国家电网公司微应用平台架构设计与实现*

冯 扬[1] 董爱强[1] 夏元轶[2] 石 超[1] 王琰洁[1] 王小平[4]
(1.北京中电普华信息技术有限公司 北京 100192; 2.国网江苏省电力公司信息通信分公司 南京 210000)

摘 要: 为了适应国家电网公司信息系统建设从传统架构模式向"大平台+微应用"架构模式转换,运用当前先进的技术和架构设计理念,针对传统单体式架构存在的问题,凭借其微应用自身特有的优势,从功能架构、技术架构、运行架构方面提出了设计方案。并在UAP微应用项目进行了实际部署和实现,举例给出了应用效果,通过测试和数据分析得出了微应用架构便于开发、运行、部署的结果。作为新兴事物,微应用在技术和管理上还有很多不足和需要改进的地方,这些将是下一步工作的方向。

关键词: 国网公司微应用平台;架构设计;UAP统一应用开发平台;实现与部署
中图分类号: TN915.09 **文献标识码**: A **国家标准学科分类代码**: 520

State grid micro-application architecture design and realize

Feng Yang[1] Dong Aiqiang[1] Xia Yuanyi[2] Shi Chao[1] Wang Yanjie[1] Wang Xiaoping[1]
(1. State Grid Electric Power Research Institute Beijing China-Power Information Technology Co. Ltd, Beijing 100192 China;
2. State Grid Jiangsu Information & Telecommunication Company, Nanjing 210000, China)

Abstract: In order to adapt to State Grid the information system from traditional architecture model to transform "large platform + micro application" architecture, used the current advanced technology and architecture design idea proposed solutions, in view of the problems existing in the traditional architecture, with its own unique advantages, from the functional architecture, technical architecture, running architecture provide a design and resolvent. And actually carried out the deployment and implementation in UAP micro application project, provide an example of application effect. Through test and data analyze, obtain the results, that the micro application architecture is convenient of development, operation, and deployment. As a new thing, micro application in technology and management, there are still many problems and need to improve, these will be the next step in the direction of the work.

Keywords: state grid micro application platform; architecture design; UAP micro application program; implementation and deployment

1 引 言

国家电网公司(以下简称"公司")的"十三五"信息化规划指出信息通信建设工作应以"厚云薄端"为导向,积极引入云计算、大数据等先进技术,构建流程标准化、应用组件化和场景多样化的信息化操作管理平台系统。

公司提出从传统单体架构模式向一个平台多种场景的"大平台+微应用"架构模式转变,"微应用"在这种大环境下产生,是呈现给业务人员的直接操作软件界面,满足用户业务需求、提高用户操作体验的功能组合,最终在PC、移动设备、大屏等各类终端的客户端容器中运行。该模式融合了多种当前先进的技术,同时也能满足公司软硬件资源系统统一管理、便捷维护的需要。

2 背景介绍

2.1 项目背景

公司微应用平台是基于"平台即服务"的云计算关键技术,对"大平台+微应用"的架构模型、容器化技术、服务化技术、自动化部署、运维和监控等相关关键技术进行研究,提出解决方案,解决了国家电网传统单体架构应用庞大而带来的开发难度大,部署、运维复杂,无法快速自动弹性伸缩、难以快速响应用户需求和成果持续交付困难等问题。通过本项目的研究,能够有效支撑公司在"大平台+微应用"架构下的信息化建设工作,提升技术实现水平、运营管理水平和服务水平。

收稿日期:2016-10
*基金项目:国家电网公司科技项目(52680116000400K0000000)资助

2.2 技术背景

“大平台+微应用”的架构模式,可以实现构建小核心、大外围的软件系统。其中“大平台”重点解决应用间的标准统一、差异屏蔽、组件复用,核心、复杂的业务模块以组件形式封装到平台中,将以往复杂的前端流程更多转移到了后端。而将大型软件系统拆解为“微应用”小系统,便于单个功能或模块的独立使用,同时为开发小型系统,提供更灵活、更个性化、更快速响应、更易于部署、更具可扩展性的服务。

2.3 国内外研究发展情况

微应用离不开微服务,微应用中包含大量的微服务。微服务,是一些协同工作的小而独立的服务,关于微服务的起源最著名的微服务架构原理就是康威定律(Conway's Law),Melvin Conway 在 1968 年指出:设计系统的组织,其产生的设计和架构等价于组织间的沟通结构。Dan North 对此还补充说这些系统在建成之后反过来还会约束和限制组织的改变[1]。

微服务中架构设计中的基本原理,源自于面向对象的设计思想。关注每个微服务的单一职责,关注微服务之间业务逻辑的相互分离,设计模块化或服务化,将大的系统架构按标准拆分。

