腾讯开源基于微服务的平台Tars：RPC开发、服务治理及一体化运营管理

原创 2017-04-11 钟科 InfoQ

作者丨钟科

编辑丨木环

Tars取名于电影“星际穿越”中的机器人，是腾讯内部使用将近十年的基于微服务的统一应用框架TAF（Total Application Framework），目前腾讯有160多个业务（如手机浏览器、应用宝、手机管家等）在1.6多万台服务器上使用Tars。

编者按

Tars主要是支持多语言的高性能RPC开发框架和配套一体化的服务治理平台，可以帮助企业或者用户以微服务的方式快速构建稳定可靠的分布式应用。目前开源的Tars支持C++、Java两种语言，同时

Node.js语言的Tars开源地址为：https://github.com/Tencent/Tars。

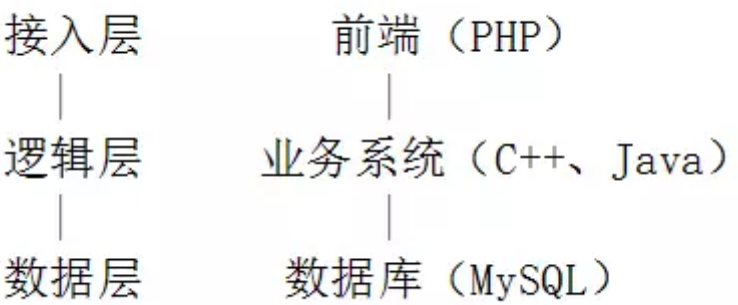
Tars取名于电影“星际穿越”中的机器人，是腾讯内部使用将近十年的基于微服务的统一应用框架TAF（Total Application Framework），目前腾讯有160多个业务（如手机浏览器、应用宝、手机管家等）在1.6多万台服务器上使用Tars。

近几年，业内已经有越来越多人关注并开始落地微服务，那么腾讯这款经历近十年探索的微服务治理平台是怎样的？它的开源会为社区带来哪些福音？为此，InfoQ采访了腾讯专家钟科。

1

业务发展，原有应用框架遭遇瓶颈

在业务种类不多、规模不大的早发展期，应用框架一般采用如下的技术方案：



在业务变化不大的情况下，这种方案一般能很好的工作几年、甚至十年。但是，彼时业务发展很快，业务需求、系统复杂度、业务规模、参与人数、代码腐化程度都在不断上升，各个业务团队慢慢陷于一种混乱状态，主要包括以下几点：

1. 业务逻辑集中，耦合性强，开发维护成本高；
2. 代码重复率高，基础或者公共组件能力不齐，交付效率和质量低；
3. 非功能需求不达标，性能、高可用性、可扩展性等方面难以适应业务海量访问发展趋势；
4. 运营数据缺失，管理能力薄弱，故障时分析和查找问题困难；

2

转变思想，微服务框架的研发历程

面对业务海量访，腾讯开始采用微服务架构的思想，设计和实现一个通用的统一应用框架平台，对业务提供涉及到开发、运维以及测试的一整套解决方案，让开发更简单，运维更高效。

Tars体系已经发展十年了，整个发展历程可以分三个阶段：

第一个阶段，框架产生的阶段。目的是做以微服务为基础的分布式架构。这个时期，开发侧面对的情况如下：服务模型多样化、业务协议不统一、基础组件或框架能力参差不齐等，导致开发大部分精力关注在协议的交互、网络通信的实现、系统的容错容灾、以及服务的部署、发布、扩容等，不能聚焦业务自身的逻辑，同时跨业务团队之间的系统互通效率底下；而运营侧处于的状况：运维工具各异，部署管理凌乱，监控能力薄弱，导致系统出现问题和故障后，基本靠用户投诉才能定位和发现。

为了解决面对的这些问题，同时更好地应对在互联网/移动互联网时代下业务海量访问的发展趋势，我们设计和实现了一个通用的应用框架，它具有以下能力：

1. IDL（接口定义语言）：统一不同语言的访问协议，二进制、可扩展、代码自动生成、支持多平台。
2. RPC：支持同步、异步、单向调用。
3. 高性能：框架提供10w/s以上的吞吐量。
4. 容错：任何一台服务down掉不影响业务访问。
5. 伸缩性：服务可以方便的平行扩展。
6. 管理：在web上就能对系统的服务进行部署、发布、上线、扩容、缩容等管理操作。

这便是Tars微服务架构的雏形，有了它之后，业务开发人员开始聚焦业务真正的逻辑，提高了开发效率，系统的运营管理开始简单化、规范化、流程化，提升了运维效率。

第二个阶段，框架优化重构的阶段。目的是解决框架性能和开发易用性的问题。

性能问题：以前程序大都是在2核或者4核的机器运行，而这个时期公司使用的机型在不断发展，慢慢出现了8核、16核、24核等的机器，同时系统也支持多队列网卡，由于框架是多线程模型的，同时为了保证代码良好的可读性和结构的简单性，在高性能这块，没有做太多精细的优化，线程共享资源之间粗粒度锁的使用和数据交互之间存在的内存拷贝等情况比较多，同时先前的网络收发只会被cpu0处理，导致一些对性能要求比较高的业务，在新的机型上使用框架时出现瓶颈问题，cpu和网络等资源利用不起来。

为了让框架在多核和多队列网卡的机器上充分利用资源和提供更高的性能，同时更好地为服务业务，我们对框架进行重构，原来只支持单线程的收发包模式，改造成多线程收发包，原来框架内影响线程并发运行的粗粒度锁和数据拷贝，改造成锁细粒度化或者无锁化，数据交互尽量做到零拷贝，最终框架的性能在8核机器上由19w/s提升到40w/s，cpu的最大利用率由62%提升到90%。

