

题 目 基于微服务架构的统一分析运行平台的设与实现

软件学院 院（系） 软件工程 专业

学 号 71113333

学生姓名 陈浩远

校内导师 刘奇其

企业导师 薄宏剑

起止日期 2017年1月 ～ 2017年6月

设计地点 华为技术有限公司南京研究所

基于微服务架构的统一分析运行平台的设计与实现

# 摘 要

随着电信行业3G/4G等业务的增加，运营商手中的数据量出现了爆炸式的增长，数据的规模已经达到了PB级别。海量数据带来了大量的信息，同时也迫使电信行业为了处理这些数据而不得不使用hadoop/spark等分布式大数据处理技术。由于电信运营商自身结构复杂，产品和业务繁多，电信用户多，每天能产生大量的数据，因此产生了很多不同的大数据任务需求，对于如何管理和调度众多的大数据任务也成为了运行商头疼的一件事。

为了实现对众多任务的统一调度和管理，解决运营商在数据分析中面临着的挑战，本文首先对企业软件的发展现状做了调查，结合对统一分析运行平台的需求分析，设计了该平台的总体方案，对系统的各模块进行了详细的设计，并针对企业平台所需的安全性、容错性、高效性等进行了分析，给出了统一运行平台的实现细节，针对统一运行平台所需的特性进行了测试，最后给出了统一运行平台的部署方案。

关键词：微服务；分布式；大数据；docker

A PAAS Platform Based on Micro Service Architecture

# ABSTRACT

With the telecom industry 3G / 4G and other business increases, the amount of data in the hands of operators there has been explosive growth, the size of the data has reached the PB level. Massive data has brought a lot of information, but also forced the telecommunications industry in order to deal with these data had to use hadoop / spark and other distributed large data processing technology. Because telecom operators have their own complex structure, many products and services, many telecom users can produce a lot of data every day, resulting in a lot of different large data task requirements, how to manage and schedule a large number of large data tasks have become runners A headache for one thing.

In order to realize the unified scheduling and management of many tasks and solve the challenges faced by operators in the data analysis, this paper firstly investigates the development of enterprise software, and analyzes the requirements of the unified analysis and operation platform. The overall program, the system of the modules were designed in detail, and for the enterprise platform for the security, fault tolerance, efficiency and other analysis, given the implementation of a unified platform for the details of the platform for the unified operation of the required The characteristics of the test, and finally gives a unified platform for the deployment of the program.

Keywords: Micro service; distributed; big data; docker

目 录

[摘 要 i](#_Toc484143549)

[ABSTRACT ii](#_Toc484143550)

[第一章 引 言 1](#_Toc484143551)

[1.1 选题背景和意义 1](#_Toc484143552)

[1.2 国内外研究现状 1](#_Toc484143553)

[1.3 论文主要内容 2](#_Toc484143554)

[1.4 论文组织结构 3](#_Toc484143555)

[第二章 系统需求分析 4](#_Toc484143556)

[2.1 系统功能需求 4](#_Toc484143557)

[2.1.1 网络流量采集 5](#_Toc484143558)

[2.1.2 网络流量分析 6](#_Toc484143559)

[2.1.3 流量监测配置 6](#_Toc484143560)

[2.2 系统数据需求 7](#_Toc484143561)

[第三章 系统方案设计 8](#_Toc484143562)

[3.1 系统架构设计 8](#_Toc484143563)

[3.2 网络流量采集功能设计 9](#_Toc484143564)

[3.3 网络流量分析功能设计 12](#_Toc484143565)

[3.4 流量监测配置功能设计 13](#_Toc484143566)

[3.4.1 基础配置 13](#_Toc484143567)

[3.4.2 高级配置 13](#_Toc484143568)

[3.4.3 组配置 14](#_Toc484143569)

[3.5数据库表结构设计 14](#_Toc484143570)

[3.5.1 汇聚数据表结构 14](#_Toc484143571)

[3.5.2 配置数据表结构 15](#_Toc484143572)

[第四章 系统实现与测试 20](#_Toc484143573)

[4.1 网络流量采集功能的实现 20](#_Toc484143574)

[4.1.1 网络流量采集流程 20](#_Toc484143575)

[4.1.2 网络流量接收服务 20](#_Toc484143576)

[4.1.3 网络流量聚合服务 22](#_Toc484143577)

[4.2 网络流量分析功能的实现 24](#_Toc484143578)

[4.3 流量监测配置功能的实现 31](#_Toc484143579)

[4.4 系统测试 35](#_Toc484143580)

[4.4.1 测试目的 35](#_Toc484143581)

[4.4.2 测试环境 36](#_Toc484143582)

[4.4.3 测试过程及结果 36](#_Toc484143583)

[第五章 工作总结与展望 38](#_Toc484143584)

[5.1 工作总结 38](#_Toc484143585)

[5.2 工作展望 38](#_Toc484143586)

[致 谢 39](#_Toc484143587)

[参考文献 40](#_Toc484143588)