互联网 web 时代的到来,开始越来越重视用户体验的重要。架构设计师也在采用将 UI 端(前端)简化(弱化),而把更多的逻辑和计算放在服务端,服务端程序处理几乎所有的工作,然后把展示页面发送给客户端,所以弱化了客户端的要求和限制。给用户使用带来的方便,强化了的后端提供更加系统强大的支持。

2.4 问题分析

传统架构是单体式的,相对稳定,但是在面对系统功能、性能和业务需求的更新,应对起来显得力不从心。主要存在问题如下。

1)系统扩展困难

随着业务规模的不断增长,业务系统需要满足高并发请求以及海量数据交互要求。传统的企业应用架构模式下,大多通过垂直方式复制应用程序与数据,以解决容量和性能问题。因此业务系统在上线前,会根据最大在线用户数和并发用户数的预估,购买相应的硬件。当随着业务量的增加,出现各种瓶颈时,只能通过升级硬件或增加硬件和扩容等物理方式解决,系统扩展成本较高。

2)持续交付问题

传统应用架构模式下,业务模块之间的循环依赖、重复 API 定义、冗长复杂的业务流程等问题对新业务的上线造成极大困难。而且传统业务应用的“体量”较大,构建、部署和启动时间较长,配置较复杂,在开发环境中的效果与运行环境中的效果很难一致,不利于快速上线、频繁部署的应用场景,阻碍了持续交付。

3)运维难度大

传统企业架构的应用在运维阶段,某功能组件出现无法恢复的故障时,整个节点将处于不可用状态。此时业务应用的功能越复杂,影响面则越大,即使一个小问题都会导致大面积的不良影响。

3 微应用架构设计

3.1 微应用架构设计要求

针对传统架构设计的缺点和问题,微应用架构设计模式需要考虑的设计内容和设计要求如下[2]。

1)高内聚松耦合

所谓高内聚是指微应用内部单一责任,内部关系紧密,依赖性强。简化服务之间接口的复杂性,调用方式和传递方式,减少服务与服务之间的依赖,服务于服务之间相互独立,这样有利于修改和组合。

2)去中心化

去中心化是微应用的找一个最显著的特点。在整个系统中每个服务都具有高度独立自治的特点。服务与服务之间可以自由连接,每个服务都有可能成为阶段性的控制中心。每个微应用之间形成开放式、扁平化、平等的系统结构或关系。微应用平台不是绝对的没有中心,完全平等,而是没有固定的中心,服务之间组合灵活,控制关系弱化。

3)敏捷交付灵活部署

每个微应用都可以由较小规模的研发团队负责设计、开发、测试、部署、运行治理及灰度发布等,是实现开发运维一体化的基础。

微应用可以按独立进程去部署,即可以将多个相同的微应用部署到不同的服务器上,也可以在一台服务器上部署多个微应用实例,具备高可靠的水平扩展能力。如果是云端部署则可以利用轻量级的虚拟机容器进行部署,有效的降低部署成本,提高资源利用率。

3.2 功能架构

微应用平台框架用一组应用的方式来构建一个业务系统,每个微应用独立部署在不同的进程中,不同的微应用通过一些轻量级交互机制进行通信,微应用可独立扩展伸缩,不同的微应用可由独立的团队来维护,满足了业务系统快速开发及部署的需求,增强了开发平台的支撑性。公司微应用框架功能架构如图 1 所示。