开发易用性问题：随着业务的发展，业务的服务模块越来越多，服务模块之间交互的关系越来越复杂，线上业务服务为了保证服务的稳定性，一般采用异步Callback方式进行服务交互，而异步Callback方式的代码逻辑分散，阅读和维护起来比较困难。为了让开发编写代码更加简单，我们提供了协程和future/promise两种编码方式。对原来的同步调用方式，通过协程技术，不改变原来的同步编码方式，

对业务透明，只需通过配置开关，可将其实际执行变同步为异步；对于原来的异步调用方式，通过future/promise技术，让编码方式像同步一样。

第三个阶段，框架弹性调度Docker化的阶段。目的是达到服务混合部署与自动化运营的效果。

服务混合部署问题：这个时期，使用tars的机器规模达到了1w多台，服务进程数量达到了6w个，而其中小服务比例很高，80%的进程CPU峰值小于1核，这中间还有75%的进程连0.1核都用不到。为了提高资源利用率，必须要将服务混合部署。但是进程依赖的运行环境差异大，不同进程的依赖经常存在冲突，导致混合部署难度很大。

即便没有环境冲突，各个服务进程部署到一台机器上之后，经常出现资源竞争，影响服务质量的问题。于是我们通过使用Docker方案将每个进程的运行环境和资源消耗隔离起来。服务部署、打包、发布等标准化，这样提高了部署效率，保证了交付质量，提升了资源利用率。

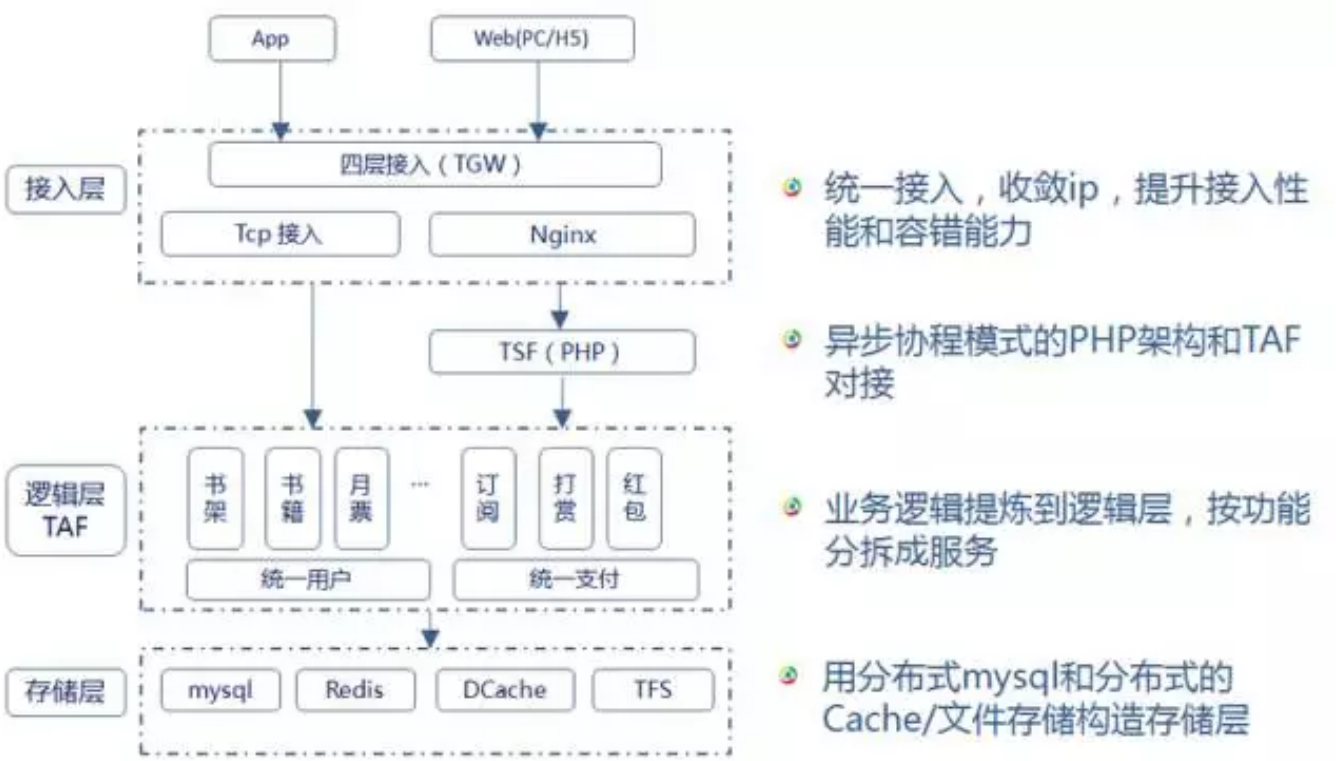
弹性调度问题：通过使用Docker技术，服务进程不再对机器有特殊要求，使得进程可以被部署到所有类型的机器上，给弹性调度巨大的操作空间。同时，通过Docker技术，一个服务下的所有容器可以被规整为相同的规格，使得服务的流量调度变得更加简单，简化了弹性调度的操作。

