文件编号：

项目编号：

密级：

分布式RPC架构初探

**V0.2**

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **修订版本** | **描述** | **作者** |
| *2013-10-14* | *0.2* | *初稿，初步描述构想、范围、格式与内容* | *肖海星* |
|  |  |  |  |
|  |  |  |  |

目录

[1 设计目标 4](#_Toc372126085)

[2 参考资料 4](#_Toc372126086)

[3 目标环境平台 5](#_Toc372126087)

[3.1 操作系统 5](#_Toc372126088)

[3.2 硬件平台 5](#_Toc372126089)

[3.3 软件平台 5](#_Toc372126090)

[4 系统概述 6](#_Toc372126091)

[4.1 示意图 6](#_Toc372126092)

[4.2 注册服务器ServiceRegister 6](#_Toc372126093)

[4.3 分布式服务框架ServiceFramework 7](#_Toc372126094)

[4.4 应用服务间通信协议 7](#_Toc372126095)

[4.5 核心服务开发框架 8](#_Toc372126096)

[5 用例与子系统 9](#_Toc372126097)

[5.1 服务定义、安装与查询 9](#_Toc372126098)

[5.2 服务节点登记与状态更新 10](#_Toc372126099)

[5.3 获取服务引用 11](#_Toc372126100)

[5.4 服务升级 11](#_Toc372126101)

[6 系统设计 12](#_Toc372126102)

[6.1 服务定义 12](#_Toc372126103)

[6.2 服务注册 12](#_Toc372126104)

[6.3 服务订阅 13](#_Toc372126105)

[6.3.1 订阅号 13](#_Toc372126106)

# 设计目标

1. 打造一个松耦合、基于服务的架构，简化服务的发布与调用
2. 构建一个统一的模型，统一系统间的操作，从而能把精力集中于服务的接口与业务实现
3. 为服务提供水平伸缩能力，错误处理哲学：允许调用失败，统一失败处理流程，具备分流能力，应用fast-failed策略保证平台不因为局部节点错误产生惊群效应。（需保证可靠结果的服务请参考“可靠消息队列中间件”）

注：将服务进行分类，分别使用不同的框架来实现：应用服务框架、可靠消息队列

1）对时间有要求，宁可失败、中断、丢失部分数据，但不能容忍阻塞，比如浏览网页

2）对可靠性有要求，允许产生一定的延迟，比如订单结算等

# 参考资料

#### 书籍

《OSGi原理与最佳实践》 林昊

《分布式java应用》 林昊

《淘宝技术这十年》 子柳

《Scala程序设计》 Venkat Subramaniam

#### 网络文章

[Twitter研发人员John Oskasson分析Twitter后台软件栈](http://www.infoq.com/cn/news/2013/02/twitter-stack)

[Java中间件：淘宝网系统高性能利器](http://www.searchsoa.com.cn/showcontent_52926.htm)

[淘宝-HSF](http://foreversunyao.iteye.com/blog/963698)

#### 开源项目

[Apache Thrift](http://thrift.apache.org/)

[Finagle](http://twitter.github.io/finagle/)