功能架构设计包括 5 个组成部分。包括应用开发套件,基础服务框架,部署监控组件,集成开发工具,对开发人员提供了,外部集成接口,为第三方用户提供服务。

3.3 技术架构

微应用平台框架技术架构分为 3 层,分别是展现层,应用层,持久化层,如图 2 所示。

1)展现层

微应用平台框架展现层基于 Jquery 构建,支持 Bootstrap 响应式布局,采用 Knockout 支持 MVVM,图表

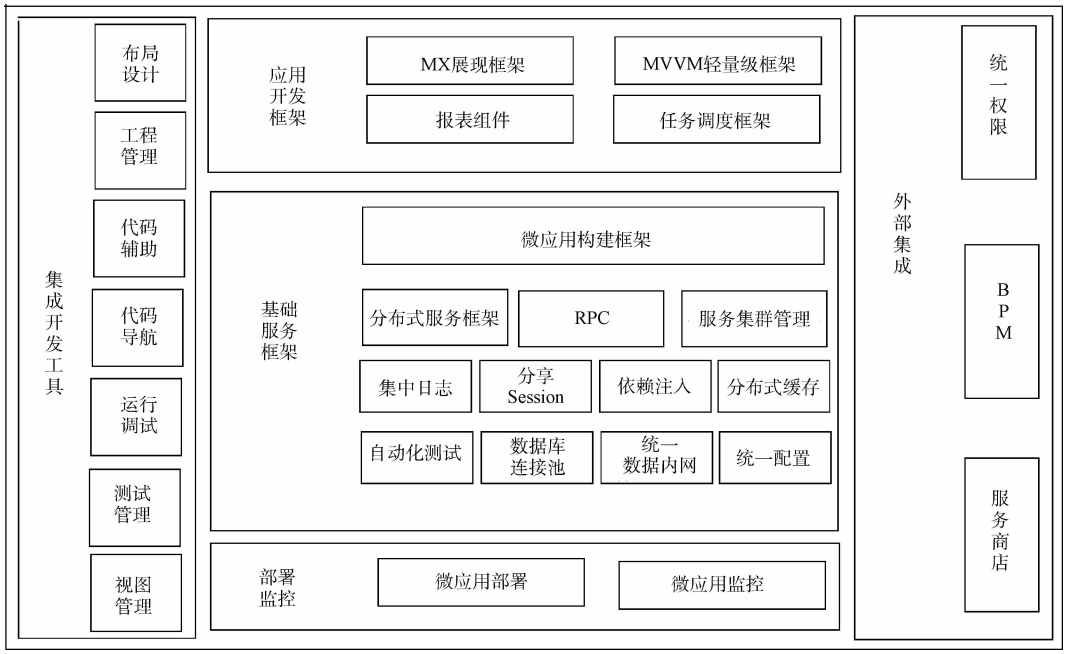


图 1　微应用功能架构

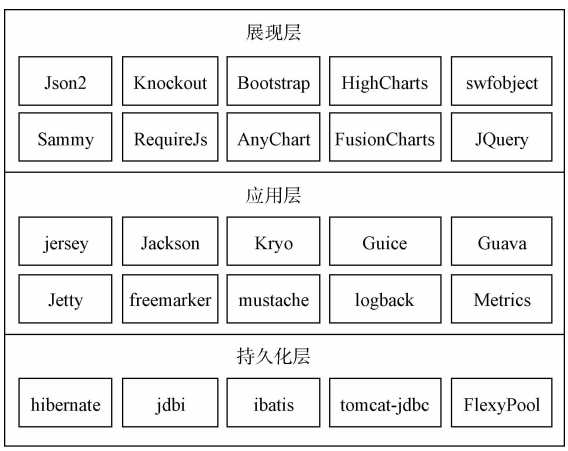


图 2　微应用技术架构

支持 AnyChart、FusionCharts、HighCharts，采用 RequireJs 支持异步加载，采用 swfobject 支持嵌入 Flash 播放器，采用 Json2 操作 JSON 对象，采用 Sammy 支持客户端 URL 路由。

2)应用层

微应用平台框架应用层采用嵌入应用中间件 Jetty 到微应用的方式以简化应用配置和发布，采用 jersey 作为 RESTful 服务框架，采用 Jackson 实现操作 JSON 对象，采用 Kryo 序列化 RPC 对象，采用 Guice 以支持 IOC 及 AOP，采用 Guava 简化 JDK 应用程序接口的开发，视图模板支持 freemarker 及 mustache，采用 logback 作为日志输出组件，采用 Metrics 度量各项运行指标。

3)持久化层

微应用平台框架持久化层支持 hibernate，jdbi，ibatis，数据库连接池采用 tomcat-jdbc，数据库连接池管理采用 FlexyPool。

4)技术分析

相对传统架构，微应用平台架构采用的技术是轻量级的，具有单一职责，实现简化等特点。改变了传统技术大而全的模式。接下来对微应用常用到的技术进行归类和分析。

(1)轻量级嵌入式容器

微应用平台提供了轻量级嵌入式应用容器，容器中封装了 Web 应用服务器、服务注册发现、轻量级 RPC 和服务调用的负载均衡策略等功能组件。在开发阶段根据微应用框架 Bundle 特性，能够实现 SG-UAP 平台各模块的 Bundle 插件集成到微应用中，如任务调度、大数据组件、报表和展现框架等[3]。

(2)轻量级嵌入式组件

微应用平台引入了性能高效的 Guice 框架作为微应用中的 IOC 组件。使用 IOC 组件可以有效解除对象创建时类之间的强耦合性，避免修改和维护应用的繁琐，并能解决业务应用运行效率低下等问题。

(3)服务注册和访问

微应用平台服务的注册、刷新和发现功能都是在本地执行的，所以集群中每个节点都有各自的服务注册中心，其内部通过 Gin 框架提供 Restful 服务，节点启动注册中心，从而对外提供服务注册、服务刷新、服务发现 3 个服务。首先微服务调用服务注册器进行本地注册，注册成功后，微服务再去调用服务发现器，最后微服务启动定时器，