3

当前Tars微服务架构设计

Tars微服务架构的主要设计思想是以开发与运营之间DO分离为原则，采用分层思想，将各个层次之间相互解耦或者松耦合。

- 最底的协议层：设计思路是将业务网络通信的协议进行统一，以IDL(接口定义语言)的方式，开发支持多平台、可扩展、协议代码自动生成的统一协议。在开发过程中，开发人员只需要关注通讯的协议字段的内容，不需要关注其实现的细节。
- 中间的公共库、通讯框架、平台层：设计思路是让业务开发更加聚焦业务逻辑的本身。因此，从使用者的角度出发，封装了大量日常开发过程中经常使用的公共库代码和远程过程调用RPC，让开发使用更简单方便；从框架本身的角度出发，做到高稳定性、高可用性、高性能，这样才能让业务服务运营更加放心；从分布式平台的角度出发，解决服务运营过程中，遇到的容错、负载均衡、容量管理、就近接入、灰度发布等问题，让平台更加强大。
- 最上面的运营层，设计思路是让运维只需要关注日常的服务部署、发布、配置、监控、调度管理等操作，运维更高效。



分层的思想主要来自于多年海量分布式服务下的实践、总结和提炼。

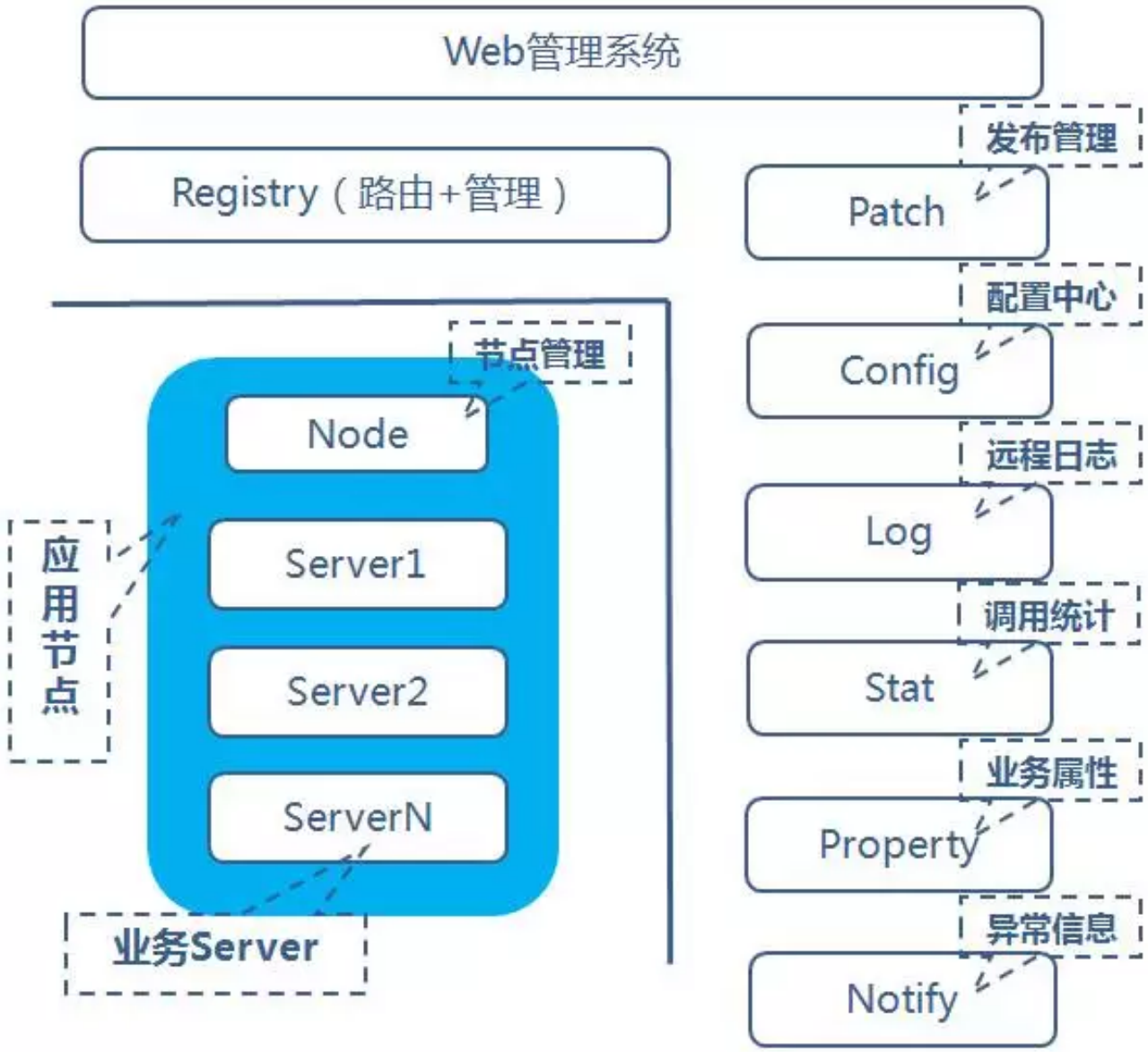
一方面，希望框架提供的功能和特性更全面、更完整，解决开发和运维面对的一些共性问题，比如涉及到分布式系统的高可用、容错容灾等，涉及运营操作的服务部署、发布、配置、监控、日志等，让业务开发更加聚焦其业务本身的逻辑开发，提高开发效率，同时让运维管理更加简单化、规范化、流程化，提升运维效率；

另一方面，希望框架支持的能力更灵活，业务可以使用整个系统中的某些部分或者整个功能和特性，这样业务在接入使用时，可以自主选择，减少改造成本，满足不同业务不同场景的需求。比如：终端与后端需要交互的业务，可以只使用统一的协议层进行通信；已有自身通信协议的业务，可以使用支持自定义协议的通信框架层等；对使用其他框架开发服务的业务，可以使用平台层提供的服务管理、监控等功能特性。

