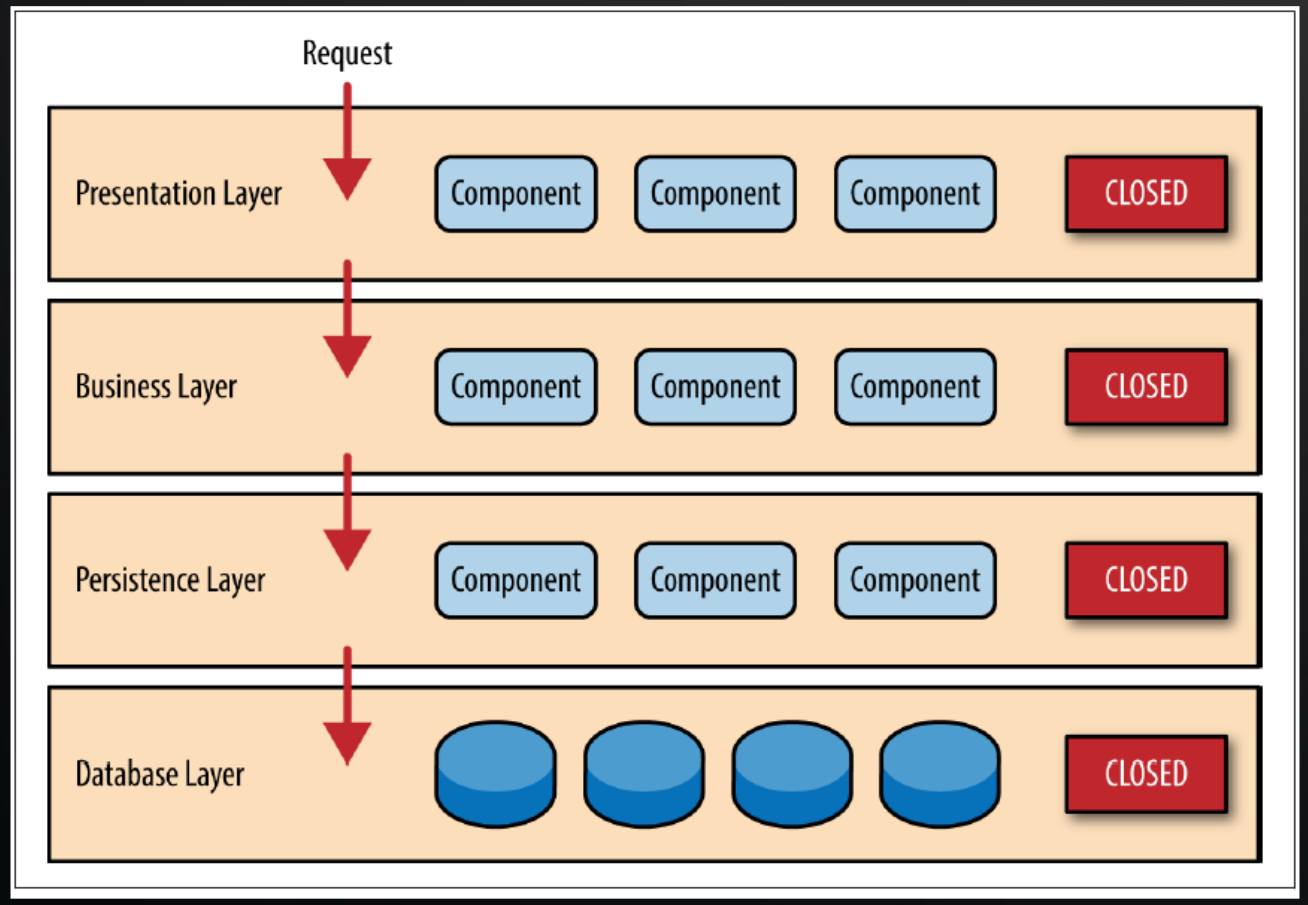
Software Architecture Introduction

（二）常见架构方法

王丰 （Feng WANG）

01 分层架构



Presentation: 用户交互
Business: 实现业务逻辑
Persistence: 提供数据
Database: 保存数据

关键点：

- 层数和内容，根据需要而设计
- 清晰定义每层职责，对上接口形式统一
- 每层独立，可以单独开发、测试、发布
- 从下往上，从稳定到变化

优点

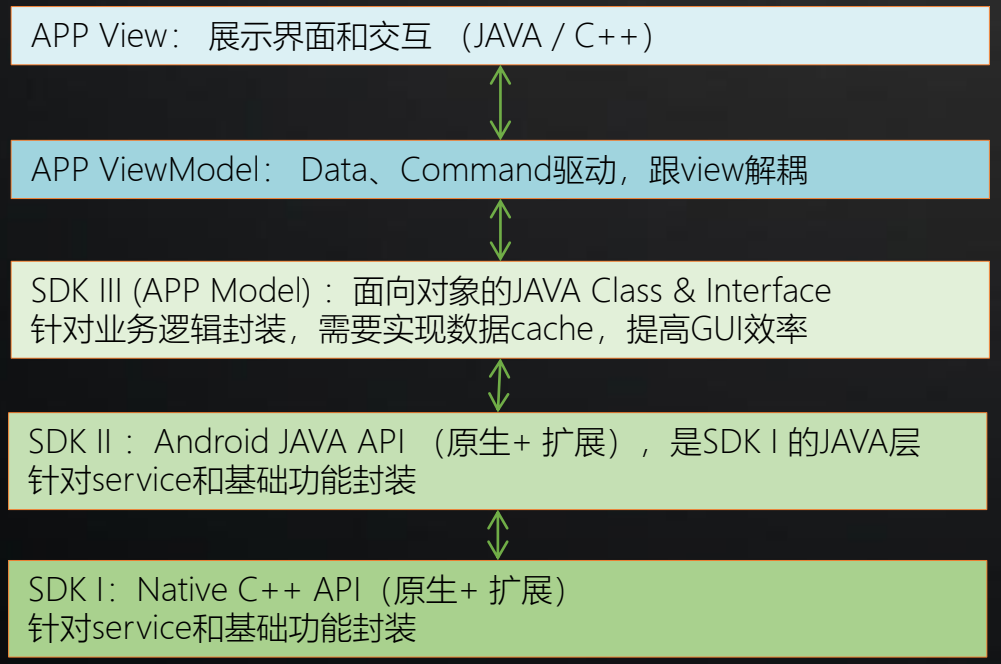