周期性的调用服务刷新器,来完成服务注册和访问。

(4)远程调用

微应用平台远程调用(RPC)包括客户端和服务端,分别为服务的调用方与服务的提供方。一次 RPC 调用完成,RPC 将原本的本地调用转变为调用远端的服务器上的方法,给系统的处理能力和吞吐量带来了大幅度的提升。平台实现基于 HTTP 协议的 RPC 远程调用,通过远程调用让服务调用更加简单、透明。

3.4 Gossip 算法分析

Gossip 字面意思是流言传播。Gossip 算法又被称为反熵(anti-entropy),熵是物理学上的一个概念,代表杂乱无章,而反熵就是在杂乱无章中寻求一致,Gossip 算法本质就是在一个有界网络中,每个节点都随机地与其他节点通信,经过一番杂乱无章的通信,最终所有节点的状态都会达成一致,节点之间完全对等,不需要任何中心节点。是一种去中心化、容错性强而又最终形成一致的算法。Gossip 算法常见于大规模、无中心的网络系统,可以用于众多能接受"最终一致性"的领域:失败检测、路由同步、Pub/Sub、动态负载均衡。

3.4.1 算法关键点

1)协调与同步

协调机制是每次两个节点通信时,如何选择交换数据能最快达到一致性。Gossip 算法能够有效实现有限网络负载的情况下,节点通信的弹性扩展,和快速收敛。信息同步也是 Gossip 算法的关键点,比如 Merkle tree(MT)结构,就是一个能偶快速同步的 hash 算法。

2)信息交换

网络中所有参与交换的节点称为为 peer,信息发送节点成为 p,信息接收节点称为为 q。为了实现所有节点之间最终对等,一致的通信。信息交互的两个节点之间有 3 种不同的信息交换方法。

(1)push-gossip:最简单的情况下,一个节点 p 向 q 发送整个 GlobalMap 信息。

(2)pull-gossip:p 向 q 发送 digest,q 根据 digest 向 p 发送 p 过期的 (key,(value, version)) 列表。

(3)push&pull-gossip 与 pull-gossip 类似,只是多了一步,p 再将本地比 q 新的数据推送给 q,q 更新本地。

如果把两个节点数据同步一次定义为一个周期,则在一个周期内,push 需通信 1 次,pull 需 2 次,push/pull 则需 3 次,从效果上来讲,push/pull 最好,理论上一个周期内可以使两个节点完全一致。push/pull 的收敛速度是最快的。

假设某个节点在第 i 个周期被感染的概率为 p_i,第 $i+1$ 个周期被感染的概率为 p_{i+1},则 pull 的方式:

$$p_{i+1} = p_i^2 \tag{1}$$

而 push 为:

$$p_{i+1} = p_i\left(1-\frac{1}{n}\right)^{n(1-p_i)} \tag{2}$$

显然 pull 的收敛速度大于 push,而每个节点在每个周期被感染的概率都是固定的 $p(0<p<1)$,因此 Gossip 算法是基于 p 的平方收敛,也成为概率收敛,这是一种有效易于实现的一致收敛算法。

3.4.2 算法设计

Gossip 算法的目的是节省传输时间和提升通信效率。在网络中任意选择一个节点作为信息的发起者,通过第 n 周期随机选择传播给附近的 k 个邻居节点,通过第 N 周期所有的节点都能平等的接收到了信息。最终实现一致收敛。

完成第 n 周期传播过程,最少需要的信息量定义为 $f(n,k)$[4]。

$$f(n,k)=\begin{cases}\left\lceil\frac{n-k}{k-1}\right\rceil+\left\lceil\frac{n}{k}\right\rceil, & 1\leqslant n\leqslant k^2\\ 2\left\lceil\frac{n-k}{k-1}\right\rceil, & n\geqslant k^3\end{cases} \tag{3}$$

3.5 运行架构

微应用平台在运行时分为 3 个层次,运行架构如图 3 所示。

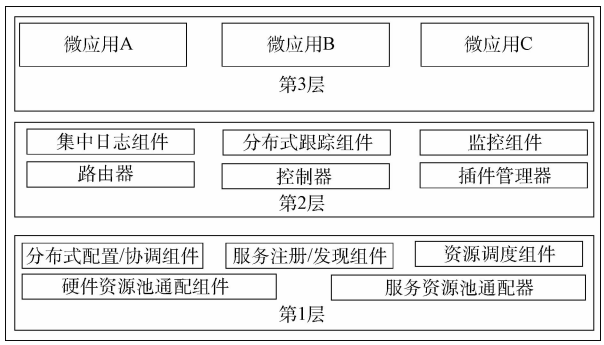


图 3　微应用平台运行框架

第 1 层是核心层,对运行时服务部署的软硬件环境进行抽象隔离,是第 2 层组件运行的基础。第 2 层是功能层,为业务系统提供提供通用功能组件。基于平台构建的业务系统位于第 3 层,包括不同微应用 A,微应用 B,微应用 C…。第 2 层和第 3 层组件发布及集成方式相同,这样的运行架构便于业务系统将自己的通用组件方式添加到微应用平台的货架中,也便于业务系统间灵活组合集成。