# 第一章 绪 论

## 1.1 引言

随着上网用户带宽和流量的增加，运营商对用户产生数据分析和处理的需求也越来越大，本文旨在解决电信行业存在的大量的hadoop和spark任务的调度问题，以分布式的结构，提供一个提供电信分析应用所需的丰富的、高性能的分析引擎，为统一分析开发平台、统一数据治理平台提供统一的运行环境和运维功能。将单独部署的的多种不同的分析引擎，按照运营商自身的业务场景和它们所需的模型进行整合，协助运营商解决以往的运行环境过于简单的问题。

之所以采用微服务，是为了避免传统的分层架构带来的诸多问题，微服务也整合了过去十年来的新概念和技术，因此可以避开许多面向服务的架构中的陷阱。

在多年前，很多组织就开始尝试使用更细粒度的架构来实现更快的交付，结果发现其带来了更好的可扩展性，增强了团度的自治。微服务就是一个这样的快速发展的主题，尽管它不是一个新的想法，但它受了多年来的各种各样的架构的影响，因此，它在当前阶段具有很好的前瞻性和挑战性。

## 1.2 运营商大数据现状

全球信息数据量的迅猛增长成为大数据产业发展的基础。市场调研机构IDC预计，未来全球数据总量年增长率将维持在50%左右，到2020年，全球数据总量将达到40ZB。其中，我国数据量将达到8.6ZB，占全球的21%左右。另外，截至去年12月，我国网民规模达6.88亿人。其中，手机网民规模达6.20亿人，移动互联网流量呈现爆发式增长。工信部统计显示，去年全年移动互联网接入的流量超过400万TB，同比增长了103%。随着全球数据总量以及移动流量的不断提升，运营商的大数据任务不断增加，因此有必要开发一个统一运行平台来对这些任务进行编排和调度。

电信行业迈入数字化精细运营时代，大数据成为使能和引擎核心引擎。大数据既是运营商数字化转型的使能和引擎、同时也是数字化服务的重要内容。在技术上，大数据技术不断地演进和成熟；商业上，数据是运营资产；运营上，大数据作为新思维、新工具可以驱动运营效率的提升和创新；组织上，大数据是组织变革的切入点。

根据Accenture、Gartner趋势分析报告，2014年大数据从技术驱动转变为商业驱动，企业经营优化和数据资产变现成为主要诉求，2017年基于大数据的商业模式创新(Big Data As A Service、数据交易）将会爆发式增长。数据处理技术经过2010-2012年的疯狂发展，将会逐步标准化、云化，成为类似于 RDBMS的通用的IT软件基础设施，但大规模并行计算运维 仍然会成为关键壁垒。

运营商当前在大数据应用时面临如下问题：

* 工具不统一，多厂家异构集成后难以将工具功能集成到端到端解决方案，最大化呈现业务价值。
* 海量电信数据烟囱式分布，难以整合。
* 数据管理分散，难以共享。
* 数据获取过程繁琐，开发困难，对专家依赖高。

具体来说：

* 开发环境：多厂商、多平台造成的多种不同的开发环境，分散了开发资源；关联业务功能/模块被切割，额外增加集成成本；难以形成统一的、相关联的管控机制。
* 运行环境：较多应用运行在不同的环境中，资源利用率不高；跨环境集成降低了运行效率；缺乏统一管理能力。
* 生命周期管理：没有统一的运维管理手段，无法开展多租户管理，面向市场开放能力。
* 调度管理：调度计划难以管理；调度进程难以监控；调度策略难以改进优化。

## 1.3 企业软件架构发展现状

从50-60年代开始，软件技术主要为高级数据/控制结构，具体来说就是函数的重用，为了解决代码的重复出现，这一阶段是编程语言高速发展的阶段。随后70年代，出现了面向结构的架构，这种架构采用自顶向下、逐步求精的设计方法；任何的程序都可以由顺序选择和循环三种控制结构来实现；处理的对象为数值数据。到80年代则出现了面向对象的架构，对象是人们要进行研究的任何事 物，对于同类的事物，可以由一个对象来表示；通过对象抽象，可以大大减轻设 计人员的工作量，简化实际的模型；以数据对象为基础构筑，增加代码可重用性。90年代出现了面向组件的架构，面向组件技术建立在对象技术之上的进一步发展；组件技术的主要目标是粗粒度组件重用；复用性、灵活性大大提高。到04年，SOA已经成为了各大厂商流行的架构风格，SOA引入了服务的概念，它将系统的扩展变得更为灵活，更能响应软件业务的变化，它是基于业务总线ESB的一种架构。而几年来，微服务则成为各大厂商争相关注的对象，相比SOA，微服务不再关注业务总线，它将服务的概念引入了每一个小模块，而且更为依赖网络通信。