- *结构简单，容易理解和开发*
- *不同技能的程序员可以分工，负责不同的层次，天然适合大多数软件公司的组织架构*
- *每一层可以独立测试，其他层的接口通过模拟解决*

缺点

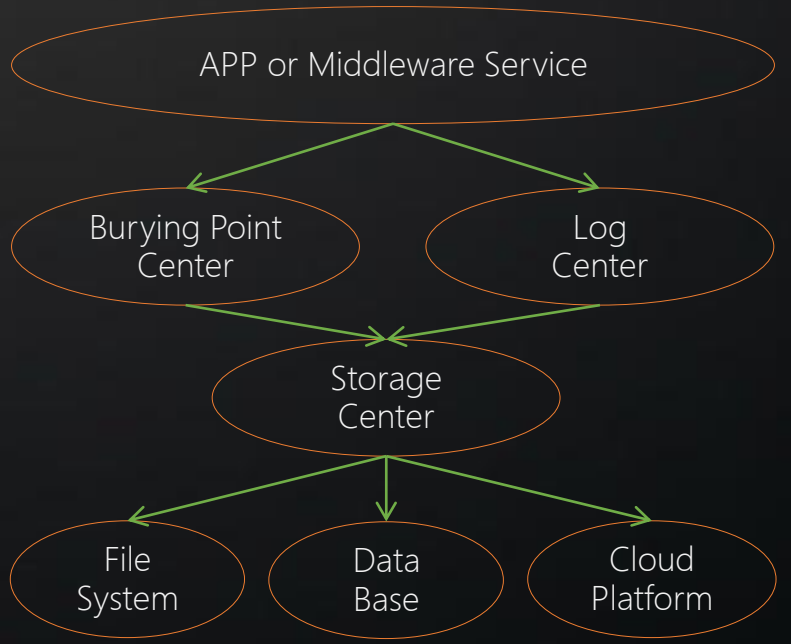
- *一旦环境变化，需要代码调整或增加功能时，通常比较麻烦和费时*
- *部署比较麻烦，即使只修改一个小地方，往往需要整个软件重新部署，不容易做持续发布*
- *软件升级时，可能需要整个服务暂停*
- *扩展性差。用户请求大量增加时，必须一次扩展每一层，由于每一层内部是耦合的，扩展会很困难*