4 实现与部署

4.1 UAP 统一应用开发平台

UAP 统一应用开发平台是微应用架构设计的实际落地运用,积累了丰富组件和外部接口,扩展兼容性强,从而实现国网对开发平台统一技术标准、专注业务发展、降低建设成本的建设要求。

4.2 微应用实现

UAP 统一应用开发平台实现了微应用架构的设计理念,该平台主要依赖微服务的使用来实现微应用模式下的统一开

发和部署，具体实现流程包括以下几个过程，如图 4 所示。

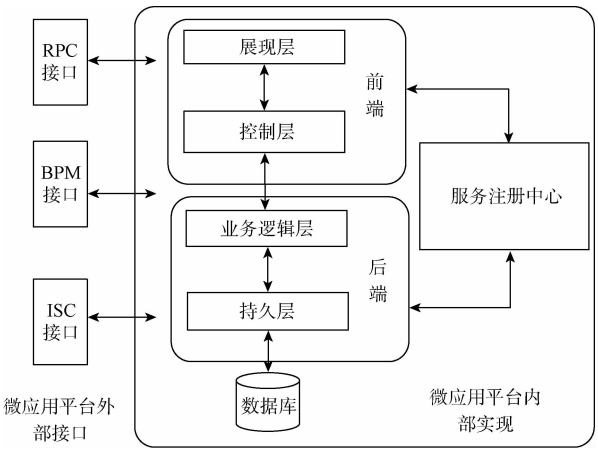


图 4　微应用平台实现流程

微应用实现流程包括横向和纵向两部分，纵向流程包括以下内容。

1）控制层接收来自展现层的 API 业务请求。

2）业务逻辑层作为控制层和持久层的过度，实现事务封装，业务逻辑的封装。

3）持久层实现对数据库的操作。数据库包括 Hibernet 等。

微应用平台横向流程包括平台的分层结构（前端（客户端）、后端（服务端））、服务注册中心以及为第三方应用提供可插拔接口。

服务注册中心，主要实现前端和后端服务的注册对应。包括 IOC 序列化，反序列化。Json 和业务对象的转换。IOC 依赖注入。

Rest 服务调用过程如下，首先后端在服务中心注册信息，接下来前端在服务中心发现后端的注册信息，在注册列表中查询匹配，形成依赖关系。通过注册中心实现前端与后端远程交互。[5]

第三方辅助模块根据实际应用场景的不同而选择性搭配使用，包括远程（RPC）调用，组件流（BPM），权限控制（ISC）。

4.3　微应用部署

UAP 统一应用开发平台能够支持大量微应用，能够有效实现集群部署。集群部署通过 UAP 微应用平台将微应用框架和软负载均衡器、运行时容器、对等服务发现组件等功能有机结合，实现海量微应用系统的自动集群部署。[6]图 5 是基于微应用平台架构实现的 UAP 系统的集群逻辑部署图。

集群部署步骤如下：

1）微应用系统在启动后，会自动将自身的服务注册到对等发现组件（由平台提供），微应用将注册服务同步到软负载均衡器中。

2）对等发现组件使用 Gossip 协议实现去中心化的集群，当集群中的一个节点出现异常时，其他节点能够立刻感知到该变化，通知软负载均衡器去掉异常节点，并通知 UAP 运行时容器的监控中心。