## 1.3 本文研究目的和研究内容

本文主要围绕华为的统一分析运行平台的设计和实现，介绍了微服务架构的特性，以及如何通过Spring Cloud实现微服务。该平台解决了运营商对大量任务缺少统一的管理调度系统的问题，根据现有的软件体系结构，解决了企业软件所需的高性能、高容错、高安全性的需求，保障了系统的长期稳定的运行，并且对以后的开发和维护有着较好的支持性。

## 1.4 论文组织结构

本论文主要介绍了基于Spring Cloud的统一分析运行平台的设计与实现，主要内容组织结构如下：

第一章 绪论，通过对运营商现状和软件结构的发展，介绍了该论文的研究目的和研究内容。

第二章 系统需求分析，。 //TODO

第三章 系统设计，。

第四章 系统实现与测试，

第五章 工作总结与展望，对论文内容进行了总结，以及对系统当前不足提出的一些日后可以实行的优化改进建议。

# 第二章 系统需求分析

## 2.1 系统功能需求

随着电信业务的不断拓展，电信资费越来越低，电信行业产生了越来越多的数据，传统使用人工单独调用大数据任务的方法已经不能满足电信行业所需的大量任务，大量的任务存在与不同的机器上，使得对任务的管理变得越来越困难，大量的机器上运行着早已不再需要的任务，严重浪费了机器的性能和管理人员的精力。为了提高对机器的利用率，降低人力成本，运营商急需一个可以统一管理和调度的平台。

对于统一分析运行平台，需要有业务编排、上下文、按照业务流程进行运行、调用其他服务的功能，在这里我们将多个业务通过编排后组成的业务流称之为任务模板，对于任务模板，统一运行平台要有运行多个任务模板的能力，并且要有对任务模板的管理功能。

### 2.1.1 业务编排

将多个业务按照一定的逻辑和顺序编排在一起。这是统一分析运行平台的关键性功能，他可以将多个业务组合在一起，相比以往需要运维人员逐个执行的业务在这里可以轻松的一次性执行。有了这个功能就可以更好的利用运算平台的性能，并且提供相比人工而言更复杂的管理功能。

要求业务编排需要实现以下的功能：

* 顺序执行：以串行的方式按照编排的顺序逐个执行业务，只有当前一个业务结束才可以开始下一个业务。如图2.1所示，这张图展示由一个人饥饿引发的简单过程。结果是必须有人去购买材料和做饭，之后有人吃了饭，他的饥饿会得到满足。这些业务之间的关系就是顺序执行的关系。

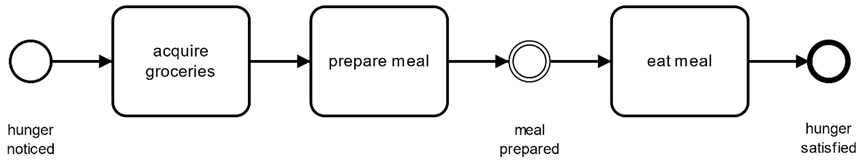


图2.1饥饿过程的顺序执行

* 条件执行：在当前业务后面跟着两个分支，在执行完当前业务之后，判断该处的条件，如果满足条件则接下来进入一个分支，如果不满足条件则进入另一个分支。如图2.2所示，这是一个在线商店从收到订单到用户收到物品之间的一个装货送货的过程，在这中间有很多个条件执行的业务，例如在第一个有效的业务中，决定是要使用普通方式配送还是特殊方式配送的方式，如果满足普通配送的条件则向上进入普通配送的流程，如果条件不满足普通配送的条件则向右进入特殊配送的流程。