# 目标环境平台

## 操作系统

Linux: CentOS6

## 硬件平台

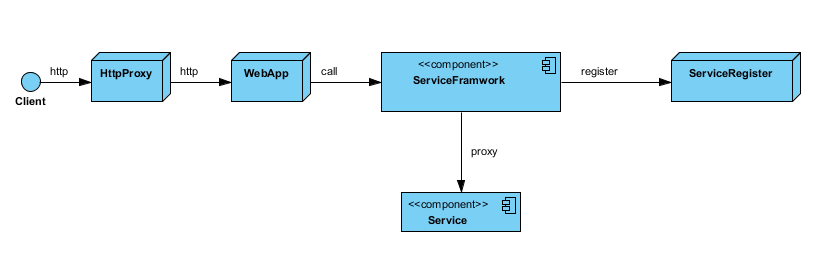
X68

## 软件平台

JVM、Erlang

# 系统概述

## 示意图



## 注册服务器ServiceRegister

注册服务器负责存储并管理服务信息，包括接口文档、节点信息及节点路由策略等。

#### 开发平台

基于在高可靠性上有着优秀表现的Erlang平台。

注册服务器功能单纯简单，要求稳定、高效、灵活、可定制扩展，被系统中绝大部分服务引用，因此将被设计成一个自持的轻量级服务器，不依赖于其他服务器与中间件

#### 通讯机制

ServiceFramework需要支持多种平台与语言并附着于所服务的应用环境运行，为了尽量减少对应用服务器的入侵与对开发团队的平台选型的影响与限制，将采用最简单的基于消息的协议

#### 客户端调用

流程：订阅>下载或更新接口信息与包>加载包>构造接口引用>获取服务节点>返回引用

以上流程应被封装于框架中，使用者只需直接获取接口引用

#### 服务器热备

注册服务器应具备热备能力以避免产生单点故障。对于目前系统及可见将来系统规模，普通PC服务器单机的负载能力即可满足注册服务器的要求，因此不使用集群技术，以免引入不必要的复杂性。

## 分布式服务框架ServiceFramework

#### 设计原则

1. 框架不应挤占应用服务资源，如低CPU与内存占用，不能有高吞吐量的通讯需求等
2. 需要一个简单易用的基于消息的异步通讯，在多语言平台下均能轻易实现
3. 简单轻量，没有过多的重型高侵入依赖，如各种容器、中间件等
4. 不应影响或限制应用服务平台的选择
5. 高可靠可恢复

#### 消息推送

使用RabbitMQ进行框架内部消息推送，鉴于注册服务器也将使用Erlang开发，这将获得简化部署、降低维护成本的额外好处

服务框架内部通讯不选用Thrift作为通讯框架的原因

* Thrift是一种RPC框架，实现了同步与异步的客户端调用，但目前(ver0.9.1)并没有服务端主动推送与分发消息的功能，需要自行扩展这种功能，Thrift本身还在发展中，自行扩展将很可能会在版本升级时遇到困难
* 通过阅读源码，Thrift的传输层在不同语言平台下的程序模型有较大差异，没有一致的表现形式，这就意味着在不同平台下必然会有不同的框架设计与实现，移植开发成本很高
* 不论从哪方面看，RabbitMQ都更适合

## 应用服务间通信协议

1. 第三方RPC

如Thrift，EJB3，WebService

优点就不说了，现成、成熟、可靠等等，问题也有：

RPC侵入、扩展与定制开发困难、应用服务器依赖

1. 方案二：基于自定义RPC协议

如Http + ProtocolBuffer

优缺点与方案一相反，采用哪种方案应根据实际需要进行权衡，如非必要建议采用方案一以降低开发成本

## 核心服务开发框架

经过初步的考察，以Finagle作为核心服务的程序模型基本符合系统目标的开发要求，请参考第二节列出的相关参考资料。

1. 优点

* 结合了Scala的Future模型，支持以异步或同步的方式编写服务
* 可以很容易的处理超时、重试和负载均衡
* 得益于Scala的并发模型，数十万级别的大规模并发成为可能
* Finagle中增加了出色的监控、跟踪的支持，并且Twitter也有相关的开源系统可以参考