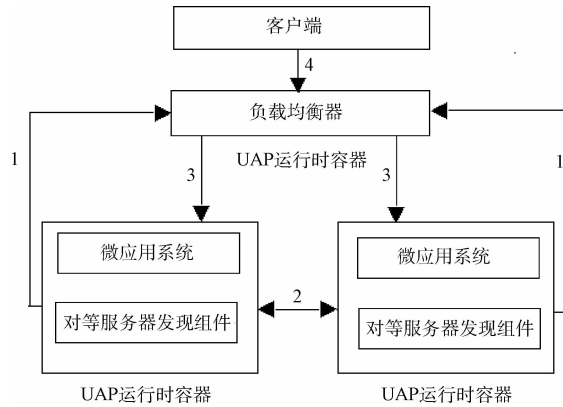


图 5　集群逻辑部署

3）软负载均衡器借助对等发现组件实现新节点的动态配置，当有新节点加入时，自动配置新节点配置信息。

4）客户端通过软负载均衡器进行请求的分发，平台提供基于粘性 session、无状态 session 等方案以兼容传统开发方式，完成部署。

4.4　微应用实例

1）代码结构

一个典型的微应用业务功能的代码结构包括：服务接口 API、前端微应用（frontApp）、后端微应用（backApp）。

前端微应用提供前端 UI 界面，发起 UI 请求；在 Controller 中响应请求并调用接口方法，实际调用后端服务方法来处理请求。

后端微应用实现接口方法，提供服务，响应 controller 的调用，进行业务处理。核心框架提供了 IOC、RPC、请求和响应的包装类，将其打成 jar 包，在 frontApp 、backApp 添加依赖。

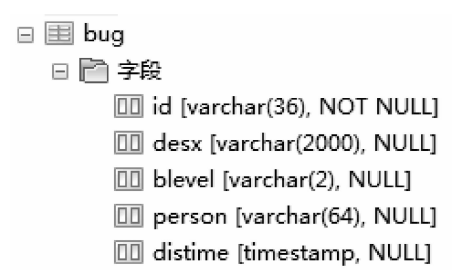
服务接口 API 后端微应用提供的服务，以接口方法的方式提供。从 frontApp 中调用，在 backApp 中实现。将其打成 jar 包，在 frontApp 、backApp 添加依赖。

2）创建实例

下文以 bug 表单创建为例介绍微应用实例创建过程。

（1）创建服务接口 API

在数据库中创建缺陷 bug 表。



创建 API 微应用,代码如下。

```
private String id;
private String desx;
private String blevel;
private String person;
private String distime;
/**虚拟主键*/
private String mxVirtualId ;
```

API 工程要打包成 fatjar 提供给前端工程和后端工程依赖使用。

(2)服务注册

启动服务中心,将 API 注册到服务中去。

(3)创建前端工程

创建名为 bugfront4test 的工程,存放资源文件(包括 js 图片、jsp 等),新建 rest 请求服务。将工程打包成 bugfront4test.jar。运行 java -jar bugfront4test.jar 启动前端工程。主要负责增、删、改、查页面渲染等功能。

(4)创建后端工程

创建,负责处理业务逻辑,数据库操作等功能。

(5)访问创建实例

通过浏览器访问一下地址:

http://127.0.0.1:8080/bug/index.html

实例创建结果展示如图 6 所示。

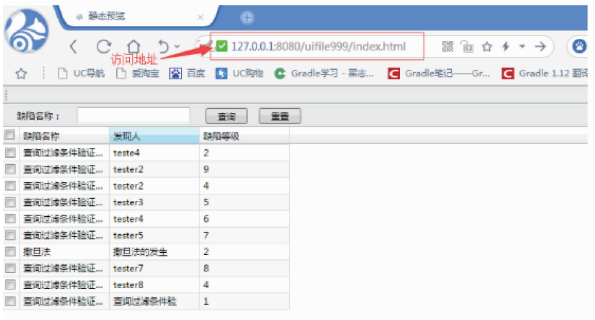


图 6　实例创建结果展示

5　结果和结论

5.1　数据分析

UAP 微应用平台通过了功能和性能测试,可以完成微应用的开发和部署。性能测试部分数据如下。

监控端 UAP 微应用平台客户端 CPU 的利用率、内存使用率、线程使用率、类使用率的使用情况监控结果截图,如下。

通过大量的数据分析可以得到这样的结果,同一系统(功能模块)在微应用平台和传统平台上运行的结果是不一样的。采用微应用架构后,因为服务调用量增加,导致显著提升 CPU 的利用率、内存使用率、线程使用率、类使用率。增加额资源的消耗。

表 1　UAP 微应用平台性能测试数据

用例	并发数	场景	平均响应时间/s	事务失败率/%
新增-非压缩模式	200	加压:一起 thinktime:忽略 持续时间:20 min 释放:一起	1.076	0
登录后加载首页	200	加压:一起 thinktime:忽略 持续时间:20 min 释放:一起	7.532	0
查询一条数据	500	加压:一起 thinktime:忽略 持续时间:20 min 释放:一起	0.596	0
新增页面,注释掉保存事务	200	加压:一起 thinktime:忽略 持续时间:20 min 释放:一起	48.475	0

