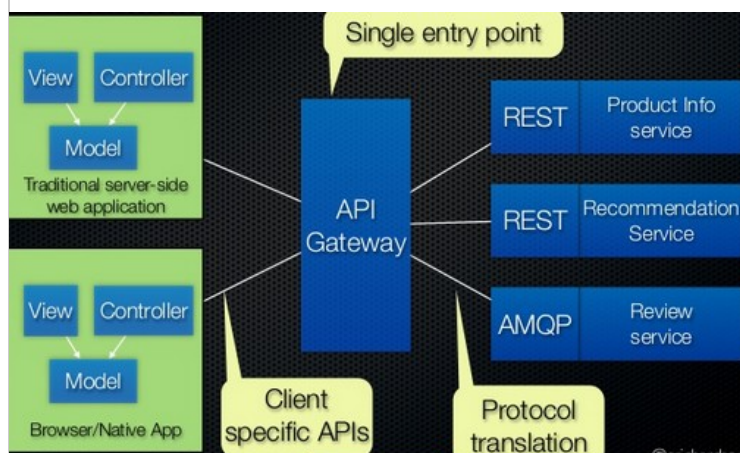


但是当系统开发变得复杂时，这样的系统就难于测试，难于适应频繁部署，逐步会降低开发效率，开发人员之间难以协调，这是传统铁板一块整体式系统的问题。

通过引入微服务，能让应用变得更小更简单，易于理解和开发，减少启动时间，这对于在云平台上运行很重要，远离Jar和ClassPath地狱，再也不需要OSGI了(banq注：这点我早在微服务刚出来就预感到了)，这样让开发变得可扩展，开发部署和扩展服务都变得独立，相互没有依赖。这样从一个单一技术堆栈演变成模块多框架的开发堆栈，每个服务都可以使用不同的技术堆栈来实现开发。这样可以引入尝试更多的新技术如Scala Akka Play Node.js等等。

1. 客户端和服务之间是如何实现交互呢？

如果前端直接连接后端**RESTful**服务会显得琐碎，也会增加网络的通讯次数，提高延时，降低性能，推荐在两者之间引入API网关代理。

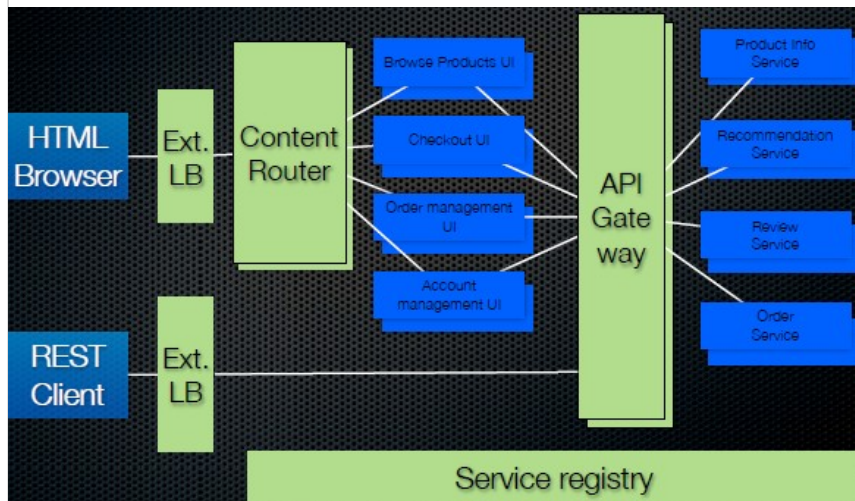


这个API Gateway可以使用Node.JS来实现，能够实现REST Proxy和Event Publishing事件发布。成熟应用可见: **Netflix 优化的API**，总之这个API网关需要满足性能和可扩展**伸缩性**：非堵塞和异步**并发编程**。

2.后端服务器之间如何通讯？

可直接通过Http或者异步消息系统，两者各有利弊。

最后别忘记在前端和后端之间引入负载均衡器，可以使用专门的负载均衡或者客户端负载均衡。最后整个架构如下，可兼容传统的服务器端MVC(也称为Web端)和现代移动设备。



3.分散的数据管理

微服务引入可以为多样化持久保存数据提供可能，持久层可以使用传统关系数据库和**NoSQL**，以订单为案例，订单中含有客户信息，我们提供订单服务和客户服务两个微服务，使得订单数据库与客户数据库分离，但是这样带来一些问题，如果一个服务需要读取另外一个服务中的数据怎么办？或者需要跨服务进行事务性的更新怎么办？

比如我们下订单之前首先要获得客户的信用卡是否有余额，这就跨越了订单服务和客户服务两个微服务，如果采取一个服务直接通过方法调用另外一个服务的传统方式，这是一种Pull数据方式，虽然简单但是可能会降低可用性，增加延时，万一另外被调用的一个服务忙碌不响应怎么办呢？

推荐在订单这个实体中复制客户信用限制这个数据(banq注：实际是**DDD**两个聚合根，信用限制作为作为值对象在两个聚合中复杂。)

Chris在这里推荐了**DDD**的有界上下文：不同的服务器是对同一个领域对象不同的视角，客户管理微服务是从客户复杂的视角出发，通过这个微服务我们能管理复杂的客户用户资料。当然不同的服务也可以对应不同的领域模型。

上述方法解决了跨服务读取的问题，那么如何解决跨服务的更新操作呢？

如果使用传统的**分布式**事务机制，无疑增加复杂性，提高了延时。Chris推荐使用最终一致性(banq注：可参考**CAP定理**和**Base思想**)，当数据发送变化时，由服务发出事件(banq注：领域事件可以由聚合根实体发出事件，也可由微服务发出事件)。

当领域对象发生变化时，发出的事件必须是原子性的。Chris推荐使用Eventsourcing，这是一个以事件为中心是和领域模型设计的方案，能够让聚合根处理各种输入命令，产生相应的输出事件。记录的是事件而不是状态，为了获得某个时刻状态，可以通过重演事件流得到，就像我们看一部网络电影，我们可以通过直接快进到影片中间某个阶段。存储事件的数据库称为Event Store \approx database + message broker。

EventStoreAPI如下：

```
trait EventStore {
  def save[T](entityId: Id, events: Seq[Event]): T
  def update[T](entityId: Id,
    version: EntityVersion, events: Seq[Event]): T
  def load[T](entityType: Class[T], entityId: EntityId): T
  def subscribe(...) : ...
  ..
}
```

[该贴被banq于2014-05-20 09:18修改过]
[该贴被banq于2014-05-20 09:19修改过]





微服务架构使得我们的应用可以方便快速在亚马逊和GAE的**云计算**平台上运行，当然也包括私有的PaaS平台，从长期来看，微服务架构必将加速 “云计算PaaS替代传统Java等中间件” 这一趋势。

Chris作为**云计算**CloudFoundry创始人非常明白这一趋势，因此发表了这篇力挺微服务的架构。

因为Java中间件技术无疑是**JavaEE**那些技术，包括**分布式**事务 线程池 对象池 消息总线等等，这些技术有的被微服务使用EventSourcing巧妙回避了，有的被PaaS替代了，比如负载均衡和消息总线等等。**(JavaEE应用服务器死了)**

当你的应用切分得可扩展可部署时，你就可以充分利用**云计算**平台带给你的强大伸缩扩展能力。

 [EDA事件驱动\(105\)](#) [DDD上下文\(21\)](#) [EventSourcing\(86\)](#) [微服务\(37\)](#)

 [发表 回复](#)