1. 缺点

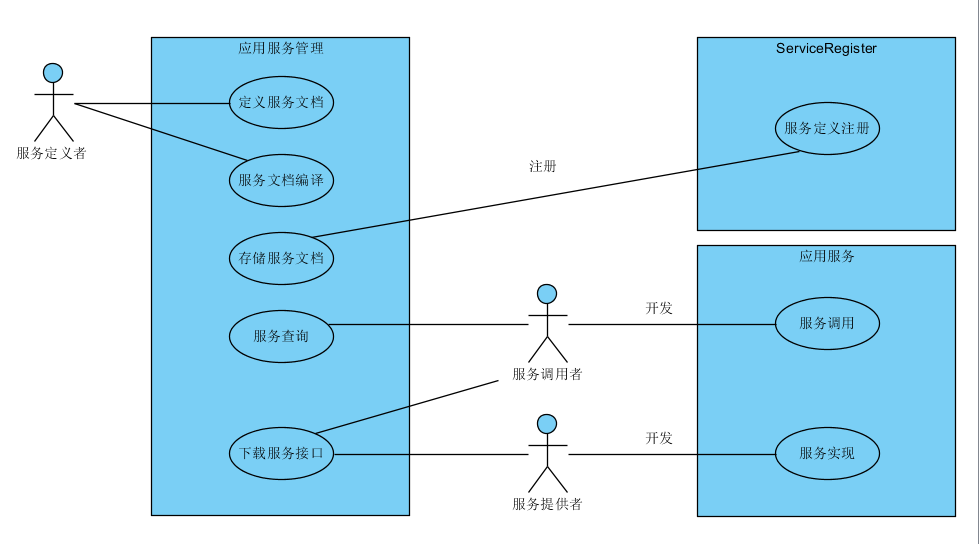
* 要熟练并正确的应用Finagle，必须对的Scala语言及设计哲学有一定的理解，Finagle参考资料少，亦缺少成功应用案例，加上Scala小众的因素，对团队成员要求较高，团队初期的建立与成长将会面临较多困难
* Finagle对Thrift进行了扩展，版本管理与升级将会比较复杂：服务本身版本、Thrift版本、Finagle版本等，需要对开发者使用的版本进行一定的限制

