浅析Dubbo服务

> Dubbo是阿里开源的一个分布式服务框架，致力于提供高性能和透明化的RPC远程调用方案，以及SOA服务治理方案。 Dubbo是阿里巴巴SOA服务化治理方案的核心框架，每天为2,000+个服务提供3,000,000,000+次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点。Dubbo是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，以及SOA服务治理方案。

转变的历程

1. 单一应用框架(ORM)

当网站流量小时，只需一个应用，可以将所有功能都部署在一起，这样可以减少部署的节点和成本。

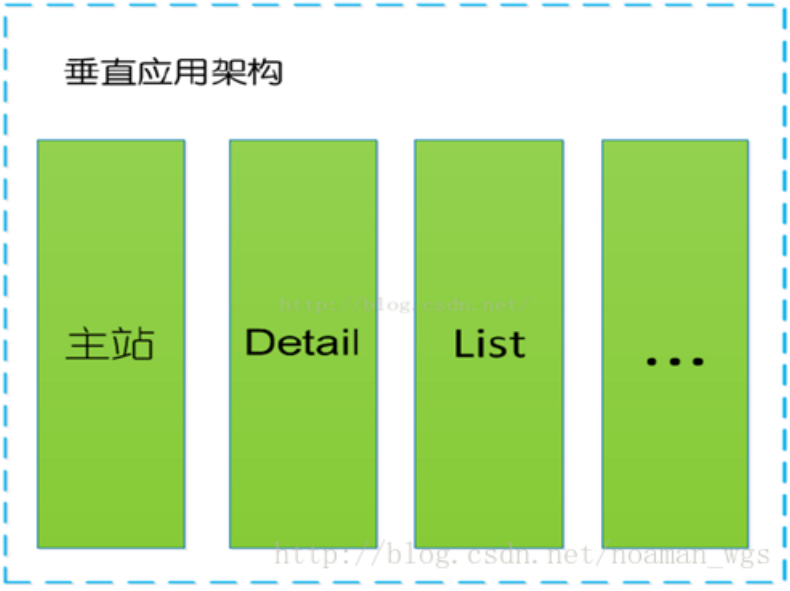
缺点：单一的系统架构，使得在开发过程中，占用的资源越来越多，随着流量的增加，将会越来越难维护。

Dubbo采用微内核+插件话体系，设计十分优雅，扩展性强。

1. 垂直应用架构(MVC)

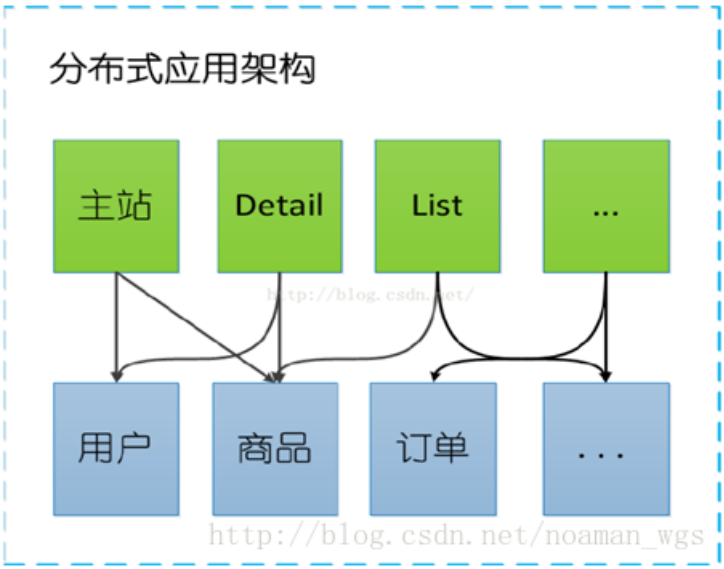
垂直应用架构解决了单一应用架构所面临的扩容问题，容量能够分散到各个子系统当中，且系统的体积可控，一定程度上降低了开发人员协同以及维护的成本，提升了开发效率。

