第一个阶段:

    当网站的访问量比较小的时候,我们只需要简单的一个应用,把需要的功能都部署到一起,此时用于简化增删改查的[**数据库**](http://lib.csdn.net/base/mysql)访问框架(ORM)是当时的主角,它的好处是减少了部署节点和成本.这个阶段采用的架构是单一应用架构.

     第二个阶段:

  当你的访问量逐渐开始变大的时候,我们会将应用拆分互不相干的几个应用来提升效率,这时候加速前段页面开发的Web框架(MVC)成关键了.这个阶段采用的架构是垂直应用架构.

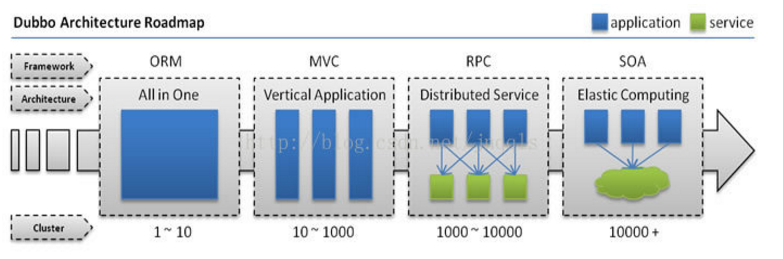
    第三个阶段:

  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。这个阶段采用的架构是分布式服务架构.

    第四个阶段:

   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。这个阶段用于提高机器利用率的 资源调度和治理中心(SOA) 站到舞台上了.这个阶段的架构是流动计算架构.

         以上的四个阶段可以用官方的Dubbo架构路线图来概括,如下:



 (1)当服务越来越多时，服务URL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。此时需要一个服务注册中心，动态的注册和发现服务，使服务的位置透明。并通过在消费方获取服务提供方地址列表，实现软负载均衡和Failover，降低对F5硬件负载均衡器的依赖，也能减少部分成本。

        (2)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。这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。

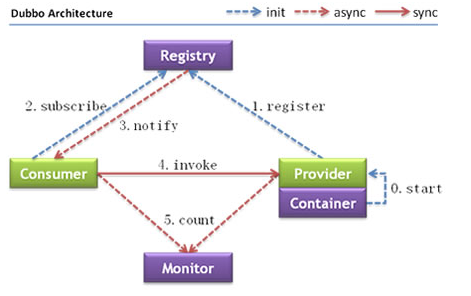
       (3)接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？

       为了解决这些问题，

       第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。

       其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阀值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

**Dubbo架构**



**节点角色说明：**

* **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:** 服务注册与发现的注册中心。
* **Monitor:** 统计服务的调用次调和调用时间的监控中心。
* **Container:** 服务运行容器。

**调用关系说明：**

* 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。

**(1) 连通性：**

* 注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小
* 监控中心负责统计各服务调用次数，调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示
* 服务提供者向注册中心注册其提供的服务，并汇报调用时间到监控中心，此时间不包含网络开销
* 服务消费者向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，同时汇报调用时间到监控中心，此时间包含网络开销
* 注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外
* 注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者
* 注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表
* 注册中心和监控中心都是可选的，服务消费者可以直连服务提供者

**(2) 健状性：**

* 监控中心宕掉不影响使用，只是丢失部分采样数据
* 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
* 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
* 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
* 服务提供者无状态，任意一台宕掉后，不影响使用
* 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

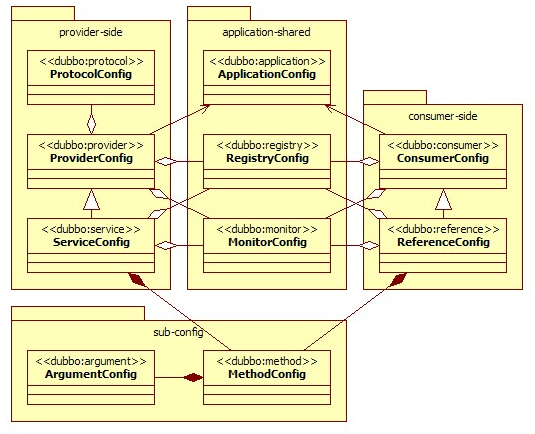
**(3) 伸缩性：**

* 注册中心为对等集群，可动态增加机器部署实例，所有客户端将自动发现新的注册中心
* 服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者

**(4) 升级性：**

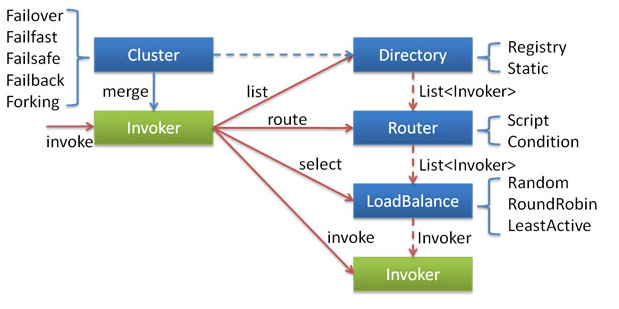
* 当服务集群规模进一步扩大，带动IT治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力：