01 分层架构

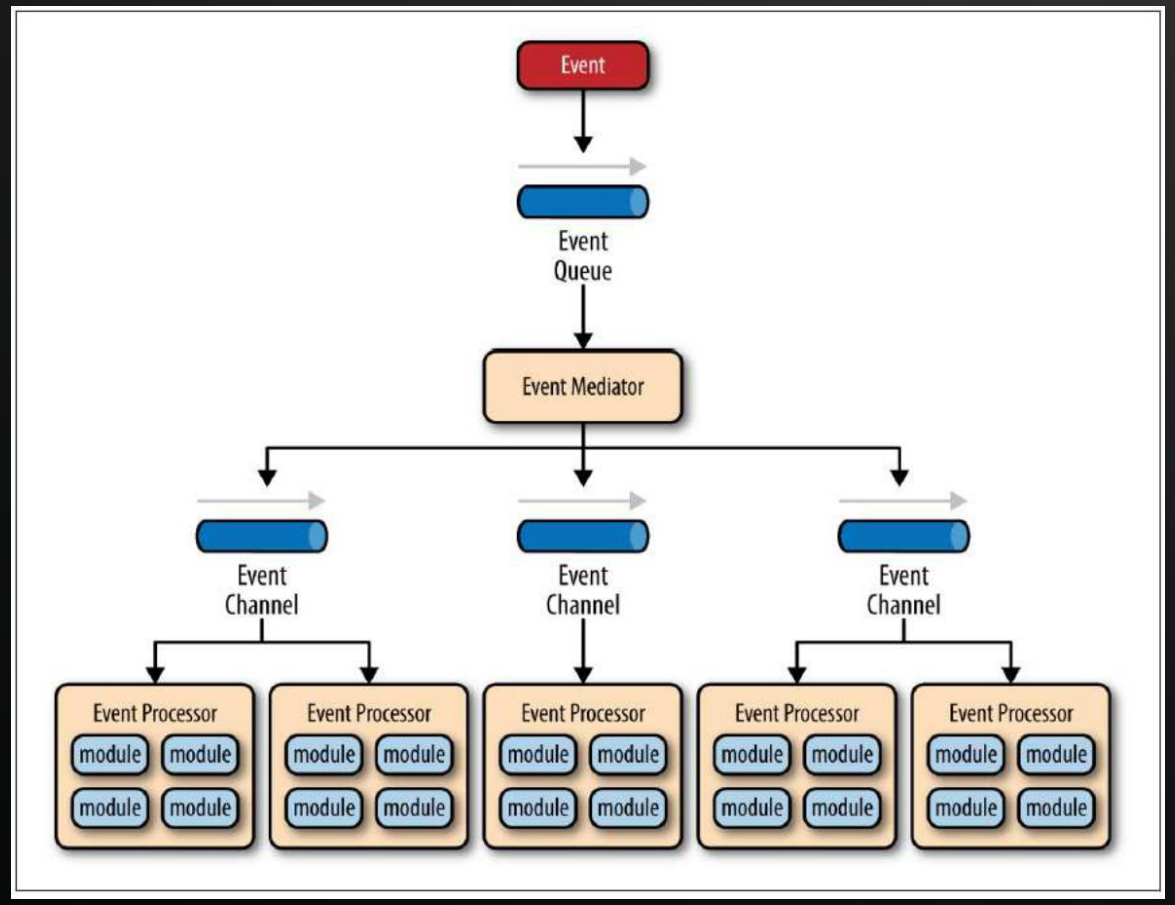
示例1:



示例2:



02 事件驱动架构



Event Queue: 接受事件，管理事件队列
Event Mediator: 分发事件到各处理单元
Event Channel: 分发器与处理器的统一通道
Event Processor: 事件处理单元，实现业务逻辑

关键点：

- 简单项目，合并Queue, Mediator, Channel
- 在Channel中，合并或踢出Event，提高效率
- 各Processor间不要横向流转Event
- Event可以带data

优点

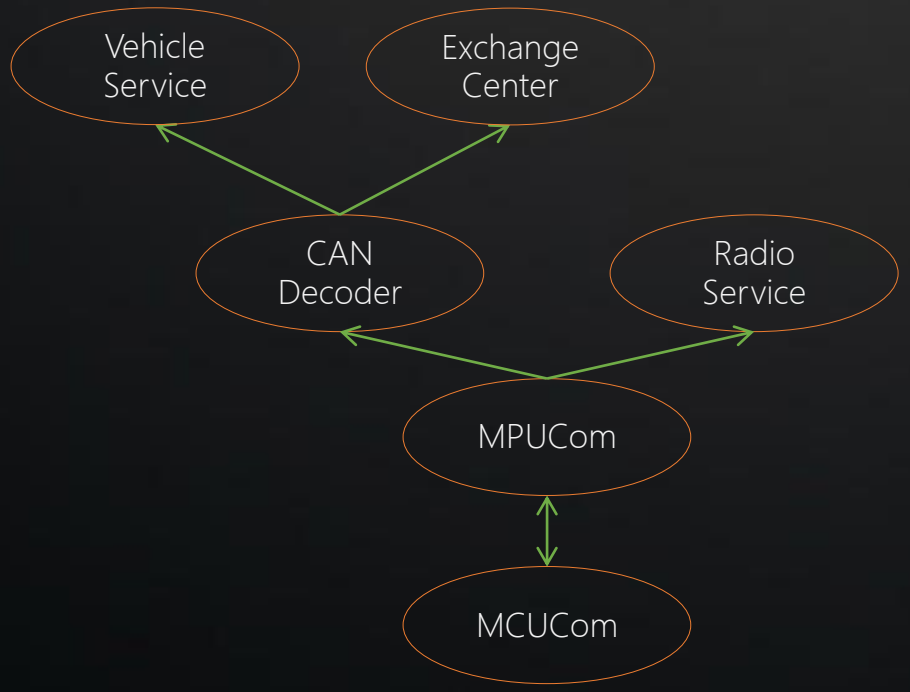
- *分布式的异步架构，事件处理器之间高度解耦，软件的扩展性好*
- *适用性广，各种类型的项目都可以用*
- *性能较好，因为事件的异步本质，软件不易产生堵塞*
- *事件处理器可以独立地加载和卸载，容易部署*