缺点：在垂直架构中相同逻辑代码需要不断复制，不能复用。



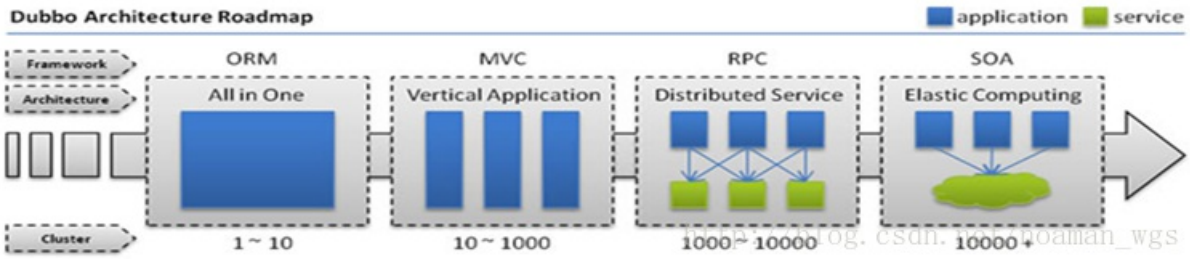
1. 分布式应用架构(RPC)

当垂直应用越来越多的时候，应用之间的交互是不可避免的，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心。



1. 流动计算架构(SOA)

随着服务化的进一步服务，服务越来越多，服务之间的调用和依赖关系也越来越复杂，诞生了面向服务的架构体系(SOA),也因此衍生了一系列相应的结束，如对服务提供，链接处理，通信协议，序列化方式，服务发现，服务路由，日志输出等行为进行封装的服务框架



- 单一应用架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。 此时，用于简化增删改查工作量的 数据访问框架(ORM) 是关键。

- 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。 此时，用于加速前端页面开发的 Web框架(MVC) 是关键。

- 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。 此时，用于提高业务复用及整合的 分布式服务框架(RPC) 是关键。

- 流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。 此时，用于提高机器利用率的 资源调度和治理中心(SOA) 是关键。

RPC(Remote Procedure Call Protocol)(远程过程调用协议)

当我们有两台服务器A、B,分别部署了不同的应用a,b，当服务器A 想调用服务器B上面提供的方法的时候，这个是不能够直接调用的，这个时候就需要通过网络来表达调用的语义和调用的数据，这个时候远程调用服务的概念就产生了。

> RPC是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC协议假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。RPC使得开发包括网络分布式多程序在内的应用程序更加容易。 RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是一个服务

> 器。首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。

Dubbo的概念

Dubbo概念

- 一款分布式服务框架
- 高性能和透明化的RPC远程服务调用方案
- SOA服务治理方案

> Dubbo是一个分布式服务框架，以及SOA治理方案。其功能主要包括：高性能NIO通讯及多协议集成，服务动态寻址与路由，软负载均衡与容错，依赖分析与降级等。

Dubbo适用于哪些场景

当网站变大后，不可避免的需要拆分应用进行服务化，以提高开发效率，调优性能，节省关键竞争资源等。

当服务越来越多时，服务的URL地址信息就会爆炸式增长，配置管理变得非常困难，F5硬件负载均衡器的单点压力也越来越大。

当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。

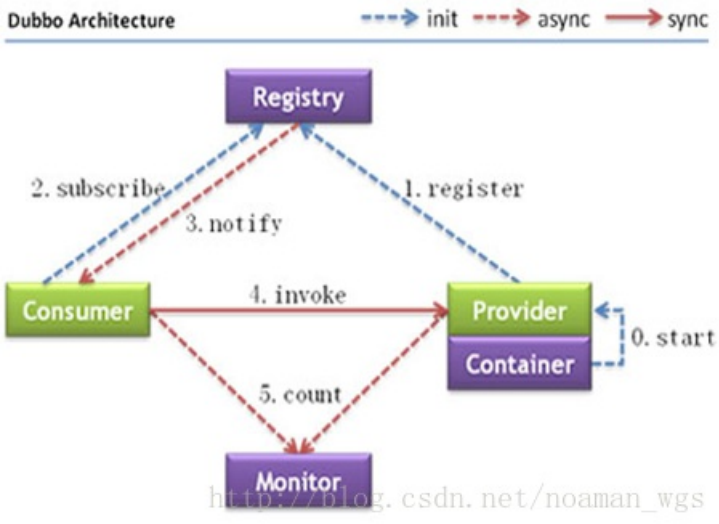
接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？等等……