4

Tars技术概览

（一）架构拓扑图



整体架构的拓扑图主要分为两个个部分：服务节点与公共框架节点。

1. 服务节点：

服务节点可以认为是服务所实际运行的一个具体的操作系统实例，可以是物理主机或者虚拟主机、云主机。随着服务的种类扩展和规模扩大，服务节点可能成千上万甚至数以十万计。 每台服务节点上包括一个Node服务节点和N（大于等于0）个业务服务节点，Node服务节点对业务服务节点进行统一管理，提供启停、监控服务节点等功能，同时接收业务服务节点上报过来的心跳。

2. 公共框架节点：

除了服务节点以外的服务，其他服务节点均归为一类。

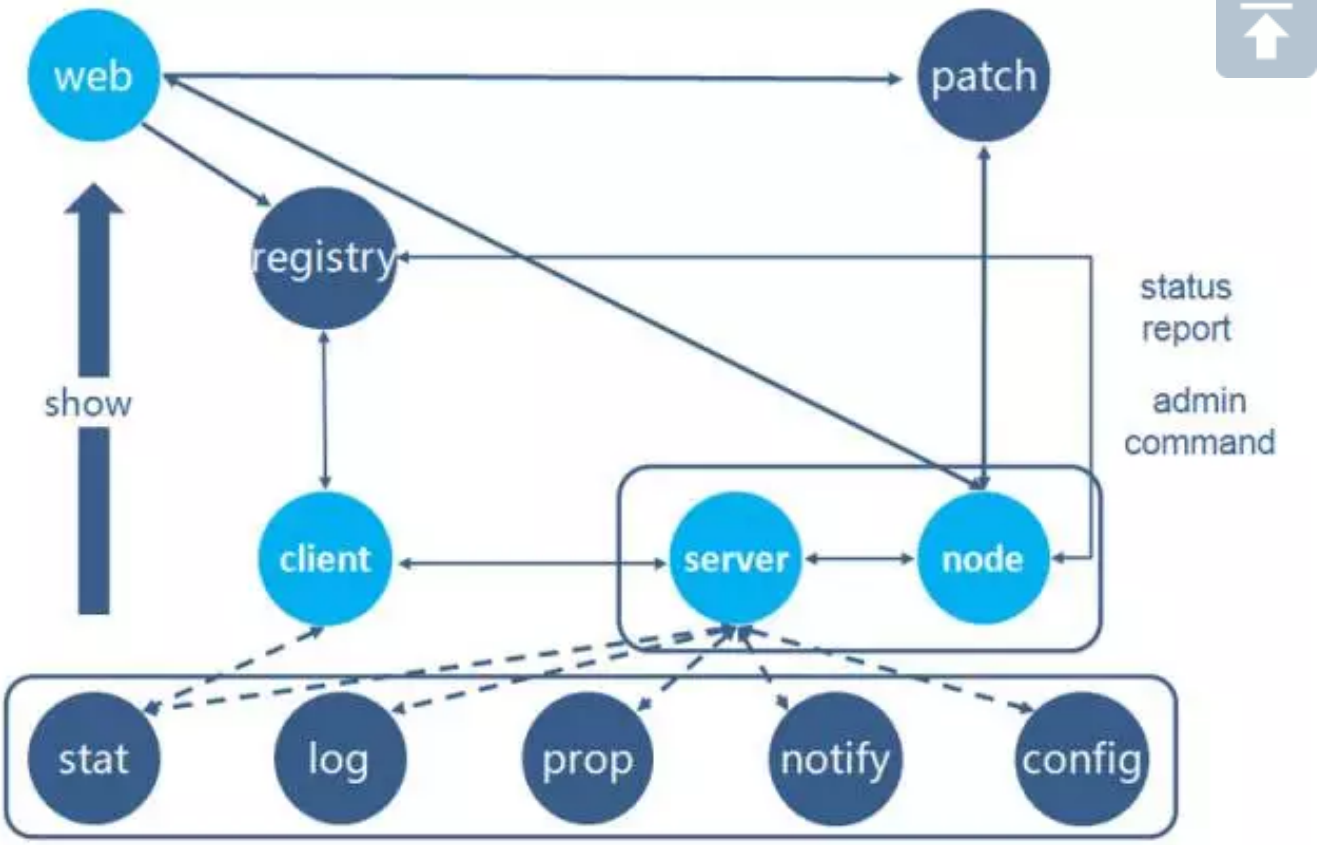
公共框架节点，数量不定，为了自身的容错容灾，一般也要求在在多个机房的多个服务器上进行部署，具体的节点数量，与服务节点的规模有关，比如：如果某些服务需要打较多的日志，就需要部署更多的日志服务节点。

主要包括如下几个部分：

- Web管理系统：在Web上可以看到服务运行的各种实时数据情况，以及对服务进行发布、启停、部署等操作；
- Registry（路由+管理服务）：提供服务节点的地址查询、发布、启停、管理等操作，以及对服务上报心跳的管理，通过它实现服务的注册与发现；
- Patch（发布管理）：提供服务的发布功能；
- Config（配置中心）：提供服务配置文件的统一管理功能；
- Log（远程日志）：提供服务打日志到远程的功能；
- Stat（调用统计）：统计业务服务上报的各种调用信息，比如总流量、平均耗时、超时率等，以便对服务出现异常时进行告警；
- Property（业务属性）：统计业务自定义上报的属性信息，比如内存使用大小、队列大小、cache命中率等，以便对服务出现异常时进行告警；
- Notify（异常信息）：统计业务上报的各种异常信息，比如服务状态变跟信息、访问db失败信息等，以便对服务出现异常时进行告警；