**XML配置**



* [<dubbo:service/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aservice%252F%253E) 服务配置，用于暴露一个服务，定义服务的元信息，一个服务可以用多个协议暴露，一个服务也可以注册到多个注册中心。
* [<dubbo:reference/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Areference%252F%253E) 引用配置，用于创建一个远程服务代理，一个引用可以指向多个注册中心。
* [<dubbo:protocol/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aprotocol%252F%253E) 协议配置，用于配置提供服务的协议信息，协议由提供方指定，消费方被动接受。
* [<dubbo:application/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aapplication%252F%253E) 应用配置，用于配置当前应用信息，不管该应用是提供者还是消费者。
* [<dubbo:module/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Amodule%252F%253E) 模块配置，用于配置当前模块信息，可选。
* [<dubbo:registry/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aregistry%252F%253E) 注册中心配置，用于配置连接注册中心相关信息。
* [<dubbo:monitor/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Amonitor%252F%253E) 监控中心配置，用于配置连接监控中心相关信息，可选。
* [<dubbo:provider/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aprovider%252F%253E) 提供方的缺省值，当ProtocolConfig和ServiceConfig某属性没有配置时，采用此缺省值，可选。
* [<dubbo:consumer/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aconsumer%252F%253E) 消费方缺省配置，当ReferenceConfig某属性没有配置时，采用此缺省值，可选。
* [<dubbo:method/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Amethod%252F%253E) 方法配置，用于ServiceConfig和ReferenceConfig指定方法级的配置信息。
* [<dubbo:argument/>](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%253Cdubbo%253Aargument%252F%253E) 用于指定方法参数配置。

**集群容错**



各节点关系：

* 这里的Invoker是Provider的一个可调用Service的抽象，Invoker封装了Provider地址及Service接口信息。
* Directory代表多个Invoker，可以把它看成List<Invoker>，但与List不同的是，它的值可能是动态变化的，比如注册中心推送变更。
* Cluster将Directory中的多个Invoker伪装成一个Invoker，对上层透明，伪装过程包含了容错逻辑，调用失败后，重试另一个。
* Router负责从多个Invoker中按路由规则选出子集，比如读写分离，应用隔离等。
* LoadBalance负责从多个Invoker中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选。

##### 集群容错模式：

###### Failover Cluster

* 失败自动切换，当出现失败，重试其它服务器。(缺省)
* 通常用于读操作，但重试会带来更长延迟。
* 可通过retries="2"来设置重试次数(不含第一次)。

###### Failfast Cluster

* 快速失败，只发起一次调用，失败立即报错。
* 通常用于非幂等性的写操作，比如新增记录。

###### Failsafe Cluster

* 失败安全，出现异常时，直接忽略。
* 通常用于写入审计日志等操作。

###### Failback Cluster

* 失败自动恢复，后台记录失败请求，定时重发。
* 通常用于消息通知操作。

###### Forking Cluster

* 并行调用多个服务器，只要一个成功即返回。
* 通常用于实时性要求较高的读操作，但需要浪费更多服务资源。
* 可通过forks="2"来设置最大并行数。

###### Broadcast Cluster

* 广播调用所有提供者，逐个调用，任意一台报错则报错。(2.1.0开始支持)
* 通常用于通知所有提供者更新缓存或日志等本地资源信息。

### 负载均衡

###### Random LoadBalance

* 随机，按权重设置随机概率。
* 在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

###### RoundRobin LoadBalance

* 轮循，按公约后的权重设置轮循比率。
* 存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

###### LeastActive LoadBalance

* 最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。
* 使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

###### ConsistentHash LoadBalance

* 一致性Hash，相同参数的请求总是发到同一提供者。
* 当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。
* 算法参见：[http://en.wikipedia.org/wiki/Consistent\_hashing](javascript:if(confirm(%27http://en.wikipedia.org/wiki/Consistent_hashing%20%20\n\nThis%20file%20was%20not%20retrieved%20by%20Teleport%20Ultra,%20because%20it%20is%20addressed%20on%20a%20domain%20or%20path%20outside%20the%20boundaries%20set%20for%20its%20Starting%20Address.%20%20\n\nDo%20you%20want%20to%20open%20it%20from%20the%20server?%27))window.location=%27http://en.wikipedia.org/wiki/Consistent_hashing%27)。
* 缺省只对第一个参数Hash，如果要修改，请配置<dubbo:parameter key="hash.arguments" value="0,1" />
* 缺省用160份虚拟节点，如果要修改，请配置<dubbo:parameter key="hash.nodes" value="320" />

**配置API**

* com.alibaba.dubbo.config.ServiceConfig
* com.alibaba.dubbo.config.ReferenceConfig
* com.alibaba.dubbo.config.ProtocolConfig
* com.alibaba.dubbo.config.RegistryConfig
* com.alibaba.dubbo.config.MonitorConfig
* com.alibaba.dubbo.config.ApplicationConfig
* com.alibaba.dubbo.config.ModuleConfig
* com.alibaba.dubbo.config.ProviderConfig
* com.alibaba.dubbo.config.ConsumerConfig
* com.alibaba.dubbo.config.MethodConfig
* com.alibaba.dubbo.config.ArgumentConfig
  + 参见：[API配置](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-API%E9%85%8D%E7%BD%AE)

**注解API**

* com.alibaba.dubbo.config.annotation.Service
* com.alibaba.dubbo.config.annotation.Reference

使用Spring Boot 与Dubbo集成的时候，需要注意的是，不要使用Spring Boot提供的devtools热启动，因为devtools提供了两个ClassLoader，加载策略问题导致出现错误，无法启动。如果开发中需要热加载，那么使用Spring 提供的springloaded。