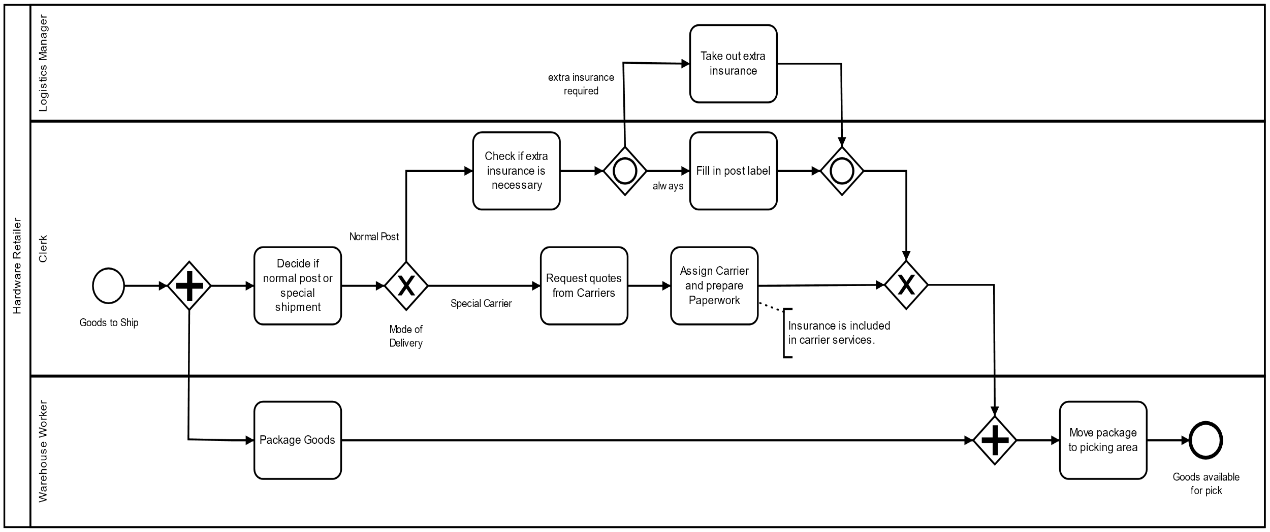


图2.2一个装货过程

* 并行执行：多个业务在同一时刻一起执行，处于并行关系的两个业务之间并不需要互相等待，他们可以一同执行。
* 定时事件：某个业务可能并不是按照顺序或者条件的逻辑作为开始标记，而是以一个时间作为自己的开始条件。如图2.3所示，这是烤蛋糕的过程，执行完检查蛋糕是否做好的业务后进入一个条件判断，如果条件判断为没有做好则定时五分钟后再执行下一个业务。

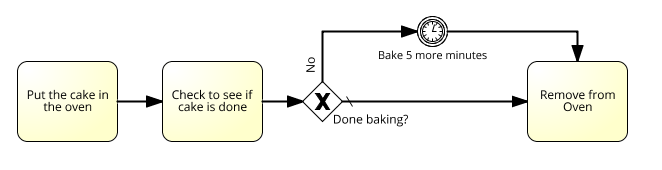


图2.3一个烤蛋糕的过程

对于业务编排之后形成的东西，我们称之为任务模板，统一分析运行平台需要有对任务模板的管理功能，具体分为：创建任务模板、修改任务模板、删除任务模板、获取全部任务模板、单独查询任务模板。

### 2.1.2 上下文

因为有时候并不止是单纯的把任务模板丢到引擎中开始执行，部分任务模板需要有一些环境变量才可以执行，有些业务可能需要传入一定的参数，例如：有些特定的hadoop业务需要在运行时指定结果的存放路径；有些业务有多种执行模式，需要依据环境变量才能判断按照哪种模式执行。

统一分析运行平台提供一个上下文的机制，可以向上下文环境中传入任意的参数，这些参数能够让引擎中的业务在执行的过程中获取，并且要对外提供一个创建上下文、修改上下文、查询上下文以及删除上下文的功能。

### 2.1.3 任务实例管理

在统一分析运行平台中，任务模板是有多个业务组成的一个业务流，而任务实例则是任务模板的一个实例化，在只有一个业务需要执行的情况下，我们也要求它先组成一个任务模板，实例化为任务实例后才能放入引擎中去运行。因此，任务实例就是统一分析运行平台的最小执行单元。

因为统一分析运行平台中会同时运行大量的任务实例，因此需要有对任务实例的管理功能。具体分为：创建任务实例、修改任务实例、删除任务实例、获取全部任务实例、单独查询任务实例。

## 2.2 系统特性需求

随着软件架构的发展，企业软件逐渐从单体应用向分布式的应用过渡，对于企业软件的各种特性要求也越来越多。针对统一运行平台而言，由于该平台是大量业务的管理调度平台，因此该平台对企业有着很高的重要性，并且该平台也会承担大量的数据访问。对于该平台，我们对他的安全性、容错性、高效性、扩展性、部署难度等都有较高的要求。

### 2.2.1 安全性

随着时间的推移，越来越多的大型系统被曝出安全漏洞，这些漏洞往往会导致很多严重而又危险的事情。例如，棱镜计划就是美国国家安全局存在着的安全漏洞，最终导致了整个国家的安全都受到威胁。统一运行平台对公司来说同样是极其重要的一个平台，因此我们在设计系统时一定要尽可能地保证他的安全性。具体需求如下：

* 身份认证和授权：对于整个系统，我们要求它拥有认证和授权机制，并且可以对不同的用户设置不同的权限。对于分布式系统的单个节点，我们要求该节点只对需要直接交互的其他节点进行认证，其余外在系统和平台内不与其直接交互的节点认证均不通过（即进行细粒度的授权）。
* 数据安全：虽然授权机制能在一定程度上保证统一运行平台的安全性，但是一旦操作系统被黑客攻破，我们的敏感数据和密钥都面临着巨大的威胁，因此，我们要求对密钥和敏感信息进行加密处理。
* 防火墙：华为要求所开发的软件必须明确其开放端口，对于不需要的端口我们会全部关闭，以保证系统没有后门也没有多余的端口暴露漏洞。
* 日志：系统要求对所有涉及到数据和登陆的操作都有日志记录。完善的日志虽然起不到预防攻击的作用，但是可以在发生难以预测的事情时重现当时的情景。同样，我们需要在日志中剔除敏感信息。
* 网络隔离：对于分布式系统而言，并不是所有的节点都需要同样的网络环境，我们可以通过将节点分配到不同的网段，这可以进一步避免内部节点的暴露，提高整个系统的安全性。
* 操作系统：操作系统要求选用打好安全漏洞布丁的操作系统，并关闭操作系统的不必要的服务，定期检查该操作系统曝出的安全漏洞，在操作系统出现漏洞的情况下及时做出响应。