缺点

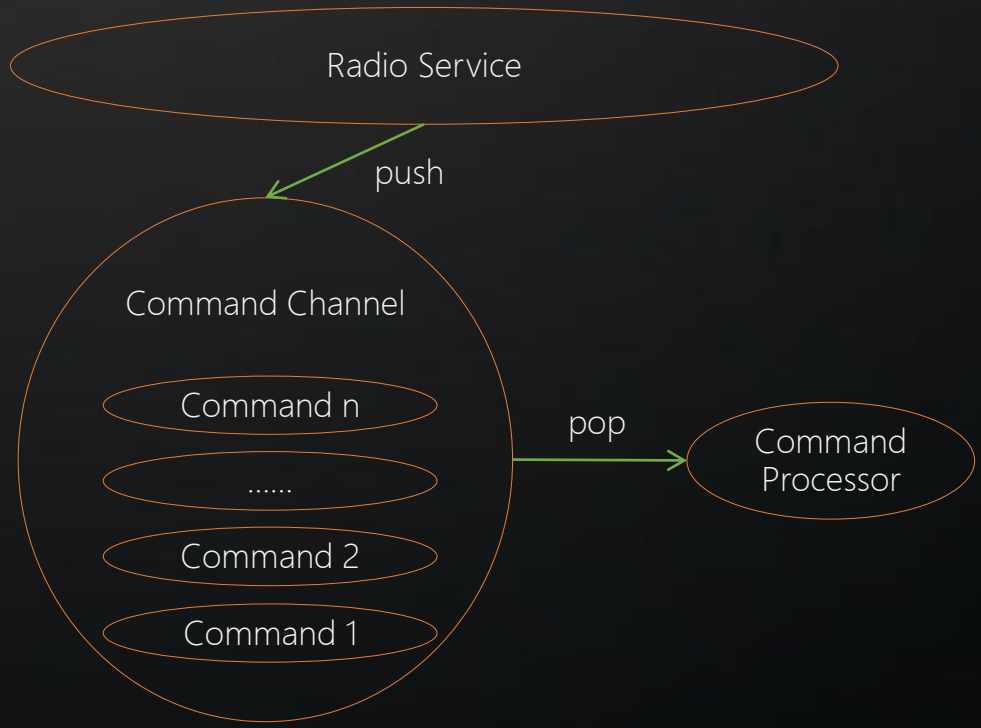
- *涉及异步编程（要考虑远程通信、失去响应等情况），开发相对复杂*
- *难以支持原子性操作，因为事件通过会涉及多个处理器，很难回滚*
- *分布式和异步特性导致这个架构较难测试*