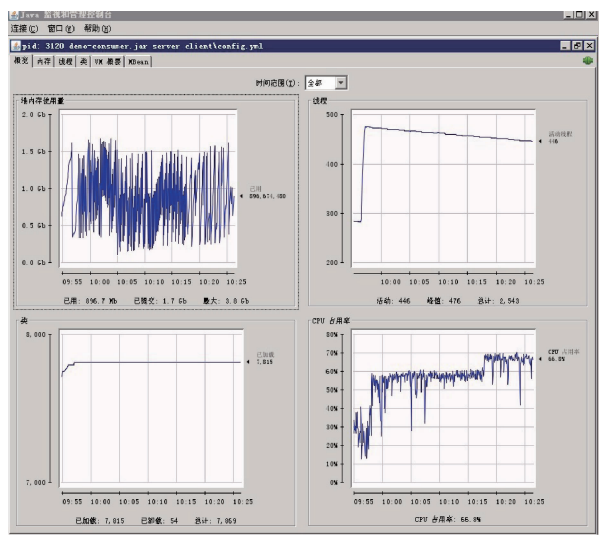


图 7　微应用平台客户端性能监控图

5.2　结果

微应用架构牺牲了资源消耗的同时,优点也是很明显的。针对本业务应用微应用实现了灵活开发、可拆分/组合部署、功能便捷升级和替换[7]。

1)设计开发灵活

单一职称模式开发,不需要传统开发的大而全的模式,设计、开发灵活。从前需要几十人甚至更多团队,几个月甚至更长时间开发一个大而全的系统,现在 3-5 个人几

周甚至更短时间就能开发完成。

2)部署方便

部署和联调通常是运维人员要花费很多时间主要关注的事情。而微应用采用的都是统一标准对外提供接口，因为微应用职责单一部署简单方便，通常是自动完成，减少了部署的人力和时间。

3)便捷升级和替换

单个微应用是外部松耦合的，升级和替换非常方便，不会出现传统模式下牵一发而动全身的局面。

综上，可以得出这样的结论，微应用不是降低能耗，而是提升了资源的利用率和开发使用等的灵活性。

5.3　结论

1)微应用模式

微应用平台的应用模式通常是分为把新的功能作为微应用模式开发，以及对传统业务系统进行微应用拆分改造[8]。

(1)微应用模式实现新功能

当传统应用已经变得难于管理的时候，不适于在原有的基础上继续扩建。如果有新的功能特性需要添加，不要在传统应用中添加代码，而要把新的代码放在另一个单独的微应用。

(2)传统业务的微应用改造

对传统业务的微应用改造包括两个方面：第一种方式是缩减原系统业务，比如将表现层组件从业务逻辑和数据访问层组件中分离出来。缩减传统应用的一个策略是将表现层从业务逻辑和数据访问层中分离出来，一个典型的传统应用一般包括 3 层：表现层、业务逻辑层和数据访问层。表现层逻辑与业务和数据访问逻辑之间通常有着明显的区分，业务逻辑层有一个粗粒度的 API 供表现层调用，所以可以分割为更小的微应用[9]。

第二种方式是在传统业务模块中拆分出单独的微应用。这个策略的目的是将传统应用中的模块，转变为单独的微应用。每次提取一个模块，改造为微应用，传统应用就慢慢缩减了。一个大型的复杂的传统应用，通常包含数十甚至上百个模块，如果都可以逐步被提取，那么就可以实现微服务的改造了。

5.4　不足和改进

微应用技术的优点在前面架构介绍和实现部署的介绍时已经讲得很充分了，架构灵活、部署便捷、易于重组和容易适应新架构等。伴随他的优点而来的还有他的一些不足。

1)微应用的不足

(1)过度累加

微应用设计的初衷是为了简化系统和便于部署，但通过一定时间的累加，代码量越来越大，代码之间关系也变得越来越复杂。从而一个简单的应用会因此变成了一个巨大的怪物。JAR 文件简化了，而之间的复杂的依赖关系，与微应用便捷使用的设计理念背道而驰。

(2)可靠性问题

“大平台＋微应用”的模式，把原有的复杂的前端模式，转移到后端，这种弱化前端，强化后端的模式，要求后端运行更加可靠。当多个微应用运行在一个进程中，任何一个模块中的一个 bug，比如内存泄露，将会有可能弄垮整个进程，让其他的模块和应用受影响，会影响到整个应用的可靠性。

(3)微应用架构迁移困难

传统结构单体式架构使得原系统更换新微应用架构和新语言非常困难。传统架构结构固定，开发语言固定，如果想改成微应用框架，无论是时间还是成本都是非常昂贵的，即使微应用框架更好。原有架构和原有语言难以更换，微应用的新的架构和新的语言难以实现。