### 2.2.2 容错性

分布式系统相比单体系统而言，其中一个重要的有点就是他可以对系统的容错性进行很好的设计和控制。系统的容错性是指当系统发生故障时，系统还能保证运行和数据安全，并且可以从错误中恢复过来，其中的故障可能是系统和软件在运行中出现的运行时错误，也可能是机房的硬件故障，甚至是断网断电的情况。对统一分析运行平台来说，要保证部分节点出故障的情况下，平台还能保证功能的完整性，对于故障节点所涉及到的数据可以完整地保存起来，以备之后的恢复。

### 2.2.3 高效性

这里的高效性是指对整个系统的性能需求，是这个系统要能够承受预估范围内的访问数据量。对于统一分析运行平台而言，其在交付后可能会同时运行上万个的任务。我们需要在能预知大概数据量的情况下，不对机器做过多的浪费。

### 2.2.4 扩展性

分布式系统相比单体应用而言，他可以针对特定的模块进行多个扩展。要求统一分析运行平台的多个模块能够按照所需的性能进行扩展，并且要求可以在运行时对模块进行扩展，扩展之后的模块要在一分钟内能够加入系统并分担一部分数据量。

### 2.2.4 部署难度

初次部署要求提供完整详细的文档和脚本，要求系统易于部署。而在之后需要修改平台代码的情况下，我们只需要重新部署这行代码所涉及到的这个节点。

# 第三章 系统设计

## 3.1 系统架构设计

传统的单体结构的应用很难应对每秒上万次的调用，对于应用的容错性也很难保障，应用内一旦出现一个致命性的错误将导致整个应用的停止运行，这对运行商来说将会带来巨大的损失，基于微服务架构的应用可以完美的解决这一问题。

微服务就是采用网络来进行分割模块，每个模块都是一个小的服务，这些小服务彼此间只通过网络进行交互，单独的微服务完全拥有自治的能力，因此微服务的架构可以带来：

* 技术异构性：因为服务之间通过网络进行协作，除此之外，服务之间完全独立，因此可以在不同的服务内使用不同的框架，甚至是不同的语言，为了达到微服务的正确运行，只需要统一服务间的网络协议和数据格式。但是，在一个组织内使用过多的语言也会导致组织难以招聘人才和找不到合适的人交接工作。
* 弹性：在单块系统中，如果一个组件不能用，整个系统都会受到影响，但是在微服务中，我们可以轻易的将一个服务部署成分布式的。
* 扩展：在单体的软件中，想要实现分布式的扩展，则必须将整个系统进行扩展，这意味着及时系统中只有一个小模块存在性能问题，我们也要浪费大量的机器去扩展它。而微服务具有天生的可扩展性，可以针对需要扩展的微服务进行扩展。
* 简化部署：在大型的单体程序中，即使只修改了一行代码，也需要把整个程序进行重新部署，但是在微服务中，我们只需要重新部署这行代码所涉及到的这个微服务即可。
* 与组织结构相匹配：微服务可以有效的将模块分离出来，这也使得整个项目组团队可以分割成在各地的小型团队，服务的所有权也可以随意的在团队间转移，能够保证所有相近服务的团队都在一起工作。
* 可组合性：微服务最大的好处就是可以对各个服务进行编排重组，不同于以往的单纯的Web、PC端程序，人们有更多的选择使用同一个功能。微服务中会开放很多接口供外部调用，当情况发生改变时，可以使用不同的方式构建应用。
* 对于可替代性的优化：在单块的系统中，我们想要修改一写历史遗留的问题时，常常要思考这次修改所需要付出的代价有多大，因此，跟多公司可能还在维护者很多年前就该淘汰的代码，这些代码没人能看得懂，甚至每人敢动他。在微服务上，我们基本可以保证一个服务可以在两周时间内重写，这也就意味着我们想要替换掉这个服务时所付出的代价并不是那么大。