02 事件驱动架构

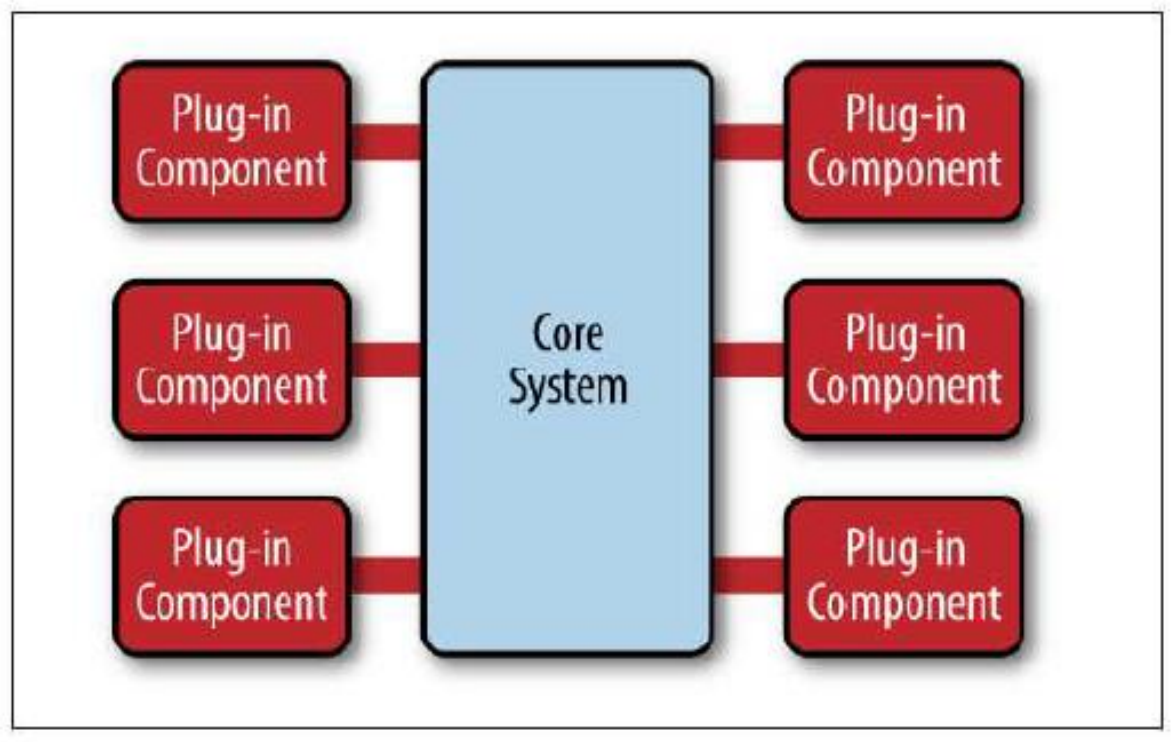
示例1:



示例2:



03 微核（插件）架构



Core: 系统运行框架和最小功能单元
Plug-in: 可插拔功能单元

关键点：

- Plug-in间独立，没有依赖
- Core建立怎样的Plug-in挂入机制
- Core如何在Plug-in间路由数据

优点

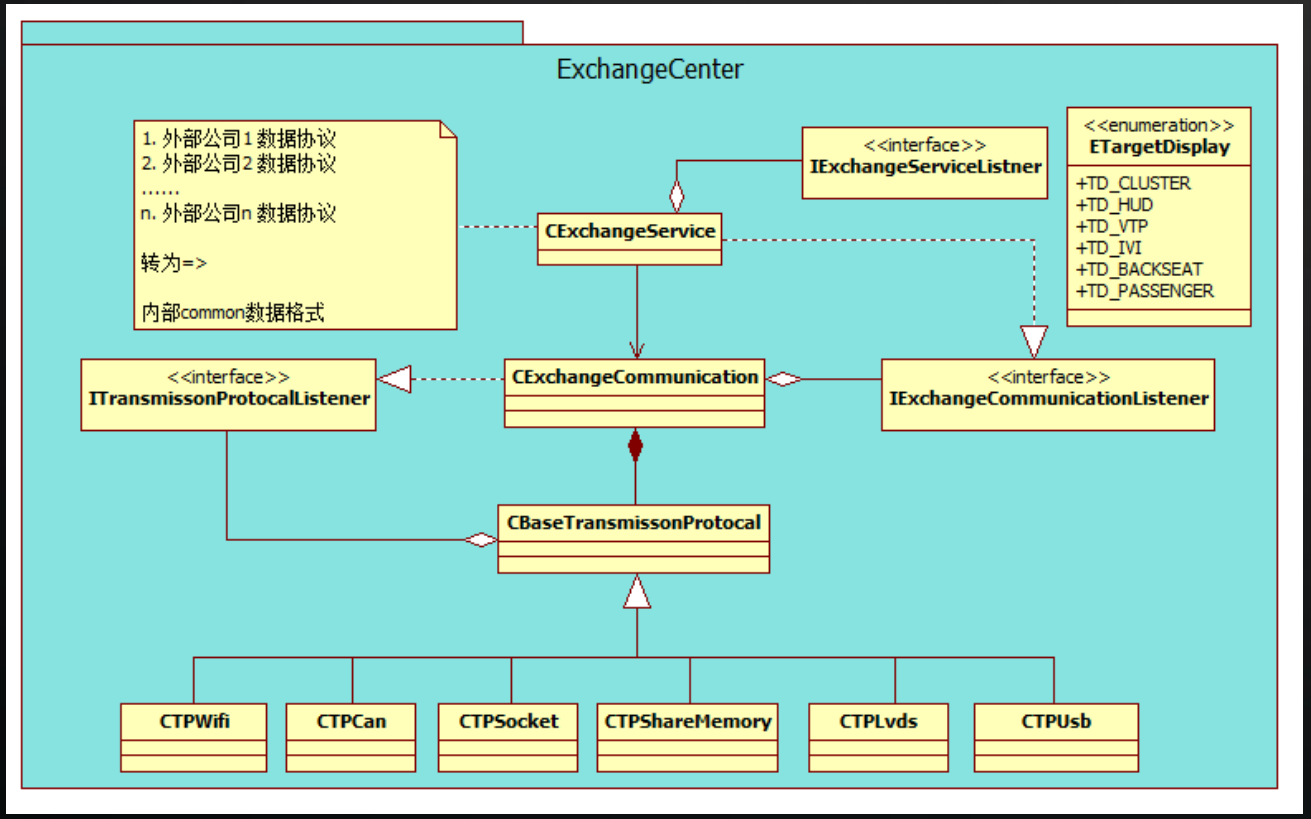
- *良好的功能延伸性（extensibility），需要什么功能，开发一个插件即可*
- *功能之间是隔离的，插件可以独立的加载和卸载，使得它比较容易部署，*
- *可定制性高，适应不同的开发需要*
- *可以渐进式地开发，逐步增加功能*