在遇到这些问题时，都可以用Dubbo来解决。

Dubbo的设计思路

该框架具有极高的扩展性，采用微核+插件体系，并且文档齐全，很方便二次开发，适应性极强。

Dubbo架构



Provider:暴露服务的服务提供方 Consumer:调用远程服务的服务消费方 Registry:服务注册与发现注册中心 Monitor:统计服务的调用次数和调用时间的监控中心

调用流程： 0.服务容器负责启动，加载，运行服务提供者。 1.服务提供者在启动时，向注册中心注册自己提供的服务。 2.服务消费者在启动时，向注册中心订阅自己所需的服务。 3.注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。 4.服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。 5.服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心

Dubbo注册中心

对于服务提供方，它需要发布服务。 对于消费方，它需要获取提供方提供的服务。 当然，也可以即是提供方也是消费方。 通过服务统一管理起来，可以有效地优化内部应用对服务发布/使用的管理。服务注册中心可以通过特定协议来完成对外的统一。

Dubbo提供的注册中心有如下几种类型可供选择： * Multicast注册中心 * Zookeeper注册中心 * Redis注册中心 * Simple注册中心

Dubbo优点

优点： 1. 透明化的远程方法调用 - 像调用本地方法一样调用远程方法；只需简单配置，没有任何API侵入。 2. 软负载均衡及容错机制 - 可在内网替代nginx lvs等硬件负载均衡器。 3. 服务注册中心自动注册 & 配置管理 -不需要写死服务提供者地址，注册中心基于接口名自动查询提供者ip。 使用类似zookeeper等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入zookeeper集群。 4. 服务接口监控与治理 -Dubbo-admin与Dubbo-monitor提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行多版本，多协议，多注册中心管理。

Dubbo序列化协议

Hessian协议

Hessian是一个轻量级的RPC服务，它是基于Binary-RPC协议实现的，他会序列化反序列化你的实例，它的传输协议时Http协议，在dubbo中，主要应用到了它的序列化的协议。

hessian适合发送二进制数据，通过hessian序列化后的包，包的体积会比java自带的小一些。

和java自带的序列化做一个对比

```
public class Main {

    private static long time1;
    private static long time2;

    public static byte[] hessianSerialize(Object obj) throws IOException {
        long nowTime = System.currentTimeMillis();
        if (obj == null)
            throw new NullPointerException();

        ByteArrayOutputStream os = new ByteArrayOutputStream();
        HessianOutput ho = new HessianOutput(os);
        ho.writeObject(obj);
        time1 = System.currentTimeMillis() - nowTime;
        return os.toByteArray();
    }

    public Object hessianDeserialize(byte[] by) throws IOException {
        if (by == null)
            throw new NullPointerException();

        ByteArrayInputStream is = new ByteArrayInputStream(by);
        HessianInput hi = new HessianInput(is);
        return hi.readObject();
    }

    public static byte[] javaSerialize(Object obj) throws Exception {
        long noewTime = System.currentTimeMillis();
        if (obj == null)
            throw new NullPointerException();

        ByteArrayOutputStream os = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(os);
        out.writeObject(obj);
        time2 = System.currentTimeMillis() - noewTime;
        return os.toByteArray();
    }

    public Object javaDeserialize(byte[] by) throws Exception {

        if (by == null)
            throw new NullPointerException();

        ByteArrayInputStream is = new ByteArrayInputStream(by);
        ObjectInputStream in = new ObjectInputStream(is);
        return in.readObject();
    }

    public static void main(String[] args) throws Exception {

        People people = new People(19, "222333222233332222", "关玮琳", "健康");
        People people2 = new People(21, "555333222233332222", "张三", "不健康");
        byte[] hessianByte = hessianSerialize(people);
        byte[] javaByte = javaSerialize(people2);

        System.out.println("hessianByte 序列化后的长度    " + hessianByte.length);
        System.out.println("javaSerialize 序列化后的长度    " + javaByte.length);

        System.out.println("hessianByte 序列化时常   " + time1);
        System.out.println("javaSerialize 序列化时常   " + time2);

    }

}
```