(4)管理难度提高

与传统架构相比，微应用架构更加扁平化，没有中心节点，同时节点总数大幅几何级增加，给节点管理带来了很大难度。原来的负载均衡技术已经无法应对管理这样大规模的节点，新的架构下的这种规模的节点管理，需要引入新的管理技术。实现成本的降低，必然导致管理成本的提升。

2)微应用的改进

早期微应用通常是对传统应用一次性到位的分解/拆分，针对这种过度分解/拆分方式带来的过度累加、管理等方面的负面问题，我们提出了一定的改进办法，就是从前端到后端逐层分解/拆。微应用可以有以下 3 种拆分方式。从小到大拆分方式依次为。

(1)只包含界面交互部分。

(2)除了包括界面交互，还包括对已有服务的调用编排逻辑。

(3)包括界面交互、业务逻辑和数据存取 3 层，是一个完整可独立部署运行的应用。

6　结　　论

“大平台＋微应用”的架构模式对传统架构模式是一个本质改变，从大而稳定的模式向小而灵活的模式转化，单一职责和轻量级的设计可以实现对软件设计模式的根本性变革。让软件设计和使用达到最好的吻合[10]。

同时微应用也带来的管理、监控等方面的问题。不是所有领域都适合微应用，尤其在不了解一个领域的业务需求，服务界线不能清晰划分，组织结构不明确，无法对已有系统或内容分类时，不适合采用微应用技术和架构。微应用有很多优势，但也不可避免导致服务间的协作大幅增加。所以在大规模使用微应用之前，一定要熟悉架构和构建工具，这样才能真正使用好微应用技术。这也是我们下一阶段微应用研究工作的方向。

(下转第 63 页)

[6] 裴远寅,夏靖. MIMO 的研究与仿真[J]. 广东电脑与电讯,2008(5):41-43.
[7] 蒋学勤. MIMO 技术的演进[J]. 贵州科技工程职业学院学报,2007,2(2):39-41.
[8] 付卫红,杨小牛,刘乃安,等. 宽带无线通信中的 MIMO 系统[J]. 电子科技大学学报:社会科学版,2007,36(2):176-178.
[9] LEE B, CIOFFI J M, JAGANNATHAN S, et al. Binder MIMO channels[J]. IEEE Transactions on Communications,2007,55(8):1617-1628.
[10] GESBERT D, KOUNTOURIS M, HEATH R W, et al. Shifting the MIMO paradigm[J]. Signal Processing Magazine, IEEE,2007,24(5):36-46.

作者简介

刘芸,1992 年出生,南京电子技术研究所硕士研究生,天线设计。
E-mail: 1125116057@qq.com

孙红兵,1978 年出生,工学博士,南京电子技术研究所高级工程师,相控阵天线设计、微波射频电路设计。
E-mail: hongbingsun@163.com

(上接第 58 页)

参考文献

[1] NEWMAN S. 崔力强,张俊(译). 微服务设计[M]. 北京:人民邮电出版社,2016:504-509.
[2] 王磊. 微服务架构与实践[M]. 北京:电子工业出版社, 2015: 13-38.
[3] 汪凯,张功萱,周秀敏. 基于容器虚拟化技术研究[J]. 计算机技术与发展,2015,25(8):138-141.
[4] 刘德辉,尹刚,王怀明. 分布式环境下的 Gossip 算法综述[J]. 中国计算机科学,2010,37(11):24-31.
[5] 李勇. 分布式 Web 服务发现机制研究[D]. 北京: 北京邮电大学, 2007:35-42.
[6] 李忠民、齐占新. 业务架构的微应用化与技术架构的微服务化——兼谈微服务架构的实施实践[J]. 科技创新与应用,2016(35):95-96.
[7] 李晓珍,刘迪,王孟强,等. 微应用架构下分布式事务的处理方法[C]. 2016 电力行业信息化年会论文集,2016:356-359.
[8] 鲁宗相,王彩霞,闵勇,等. 微电网研究综述[J]. 电力系统自动化, 2007,31(19):100-105.
[9] 李林锋. 分布式服务框架原理与实践[M]. 北京:电子工业出版社,2016:17-39.
[10] X/Open company Limited. Distributed transaction processing: The XA specification[S/OL](1991-12-29)[2016-06-24] http://pubs.opengroup.org/onlinepubs/009680699/toc.Pdf.

作者简介

冯扬,1980 年出生,工程师,硕士,研究方向为计算机、网络信息安全。
E-mail:fengyang@sgitg.sgcc.com.cn

董爱强,1979 年出生,工程师,本科,研究方向为电力企业信息化建设、信息集成。
E-mail:dongaiqiang@sgitg.sgcc.com.cn

夏元轶,1988 年出生,工程师,硕士,研究方向为信息安全。
E-mail:15951892168@163.com