缺点

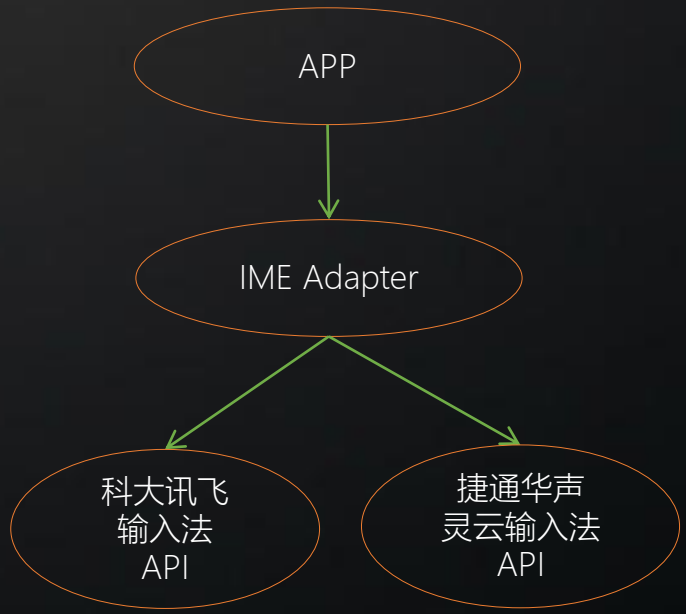
- *扩展性（scalability）差，内核通常是一个独立单元，不容易做成分布式*
- *开发难度相对较高，因为涉及到插件与内核的通信，以及内部的插件登记机制*

03 微核（插件）架构

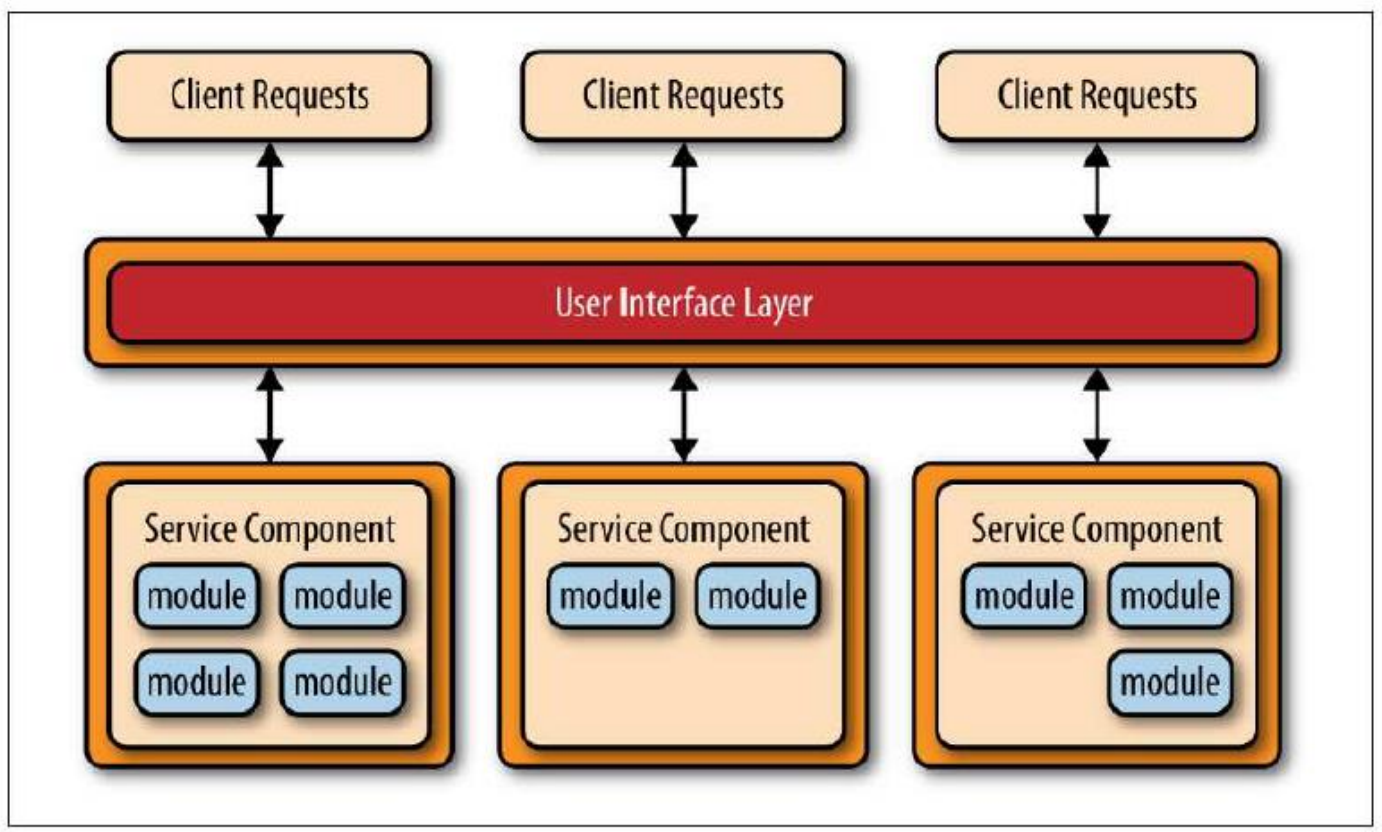
示例1:



示例2:



04 微服务架构



Client: 服务的使用者
Service Component: 服务部署单元
User Interface: Client跟Service之间的桥梁

关键点：

- Interface可以是API, 也可以是数据协议
- Interface的稳定性，向前兼容性
- IPC/RPC通信

优点

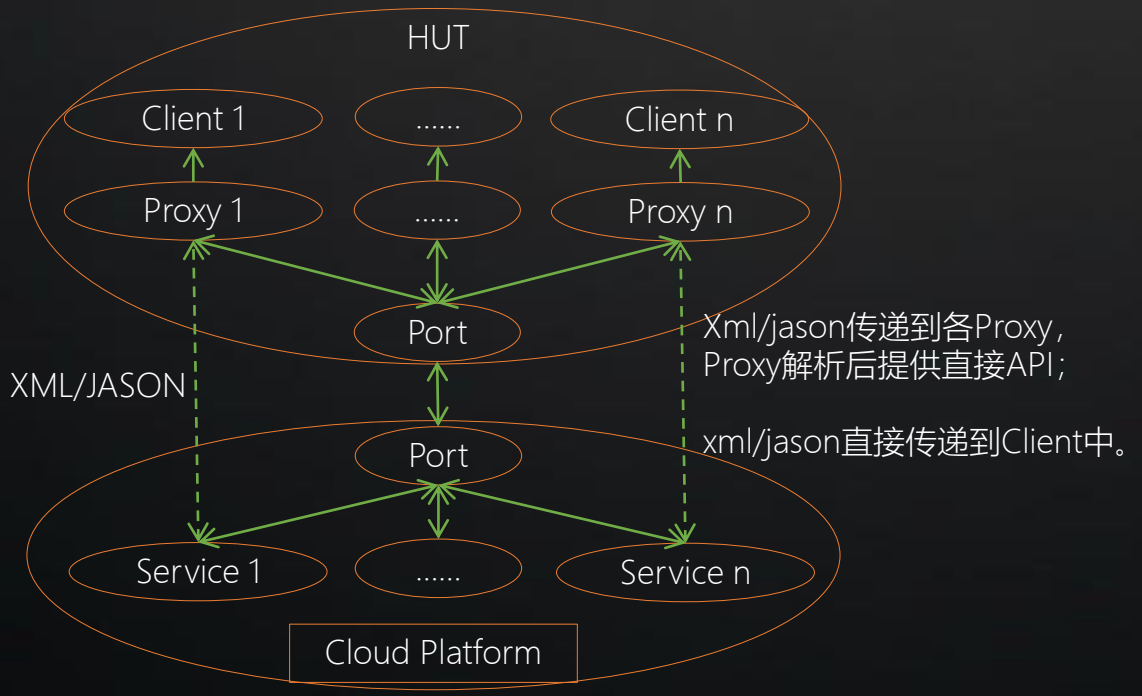
- *扩展性好，各个服务之间低耦合*
- *容易部署，软件从单一可部署单元，被拆成了多个服务，每个服务都是可部署单元*
- *容易开发，每个组件都可以进行持续集成式的开发，可以做到实时部署，不间断地升级*
- *易于测试，可以单独测试每一个服务*