原则上要求全部的节点之间网络互通，至少每台机器的node能够与公共框架节点之间都是可以连通的。

（二）服务交互流程图



系统主要交互流程如下:

- **服务发布流程**：在Web系统上传server的发布包到patch，上传成功后，在web上提交发布server请求，由registry服务传达到node，然后node拉取server的发布包到本地，拉起server服务。
- **管理命令流程**：Web系统上的可以提交管理server服务命令请求，由registry服务传达到node服务，然后由node向server发送管理命令。
- **心跳上报流程**：server服务运行后，会定期上报心跳到node，node然后把服务心跳信息上报到registry服务，由registry进行统一管理。
- **信息上报流程**：server服务运行后，会定期上报统计信息到stat，打印远程日志到log，定期上报属性信息到property、上报异常信息到notify、从config拉取服务配置信息。
- **Client访问Server流程**：client可以通过server的对象名Obj间接访问server，Client会从registry上拉取server的路由信息（如ip、port信息），然后根据具体的业务特性（同步或者异步，tcp或者udp方式）访问server(当然client也可以通过ip/port直接访问server)。

5

Tars技术详解

1．Tars协议

Tars协议采用接口描述语言（Interface description language，缩写IDL）来实现，它是一种二进制、可扩展、代码自动生成、支持多平台的协议，使得在不同平台上运行的对象和用不同语言编写的程序可以用PRC远程调用的方式相互通信交流， 主要应用在后台服务之间的网络传输协议，以及对象的序列化和反序列化等方面。

协议支持的类型分两种，基本类型和复杂类型。

- 基本类型包括：void、bool、byte、short、int、long、float、double、string、unsigned byte、unsigned short、unsigned int；
- 复杂类型包括：enum、const、struct、vector、map，以及struct、vector、map的嵌套。

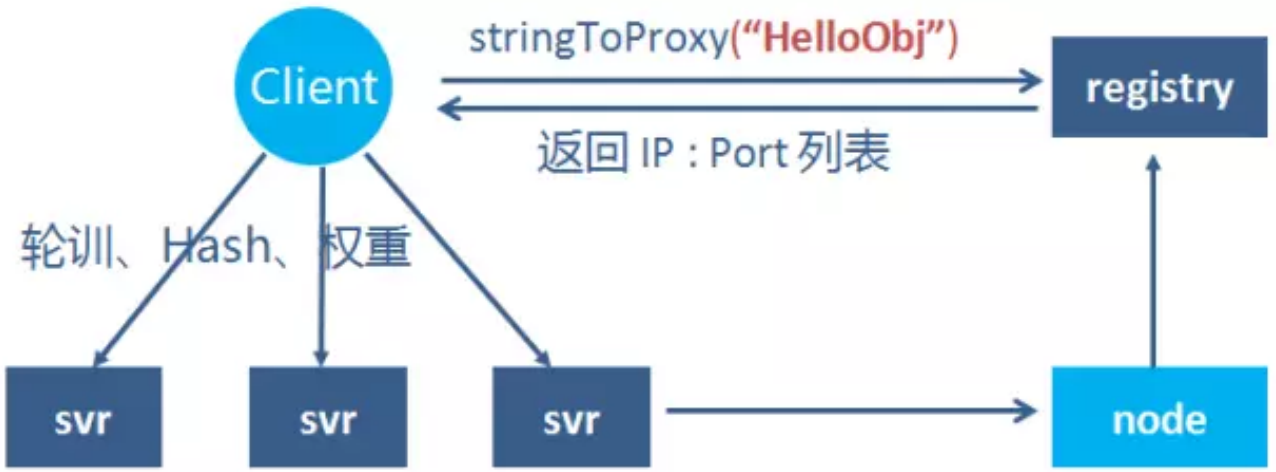
2．RPC调用

框架通过IDL语言协议，可以定义服务提供的接口，并自动生成客户端和服务端的相关通信代码，服务端只需实现业务逻辑即可对外提供服务，客户端通过自动生成的代码即可调用服务，调用方式支持三种模式：

- 同步调用：客户端发出调用请求后等待服务返回结果后再继续逻辑；
- 异步调用：客户端发出调用请求后继续其他业务逻辑，服务端返回结果又由回调处理类处理结果；
- 单向调用：客户端发出调用请求后就结束调用，服务端不返回调用结果；

