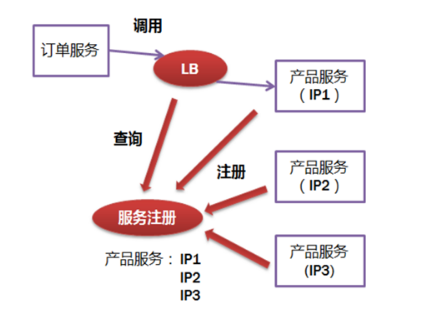
# 微服务概念

微服务具备的特性：每个微服务都可以运行在自己的进程里；一系列独立运行的微服务共同构建起了整个系统；每个服务为独立的业务开发，一个微服务一般完成某个特定的功能，比如订单管理，用户管理等；微服务之间通过一些轻量级的通信机制进行通信，例如通过REST API或者RPC的方式进行调用

微服务带来的挑战：运维要求高，分布式的复杂性增加，接口调整成本高，重复劳动

微服务的设计原则：单一职责，服务自治，轻量级通信，接口明确

* 简单的服务调用关系



# 常用微服务架构

Spring cloud：http

Motan：rpc 底层：nio（netty）



# Rpc原理

RPC（Remote Procedure Call）——[远程过程调用](https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8)，它是一种通过[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C)从远程计算机程序上请求服务。RPC是与语言无关的。

* 技术简介

RPC的核心就是能够让本地应用简单、高效地调用远程服务。一般从以下几方面考虑：

1. 远程服务代理：本地调用的方法(服务)实质是远程方法的本地代理，因此会需要一个远程服务代理对象。在Java中，远程服务代理对象可以使用JDK动态代理实现。
2. 通信模型：客户端与服务器之间的通信方式。Java中一般基于BIO或NIO。
3. 服务定位：通过IP与端口确定服务器后，再通过接口名及方法名、参数等确定具体的服务。
4. 参数序列化：将接口名、方法名、方法参数类型等要进行网络传输的数据，需要转换成二进制传输，这里可能需要不同的序列化技术方案。如:Protostuff，java原生序列化等。

* Java简单实现需要技术

1. JDK动态代理
2. Socket通信（BIO方式）
3. 反射、注解
4. Java序列化（注意所有需要序列化对象必须实现Serializable接口）

* 简单的rpc示例

代码演示

# Rpc和http区别

<https://www.cnblogs.com/leenuxqt/p/3202527.html>

<https://blog.csdn.net/wangyunpeng0319/article/details/78651998>

<https://www.zhihu.com/question/41609070>

# Motan简单示例

* Github地址：<https://github.com/weibocom/motan>
* 中文wiki：<https://github.com/weibocom/motan/wiki/zh_overview>

# Motan基本配置

配置清单：<https://github.com/weibocom/motan/wiki/zh_configuration>

## 协议常见配置

1. Loadbalance 负载均衡策略
2. haStrategy 高可用 容错策略

## 服务端常见配置

1. Version 版本

## 客户端常见配置

1. throwException 是否抛出异常
2. requestTimeout 请求超时时间
3. retries重试次数
4. check 检测服务是否存在
5. 指定特别方法的配置

# 扩展点

* 编写一个实现spi扩展点接口
* 指定service的filter
* Filter 发送/接收请求过程中增加切面逻辑，默认提供日志统计等功能
* HAStrategy 扩展可用性策略，默认提供快速失败等策略
* LoadBalance 扩展负载均衡策略，默认提供轮询等策略
* Serialization 扩展序列化方式，默认使用Hession序列化
* Protocol 扩展通讯协议，默认使用Motan自定义协议
* Registry 扩展服务发现机制，默认支持Zookeeper、Consul等服务发现机制
* Transport 扩展通讯框架，默认使用Netty框架

# 监控