微服务之间的完全通过网络进行了解耦，服务之间的联系就如图3.1所示。

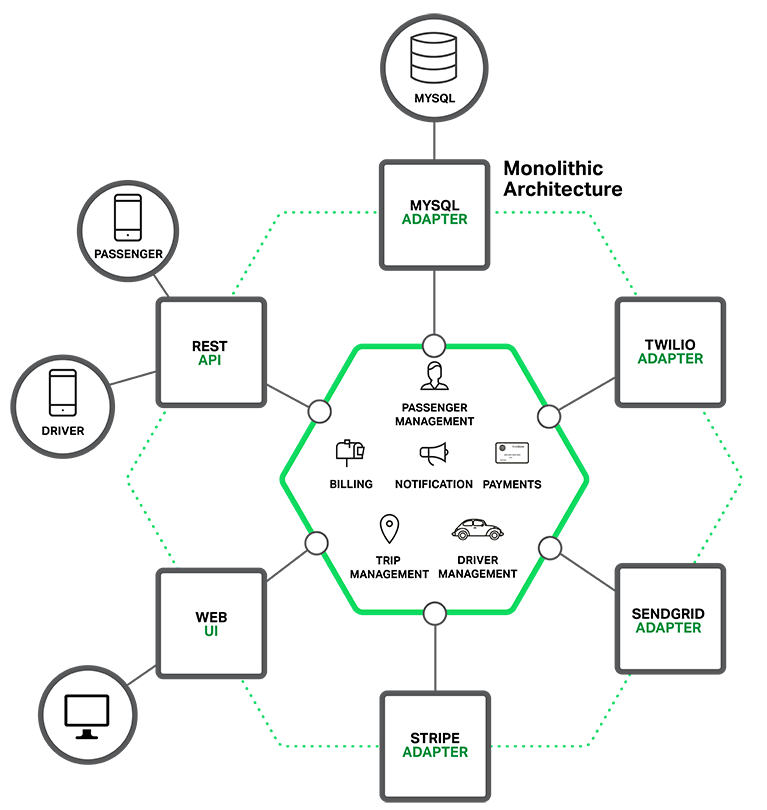


图3.1 微服务之间的联系

综上所述，统一分析运行平台采用微服务架构作为系统架构。

在统一分析运行平台中，我们的架构入图3.2所示，划分为任务模板模块、任务实例模块、BPMN引擎模块、Ignite模块和Eureka模块五个模块，其中负责和用户交互的模块为任务模板模块和任务实例模块，对于调用相应业务的功能由BPMN引擎模块完成。

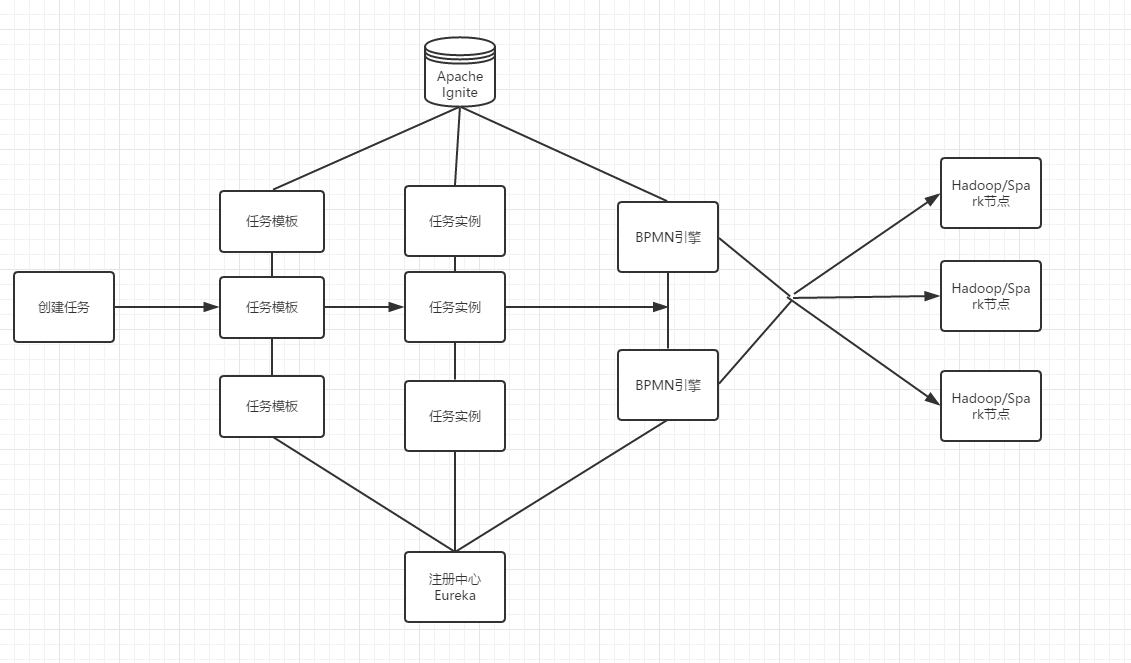


图3.2 系统整体架构

## 3.2 服务交互

对于微服务而言，服务之间通过网络进行交互，常采用的方式为RPC或者REST。下文将通过对比RPC和REST的优缺点，确定在统一分析运行平台中所选择的交互方式。

### 3.2.1 RPC

在分布式计算中，远程过程调用(RPC)是当一个计算机程序引发另一个地址上的程序执行,并且在这一过程中没有涉及到程序的代码细节,就好像它是一个正常的本地调用,无需程序员显式编码远程交互的细节。也就是程序员不管是远程调用还是本地调用都写相同的代码。这是一个典型的client-server交互，调用者是客户端，执行者是服务器，通常通过请求-相应消息传递系统来实现。