结果：

```
hessianByte 序列化后的长度    86
javaSerialize 序列化后的长度    132
hessianByte 序列化时常   51
javaSerialize 序列化时常   16
```

Hessian语法

```
        #starting production
top     ::=value
```

```
        #分割成64k每chunk的8-bit二进制数据
binary  ::= 'b' b1 b0 <binary-data> binary  #不是最后一个chunk
        ::= 'B' b1 b0 <binary-data>         #最后一个chunk
        ::= [x20-x2f] <binary-data>         #长度范围为 0-15

        #boolean true/false
boolean ::= 'T'
        ::= 'F'

        #对象的定义(compact map)
class-def ::= 'O' type int string*

        #time in UTC encoded as 64-bit long milliseconds since epoch
date    ::= 'd' b7 b6 b5 b4 b3 b2 b1 b0

        #64-bit IEEE double
double  ::= 'D' b7 b6 b5 b4 b3 b2 b1 b0
        ::= x67                         #0.0
        ::= x68                         #1.0
        ::= x69 b0                      #byte表示的double(-128.0 to 127.0)
        ::= x6a b1 b0                   #short表示的double
        ::= x6b b3 b2 b1 b0             #32-bit float表示的double

        #32-bit 有符号整型
int     ::= 'I' b3 b2 b1 b0
        ::= [x80-xbf]                   #-x10 to x3f
        ::= [xc0-xcf] b0                #-x800 to x7ff
        ::= [xd0-xd7] b1 b0             #-x40000 to x3ffff

        # list/vector length
length  ::= 'l' b3 b2 b1 b0
        ::= x6e int

        # list/vector
list    ::= 'V' type? length? value* 'z'
        ::= 'v' int int value*          #第一个int表示类型引用, 第二个int表示长度

        #64-bit有符号long
long    ::= 'L' b7 b6 b5 b4 b3 b2 b1 0
        ::= [xd8-xef]                   #-x08 to x0f
        ::= [xf0-xff] b0                #-x800 to x7ff
        ::= [x38-x3f] b1 b0             #-x40000 to x3ffff
        ::= x77 b3 b2 b1 b0             #32-bit 整型表示的long

        #map/object
map     ::= 'M' type? (value value)* 'z'    #key, value map pairs

        # null value
null    ::= 'N'

        #对象实例
object  ::= 'o' int value*

        #值引用
ref     ::= 'R' b3 b2 b1 b0 # 对流中第n个map/list/object的引用

        ::= x4a b0          # 对map/list/object的引用, 范围为1-255th
        ::= x4b b1 b0       # 对map/list/object 的引用, 范围为1-65535th

        #UTF-8 编码的字符串, 分割成64k大小的chunks
string  ::= 's' b1 b0 <utf8-data> string    #非末尾chunk
        ::= 'S' b1 b0 <utf8-data>           #长度范围为(0-65535)的字符串
        ::=[x00-x1f] <utf8-data>            #长度范围为(0-31) 的字符串

        #map/list 的类型(针对面向对象语言)
type    ::= 't' b1 b0 <type-string>     #类型名称
        ::= x75 int                     #类型引用值(用整数表示)

        #main production
value   ::=null
        ::= binary
        ::= boolean
        ::= date
        ::= double
        ::= int
        ::= list
        ::= long
        ::= map
        ::= class-def value
        ::= ref
        ::= string
```

可序列化的类型

1. 原始二进制数据
2. boolean
3. 64-bit date
4. 64-bit double
5. 32-bit int
6. 64-bit long
7. null
8. UTF8编码的string

另外包括3种递归类型：

1. list for lists and arrays
2. map for maps and dictionaries
3. object for objects

最后，它还包含一个特殊的类型：

1. ref 用来表示对共享对象的引用.

Hessian 2.0有3个内部的引用表:

1. 一个object/list 引用表.
2. 一个类型定义(class definition)引用表.
3. 一个type(class name)引用表.

参考文献

什么是Hessian协议呢？

hessian学习基础篇——序列化和反序列化

Dubbo原理解析

Dubbo入门基础与实例讲解