# 用例与子系统

本节通过用例视图概要描述子系统、参与者、边界及其关联，帮助开发人员理解需求、理清思路、迅速对系统建立直观概念，为系统进一步细化提供参考依据

## 服务定义、安装与查询

#### 用例视图



#### 服务定义

1. 服务采用XML格式定义，参考SCA标准
2. 使用Thrift IDL定义接口，以获得较好的性能与跨语言的操作能力

#### 服务文档编译

指定目标开发语言编译Thrift IDL，分别得到客户端与服务端类定义

#### 服务安装与注册

打包上传服务定义文档与类包，并向注册服务器注册，注册名以服务名与接口版本为唯一标识

#### 服务查询

用网页显示已注册服务与客户端信息，如：版本、节点、接口、接口文档、服务超时及超时处理等

#### 服务下载

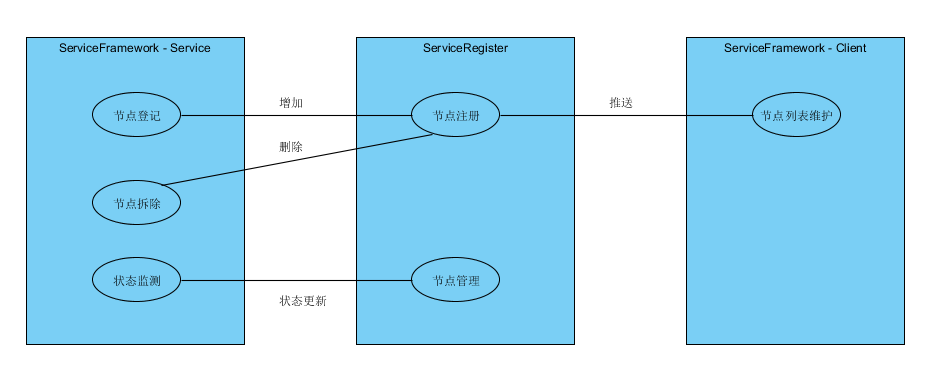
下载服务客户端与服务端包，用于调用或实现服务开发

#### 服务实现

不限定具体实现语言，建议普通业务功能开发使用Java，对高并发有要求的建议使用Scala或Erlang

## 服务节点登记与状态更新

#### 用例视图



#### 节点登记

增加服务实例节点的登记信息

#### 节点拆除

删除服务实例节点的登记信息

1. 温和策略：

决定让一个节点下线，必须先在注册服务器登记为待下线，一个标记为待下线的节点将不再接受新的客户端引用，当引用数为0时注册服务器通知节点，此时节点安全下线

1. 强制策略，某些情况下会有这种需求

节点强制离线，已获得引用的客户端将收到通知，正在进行的调用应获得失败或超时信息

#### 节点管理

1. 服务信息更新与存储
2. 节点列表存储于数据库
3. 服务与节点状态存储于内存
4. 节点根据策略脚本进行排序，并将变化推送给服务订阅者

##### 状态监测

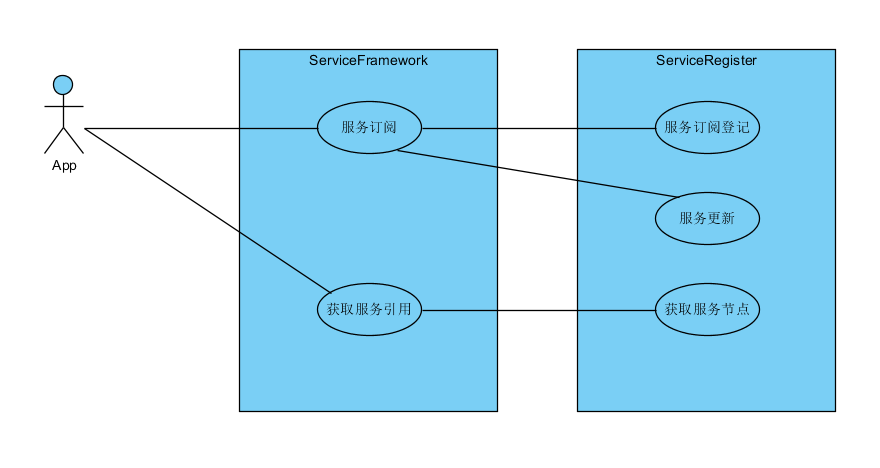
1. 节点心跳
2. 服务QoS：可用性、吞吐量、时延
3. 节点QoS：CPU、内存、磁盘

#### 节点列表维护

性能均衡代理可以快速引用此列表而无需访问注册服务器

## 获取服务引用

#### 用例视图



#### 服务订阅与登记

#### 服务更新

根据服务更新策略在发出更新事件并记录日志后选择是否使用新版本或是直接退出

#### 获取服务引用

服务引用通过动态代理获得均衡路由、功能路由、超时或错误处理等能力

#### 获取服务节点

允许通过策略脚本选择节点，典型应用：

1. 版本更新时指定部分客户端使用新版本节点
2. 通过轮询、随机、QoS等策略进行节点选择以实现性能均衡

## 服务升级

1. 接口版本升级，处理：直接注册新的服务
2. 接口不变服务端实现升级，处理：在注册服务器登记升级信息，由客户端配置自行选择处理方式

# 系统设计

## 服务定义

#### 接口定义

使用Thrift IDL定义接口文件，示例如下：



#### 服务定义

使用XML文件格式，示例如下：

<dsf>

<service name="tutorial.Calculator" version="1.0">

<describe> This service aims to teach you how to use Service</describe>

<protocol\_type>binary</protocol\_type>

<route\_strategy>roundrobin</route\_strategy>

</service>

</dsf>

## 服务注册

服务监控

1、注册的服务，超过30\*3秒没有报告状态（心跳、QoS等）视为失效，将从服务列表中删除，直到再次报告状态

## 服务订阅

### 订阅号

#### 目的

1. 标识客户端
2. 避免每次客户端离线都需要重发所有订阅内容。服务端不了解客户端离线的原因，如果客户端是因为退出离线则需要更新所有订阅的内容，如果客户端是掉线则无需重发订阅内容

#### 订阅号生成

服务端生成一个唯一序列号

#### 生命周期

客户端：进程

服务端：离线超时（5分钟）

离线超时时间不宜设置过长，以免服务端维护过多无效的信息， 比如因离线而保存在队列中等待推送的消息

Thrift

一个端口对应一个service