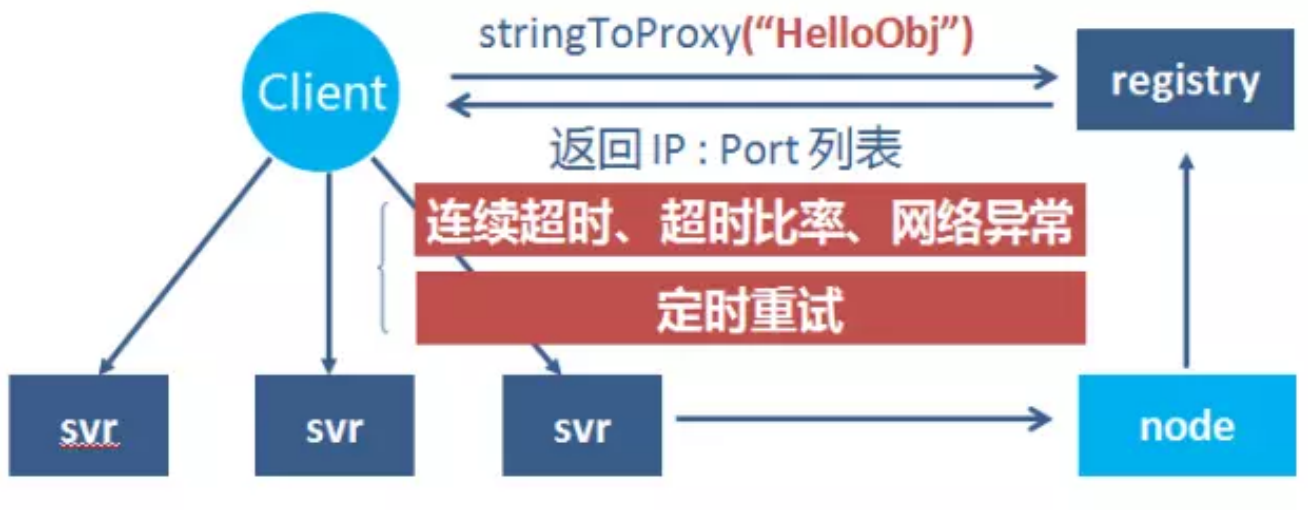
3．负载均衡

框架通过名字服务来实现服务的注册与发现，Client通过访问名字服务获取到被调服务的地址信息列表，Client再根据需要选择合适的负载均衡方式来调用服务，负载均衡支持轮询、hash、权重等多种方式。



4．容错保护

容错保护通过两种方式实现：名字服务排除和Client主动屏蔽。



名字服务排除的策略：

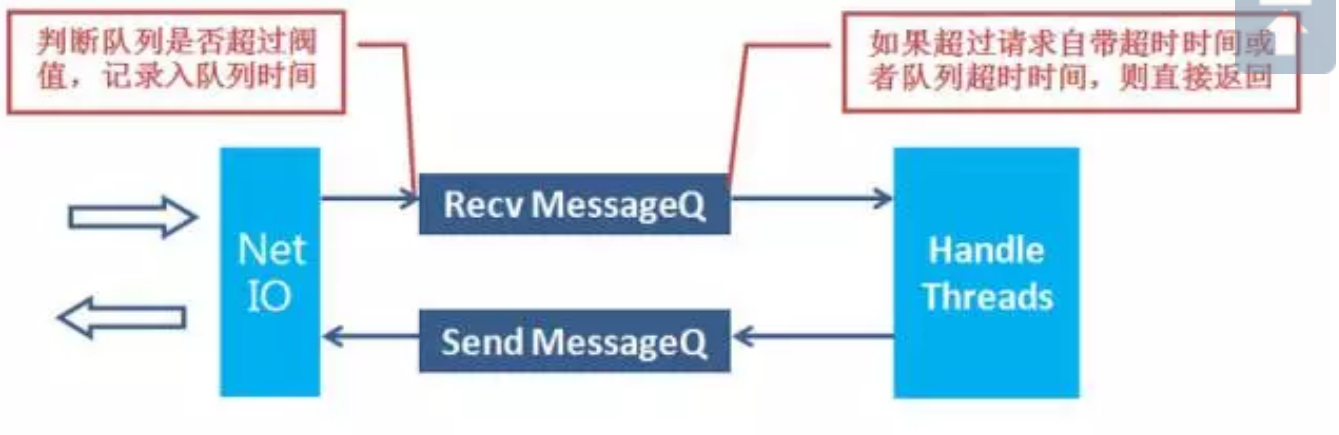
业务服务主动上报心跳给名字服务，使名字服务知道服务部署的节点存活情况，当服务的某节点故障时，名字服务不在返回故障节点的地址给Client，达到排除故障节点的目标。名字服务排除故障需要通过服务心跳和Client地址列表拉取两个过程，故障排除时间在1分钟左右。

Client主动屏蔽：

为了更及时的屏蔽故障节点，Client根据调用被调服务的异常情况来判断是否有故障来更快进行故障屏蔽。具体策略是，当client调用某个svr出现调用连续超时，或者调用的超时比率超过一定百分比，client会对此svr进行屏蔽，让流量分发到正常的节点上去。对屏蔽的svr节点，每隔一定时间进行重连，如果正常，则进行正常的流量分发。