RPC的具体实现有很多，如JAVA的RMI、或者通过暴露一个SOAP接口来实现，不同的RPC实现所用的消息格式不同，使用的技术栈也不同，但是作为调用的双方则必须使用相同的格式和技术。

因此，RPC可能带来技术上的耦合，比如RMI只能用于JAVA的平台，因此这就要求被调用者和调用者的双方都选择在JAVA平台上开发，这就对技术选型造成了一定的限制。另一方面，由于RPC像本地调用一样，所以想要修改一个方法则必须对本地和远程的代码都进行修改，这是开发过程中C/S端耦合的一部分。

另外，RPC虽然用起来和本地调用一样，因为RPC将远程调用的中间过程给隐藏起来了，但是这也带来了很多不确定性，比如性能上，不同的RPC调用的性能不同，而且还有网络通信所需要的时间，这就可能导致不明真相的开发人员把远程调用当然本地调用来使用，最后造成性能上的问题，甚至会由于网络的问题而出现各种各样的错误。

### 3.2.2 REST

REST是Representational State Transfer的缩写，它是由Roy Thomas Fielding博士在2000年提出来的一种架构风格。Roy Fielding也是http规范的主要作者之一。

REST是一种设计风格，REST本身有很多的约束和限制，但在这里，我们仅将他和RPC进行对比。REST的关键点为：

* 资源与URI：任何被引用到的事物都可以看作一个资源，例如在URI中，我们的每一个地址都可以看作一个资源，这个地址的命名可以很好的描述这是一个怎样的资源。
* 统一资源接口：这个接口预先定义了可能对资源的操作，放在http里面则对应着http的动词，例如：GET一般表明获取资源，它是安全的，它的操作是幂等的；POST表明更新资源，它意味着将要去修改服务器上的资源，它的操作不是幂等的；PUT一般表明要去创建一个资源，它是不安全的，但是是幂等的；DELETE则是要删除一个资源，是不安全的，但是是幂等的。
* 资源的表述：资源的表述基本上是指用怎样的数据格式去形容这个资源，对外表现为资源的具体呈现，用来进行传递资源。比如文本我们可以使用json或xml，图片我们可以使用PNG或JPG展现，如图3.3所示，这是一个使用json作为资源表述格式的rest请求。



图3.3 http请求

* 资源的链接：这一点奠定了互联网下一代获取资源的方式，就是我们的下一个要访问的资源链接都可以在上一个资源处找到，这也就是超媒体即应用状态引擎的概念，但是目前我们的微服务应用还不能很好的实现这一点。
* 状态的转移：REST是一种无状态通信原则，他在服务端并不会有状态，服务端只维护资源，而客户端维护状态，状态转移就是由客户端维护如何进入下一个状态。

HTTP实现了很多REST的功能，因此HTTP通常会用于构建RESTful软件，我们只需要在使用HTTP时按照REST对资源的定义来实现，就可以基本地实现符合REST风格的软件。

另外HTTP与RPC并不冲突，RPC可以通过HTTP来实现，例如SOAP，但是这并没有完全用到HTTP的统一资源接口等功能。

### 3.2.3 RPC和REST的选择

由上可知，REST的功能和易用性都比RPC要强，但是我们不能说RPC一无是处，一方面部分RPC框架可以自动的生成桩代码，这使得使用RPC的工作量将要比REST小一些，另一方面，基于http的REST的每次请求都需要完好的封装http协议，而http是建立在TCP上层的协议，而且很多的http客户端框架并不能很好的实现http的长连接，这就导致在一次http请求之后，TCP连接会断开，而再次请求还需要重新建立http请求，这就使得在有低时延需求的情景下，http比RPC的时延可能要高。

在系统开发之前，我们针对REST和RPC进行了一些相关的测试，测试方法如下：

* 选择相同的配置的四台机器作为测试机，其中两台用来测试REST的接收和发送，另外两台测试RPC的接收和发送。
* 使用Java RMI实现RPC；使用jersey实现rest服务端，通过封装JAVA内置类HttpURLConnection实现rest客户端；RPC和REST服务端均只在接受到调用后累加器加一。
* 分别为RPC和REST编写测试函数，对两种调用方式的用时进行统计。

RPC客户端代码如下：

publicclass Program {

publicstaticvoid main(String[] args){

try{

//调用远程对象，注意RMI路径与接口必须与服务器配置一致

RPCService rPCService=(RPCService)Naming.lookup("rmi://10.163.14.19:6600/RPCTest");

int counter;

long start = System.currentTimeMillis();

for(int I;I <= 20000;i++){

counter = rPCService.add();

}

long end = System.currentTimeMillis();

System.out.println("Took : " + ((end - start) / 1000));

}

}catch(Exception ex){

ex.printStackTrace();

}

}

}

RPC服务端代码如下：

//此为远程对象的实现类，须继承UnicastRemoteObject

publicclass RPCServiceImpl extends UnicastRemoteObject implements RPCService {

public RPCServiceImpl() throws RemoteException {

counter = 0;

super();