缺点

- *由于强调互相独立和低耦合，服务可能会拆分得很细。这导致系统依赖大量的微服务，变得很凌乱和笨重，性能也会不佳。*
- *一旦服务之间需要通信（即一个服务要用到另一个服务），整个架构就会变得复杂。典型的例子就是一些通用的 Utility 类，一种解决方案是把它们拷贝到每一个服务中去，用冗余换取架构的简单性。*
- *分布式的本质使得这种架构很难实现原子性操作，交易回滚会比较困难。*

04 微服务架构

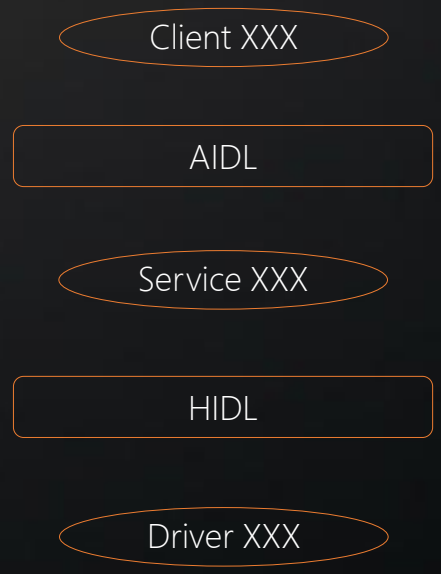
示例1:



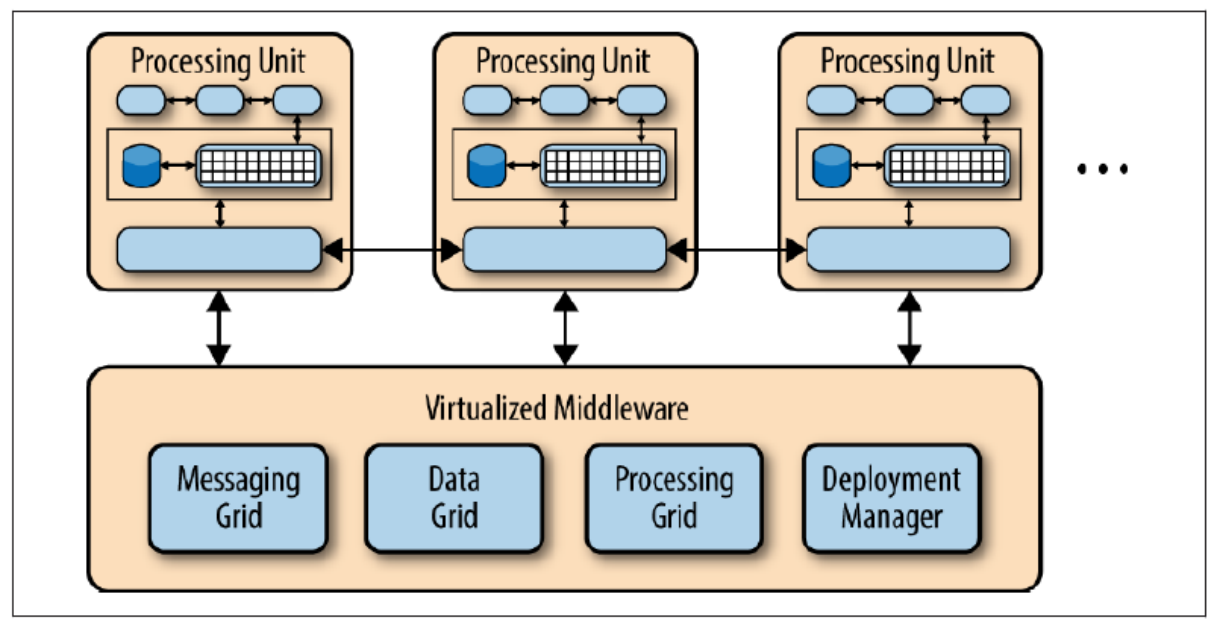
Xml/jason传递到各Proxy，
Proxy解析后提供直接API；

xml/jason直接传递到Client中。

示例2:



05 云架构



关键点：

- 没有中央数据库，数据复制在内存中
- 处理单元数量根据业务量而动态调整
- 内存数据的持久化
- 如何在各处理单元间同步数据

优点

- *高负载，高扩展性*
- *动态部署*

缺点

- *实现复杂，成本较高*
- *主要适合网站类应用，不合适大量数据吞吐的大型数据库应用*
- *较难测试*

Processing Unit: 处理单元，实现业务逻辑。
Virtualized Middleware: 虚拟中间件，负责通信，保持sessions，数据复制，分布式处理，处理单元的部署。
Messaging Grid: 消息中间件，管理用户请求和session，针对请求决定分配给哪一个处理单元。
Data Grid: 数据中间件，完成数据在处理单元间同步。
Processing Grid: 处理中间件，可选，针对一个请求需多个处理单元的情况。
Deployment Manager: 部署中间件，监控负载和响应时间，决定处理单元的启动和关闭。

THANK YOU