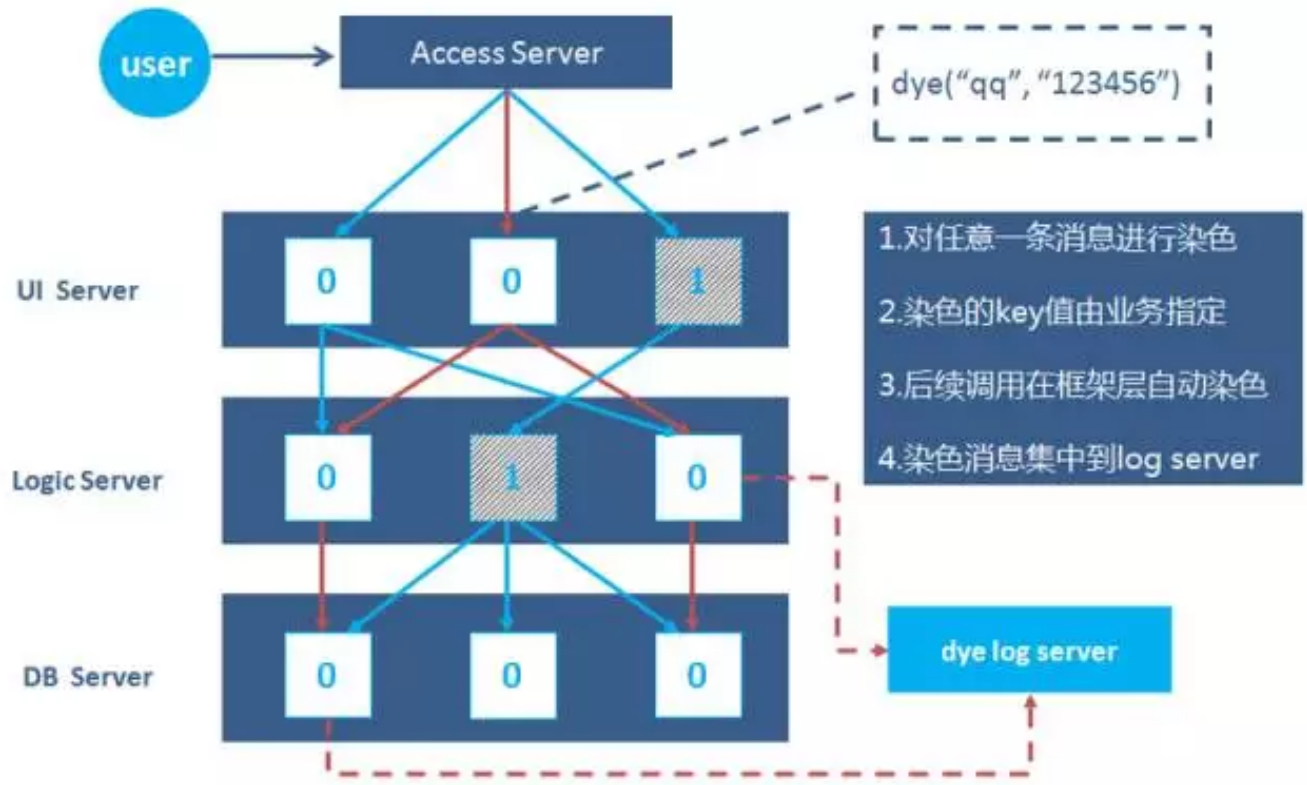
5．过载保护

为了防止业务因为访问量突增或服务器故障造成系统整体的繁忙，进而导致全部服务的不可用，框架内部做相应设计来应对。实现请求队列，服务调用通过非阻塞方式实现异步系统，从而达到提升系统处理能力的目的。并且对队列的长度进行监控，当超过某个阀值，则拒绝新的请求。对请求设置超时时间，当请求包从队列里读取出来是判断请求是否超时，如果超时则不做处理。



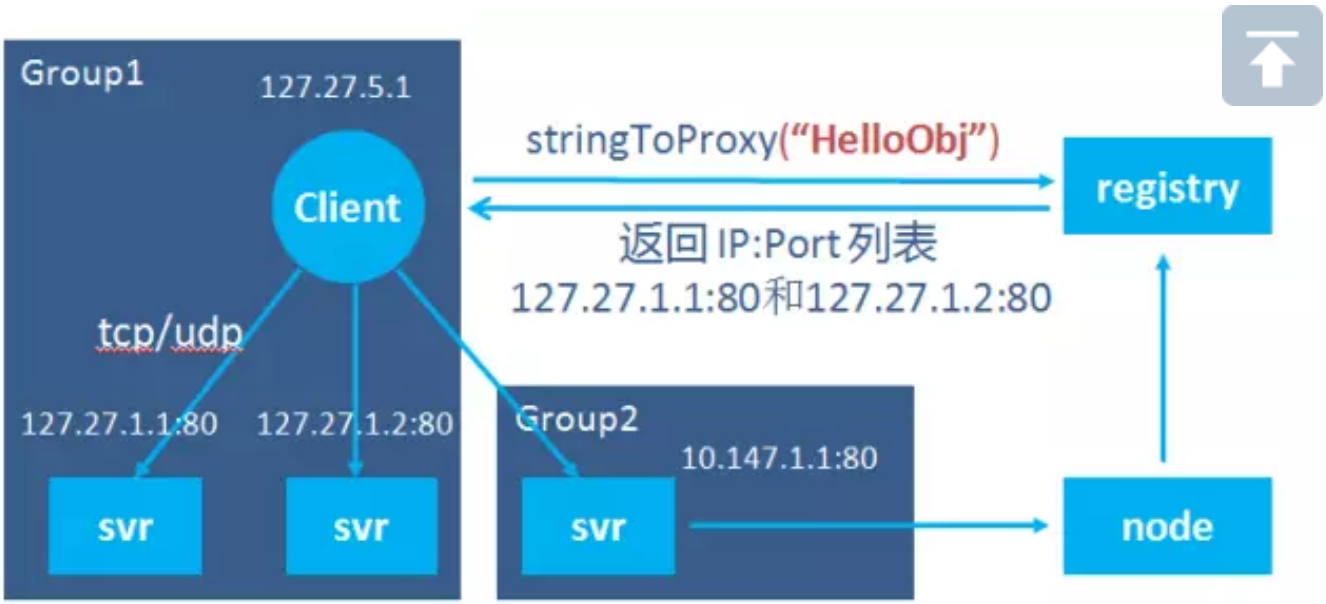
6．消息染色

框架提供了对某服务某接口的特定请求进行染色的能力，染色的消息可以透传到后面需要访问的所有服务上，对染色的请求，服务自动把日志上报到特定的染色日志服务器上，使用者只需在染色服务器上即可分析请求访问的路径，方便跟踪定位问题。