}

int counter;

@Override

Int add() throws RemoteException {

counter++;

return counter;

}

}

其中，在客户端代码中对RPC服务端进行了20000次调用，并统计了20000次调用的时间。

REST服务端代码：

@Path({"test "})

public class RestTest

{ @GET

@Path({"add"})

public int doGet(

)

{

counter++;

return counter;

}

}

REST客户端代码：

URL url = new URL("http:// 10.163.14.31:8080/test/add");

URLConnection rulConnection = url.openConnection();

HttpURLConnection httpUrlConnection = (HttpURLConnection) rulConnection;

httpUrlConnection.setRequestMethod("GET");

httpUrlConnection.setDoOutput(true);

httpUrlConnection.setDoInput(true);

httpUrlConnection.setUseCaches(false);

java.io.EOFException)

httpUrlConnection.setRequestProperty("Content-type", "application/x-java-serialized-object");

httpUrlConnection.connect();

OutputStream outStrm = httpUrlConnection.getOutputStream();

int counter;

long start = System.currentTimeMillis();

for(int I;I <= 20000;i++){

ObjectOutputStream objOutputStrm = new ObjectOutputStream(outStrm);

objOutputStrm.writeObject();

objOutputStm.flush();

objOutputStm.close();

}

long end = System.currentTimeMillis();

System.out.println("Took : " + ((end - start) / 1000));

最终测试结果如表3.1所示，其中RPC和REST均交换机器再次测试了一次，由表可以看出，以这种方式实现的RPC比REST的性能稍好，但是两者的差距并不是很大。

表3.1 RPC和REST性能测试

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **测试次数** | **总用时** | **平均时间** |
| RPC | 20000 | 18.860s | 0.943ms |
| REST | 20000 | 20.076s | 1.0038ms |
| RPC | 20000 | 17.779s | 0.88895ms |
| REST | 20000 | 20.430s | 1.0215ms |

因此，我建议，在通常的微服务开发中，选择REST风格的模式，这样更利于开发和维护；而在有低延迟需求的微服务开发中选用RPC来做通信。在本文所介绍的统一分析运行平台中，低延迟并不是我们所要保证的重点内容，因此我们选择更为规范的REST作为我们服务间的交互方式，同时选择json作为交互内容的载体。

## 3.3 微服务模块划分

为了实现业务编排、上下文和task管理的功能性需求，将系统的模块划分：任务模板模块、任务实例模块、调度引擎模块、Ignite模块和Eureka模块。

### 3.3.1 任务模板模块

任务模板模块负责对任务模板的管理，正如前面需求分析中提到的需求，我们再任务模板模块中实现这些需求：创建任务模板、修改任务模板、删除任务模板、获取全部任务模板、单独查询任务模板。

该模块对外开放这些rest接口作为交互的方式：

# 致 谢

值此论文完成之际，谨向所有关心和支持我的人们致以诚挚的谢意！

首先，我要衷心地感谢我的校内导师刘奇其教授和企业导师薄宏剑工程师。从论文选题、内容和整体结构的确定，到直至最后定稿，两位导师都以极其负责的态度给予悉心指导，为我提出了许多宝贵的意见和建议，使我获益良多。导师有严肃的科学态度，严谨的治学精神和精益求精的工作作风，这些都是我所需要学习的，感谢导师给予了我这样一个学习机会，谢谢!在此，谨向导师致以最诚挚的感谢！此外，还要感谢在企业实习时团队的成员。没有他们的帮助和共同努力，就没有项目的圆满成功，也就不会有本文的形成。在此，向他们表示衷心的感谢!

# 参考文献

1. 陈春霞. 基于容器的微服务架构的浅析[J]. 信息系统工程, 2016 (3): 95-96.
2. Fielding R T. Architectural styles and the design of network-based software architectures[D]. University of California, Irvine, 2000.
3. 唐文宇. 面向 SOA 架构微服务的安全系统的设计与实现[D]. 南京大学, 2016.
4. Abrams S, Kunze J, Loy D. An emergent micro-services approach to digital curation infrastructure[J]. International Journal of Digital Curation, 2010, 5(1): 172-186.
5. Namiot D, Sneps-Sneppe M. On micro-services architecture[J]. International Journal of Open Information Technologies, 2014, 2(9).
6. 徐鹏, 陈思, 苏森. 互联网应用 PaaS 平台体系结构[J]. 北京邮电大学学报, 2012, 35(1): 120-124.
7. Newman S. Building microservices[M]. " O'Reilly Media, Inc.", 2015.
8. 王克苑, 张维勇, 王建新. SSL 安全性分析研究[J]. 合肥工业大学学报: 自然科学版, 2004, 27(1): 87-91.
9. 王珊, 王会举, 覃雄派, 等. 架构大数据: 挑战, 现状与展望[J]. 计算机学报, 2011, 34(10): 1741-1752.
10. Brooks F P, Li Q. 人月神话[M]. 中国电力出版社, 2003.