7．IDC分组

为了加快服务间的访问速度，建设跨地区、跨机房调用带来的网络资源消耗，减少网络故障带来的影响，框架提供了跨地区、跨机房，就近接入的功能。



8．SET分组

为了方便对业务服务部署管理进行标准化和容量化，框架提供了Set部署能力，set之间没有调用关系，互不干扰，故障隔离，提高运维效率和服务可用性。

9．运营管理平台

运营管理平台主要包含以下功能：

- 业务管理：包括已部署的服务，以及服务管理、发布管理、服务配置、服务监控、特性监控等；
- 运维管理：包括服务部署、扩容、模版管理等；

10．数据监控

为了更好反映和监控小到服务进程、大到业务的运行质量情况，框架支持以下数据上报的功能:A.提供了服务模块间调用信息统计上报的功能，方便用户查看服务的流量、延时、超时、异常等情况；

B.提供了用户自定义属性数据上报的功能，方便用户查看服务的某些纬度或者指标，比如内存使用情况、队列大小、cache命中率等；

时间点	服务名	IP	特性	策略	当日特征值
20170105	Test.StressServer	10.121.108.160	Test.StressServer.memsize	%	1628056
20170105	Test.StressServer	10.121.108.160	jvm.gc.num.ConcurrentMarkSweep	%	0
20170105	Test.StressServer	10.121.108.160	jvm.gc.num.ParNew	%	0
20170105	Test.StressServer	10.121.108.160	jvm.gc.time.ConcurrentMarkSweep	%	0
20170105	Test.StressServer	10.121.108.160	jvm.gc.time.ParNew	%	0
20170105	Test.StressServer	10.121.108.160	jvm.memory.heap.committed	%	22800
20170105	Test.StressServer	10.121.108.160	jvm.memory.heap.max	%	22800
20170105	Test.StressServer	10.121.108.160	jvm.memory.heap.used	%	5450.4
20170105	Test.StressServer	10.121.108.160	jvm.thread.num	%	454.25
20170105	Test.StressServer	10.121.108.160	req.queue.waitingtime	%	--

C.提供了服务状态变更和异常信息上报的功能，方便用户查看服务的何时发布过、重启过、宕过以及遇到的异常致命错误等；

2016-12-26 16:12:21	TestApp.HelloServer_10.120.129.226	140051658925856	restart
2016-12-26 16:12:19	TestApp.HelloServer_10.120.129.226		patch TestApp.HelloServer succ, version 69
2016-12-26 16:12:17	TestApp.HelloServer_10.120.129.226		[alarm] zombie process,no keep alive msg for 60 seconds
2016-12-26 16:11:15	TestApp.HelloServer_10.120.129.226		[alarm] zombie process,no keep alive msg for 60 seconds
2016-12-26 16:10:13	TestApp.HelloServer_10.120.129.226		[alarm] zombie process,no keep alive msg for 60 seconds
2016-12-26 16:09:12	TestApp.HelloServer_10.120.129.226		[alarm] zombie process,no keep alive msg for 60 seconds
2016-12-26 16:08:11	TestApp.HelloServer_10.120.129.226		[alarm] zombie process,no keep alive msg for 60 seconds
2016-12-26 16:07:09	TestApp.HelloServer_10.120.129.226		[alarm] zombie process,no keep alive msg for 60 seconds
2016-12-26 16:05:59	TestApp.HelloServer_10.120.129.226		patch TestApp.HelloServer succ, version 69

11．集中配置

对业务配置进行集中管理并且操作web化，使配置修改更容易，通知更及时，配置变更也更安全；对配置变更进行历史记录，让配置可以轻松回退到前一版本。配置拉取服务化，服务只需调用配置服务的接口即可获取到配置文件。

为了能灵活管理配置文件，配置文件分为几个级别：应用配置、Set配置、服务配置和节点配置。

- 应用配置为最高一级的配置文件，它是多个服务配置提炼出来的公共配置，服务配置通过引用它来使用其配置内容。
- Set配置是具体一个Set分组下所有服务的公共配置，在应用配置的基础上进行补充追加。
- 服务配置是具体一个服务下所有节点的公共配置，可以引用应用配置。
- 节点配置是一个应用节点的个性化配置，它和服务配置合并成为具体一个服务节点的配置。

6

Tars概述回顾

Tars已经开源，开源地址为：https://github.com/Tencent/Tars，QQ技术交流群：579079160。

目前腾讯内部和外部开源的框架的核心功能和代码都是一样的，主要区别在于运维侧，精简和优化了和内部系统耦合比较深、以及内部运维管理相关的一些功能特性。

Tars适合消息调用客户端和服务端比较明确的业务场景，不太适合消息需要订阅/发布的业务场景。

7

与同类框架对比

Tars与业界其他同类或相识的应用框架相比：

一是Tars提供了支持多语言（C++/Java）的高性能(性能可达40w/s)RPC开发框架，比如业界开源的Dubbo只支持Java，业界开源的Thrift、gRPC性能没有Tars好；

二是Tars具有针对服务进行治理的运营管理平台，比如名字路由与发现、部署/发布/扩缩容、立体化监控、日志管理、配置管理等，让系统的运行状态一切尽在掌握，而业界的Thrift、gRPC只是RPC通信框架，业务在它们之上还要做很多时期；

三是Tars经过多年在不同业务上的实践和发展，其成熟度和稳定性更好，目前腾讯内部有160多个业务（比如手机浏览器、应用宝、手机管家、手机QQ等）、在上万台机器上在使用Tars框架。

作者介绍

钟科，十余年的互联网工作经验，主要负责腾讯移动互联网事业群的基础框架平台的建设，专注于微服务架构、云平台、分布式NoSQL存储等技术领域。

今日荐文

点击下方图片即可阅读

吴恩达：为什么我说人工智能会是划时代的变革?