基于 OSGi 的分布式 Web 应用结构①

王宇飞, 朱 伟, 刘 丹

(南京南瑞集团公司 北京中电普华信息技术有限公司, 北京 100192)

摘 要: 通常一个 Web 应用系统由多个功能模块组成, 但不同功能模块面临的压力不一样,以致各模块对资源的需求不同. 对 Web 应用系统进行性能扩展普遍采用集群方式, 集群的粒度是整个 Web 应用系统, 而不能对应用内部某些负载压力较大的功能模块进行特殊处理. 设计了一种基于 OSGi 的分布式 Web 应用结构, 可以以模块为粒度进行部署, 按照模块的实际负载情况调整部署结构. 该分布式结构基于 OSGi 规范, 把 Web 应用拆分成多个模块, 以 RFC119 为标准实现分布式节点之间服务交互. 这种结构将更加有效的利用系统资源, 节约应用部署成本. 实验表明, 该研究成果可优化大型 Web 应用结构的设计, 并且随着应用模块数量的增多, 系统并发数的增加, 采用该分布式 Web 应用结构的优势越明显.

关键词: OSGi; Web 应用; 分布式; 模块

Distributed Web Application Structure Based on OSGi

WANG Yu-Fei, ZHU Wei, LIU Dan

(Beijing China Power Information Technology Co. Ltd, NARI Group Corporation, Beijing 100192, China)

Abstract: The pressure of different functional modules in Web application system is different; the demand for resources of each module is different. The traditional way of cluster size in the cluster can only be applied for the entire cluster, rather than some internal function modules can apply for special treatment. This paper presents a distributed OSGi-based Web application structure, module size can be deployed in accordance with the actual load adjustment module deployment structure. This distributed architecture is based on the OSGi specification, the Web application into multiple modules to RFC119 standard for distributed service interactions between nodes. This structure will be more efficient use of system resources, saving application deployment costs. Experiments show that the research results can be optimized for a large Web application architecture design, and with the increasing number of application modules, the system increases the number of concurrent, distributed Web applications using the structure of the more obvious advantages.

Key words: OSGi; web application; distributed; module

在大数据、大并发的计算机应用时代, Web 应用系统面对的负载压力越来越大, 在软件本身性能达到瓶颈的时候, 传统做法是采用增加集群服务器数量的方式, 将所有的功能模块部署在所有节点上来缓解应用的负载压力.

实际应用中, 应用系统的不同功能模块面临的压力不一样, 各个模块对资源的需求不同. 通常一个应用系统中多数功能模块只有少数用户使用, 即使在使用高峰期负载也极小, 在这种情况下, 如果 Web 应用能够以模块为单位进行分布式部署, 就能根据各个模块的负载情况来调整模块的部署数量, 实现负载较小模块部署在较少的节点上, 负载较大模块部署在更多的节点上, 以此实现在节约硬件资源的情况下, 保证应用系统应有的负载能力. 与此同时, 在分布式架构下, 模块的更新和维护影响的对系统整体的影响更少, 有利于提升系统的动态性.

传统 Web 应用, 所有的应用程序都集中存放在一起, 比如 class 文件存放在 WEB-INF/classes 下, jsp 存

① 收稿时间:2014-11-04 收到修改稿时间:2014-12-12

放在 WEB-INF/jsp 目录下. 无法抽取其中一个功能模块单独部署. OSGi 框架的出现, 使得能够把应用拆分成一个个 Bundle 存放, 这样每个 Bundle 的应用程序在物理上相互隔离, 每个功能模块都由若干个 Bundle 组成, 并能够进行单独部署. 这样就为 Web 应用进行以功能模块为粒度的分布式部署创造了条件.

但是基于 OSGi 来实现功能模块的分布式部署, 会产生其它的技术难点需解决:

1) 外部请求访问应用时, 如何把请求转发到具体的分布式节点?

2) 分布式部署的功能模块之间也存在服务调用, 如何透明地将原始的进程内部调用方式转化为分布式节点之间的服务调用?

本文将致力解决上述问题及技术难点.

1 相关技术

1.1 OSGi

OSGi [1](Open Service Gateway Initiative), 一个基于 Java 语言的服务(业务)规范, 定义了一个优雅、完整和动态的组件模型. OSGi 环境的应用程序都是以 Bundle 的形式存在, 各个 Bundle 之间物理隔离. Bundle 可以在系统运行期被远程安装、启动、升级和卸载(其中 Java 包/类的管理被详细定义), 每个 Bundle 都可以被单独管理.

Bundle 之间的依赖分为 2 种方式, 一种是通过 Export 和 Import 包的方式. 另一种是通过 OSGi 服务的方式, 一个 Bundle 作为服务提供者, 对外提供服务, 使用者可以查找到这个服务, 并使用这个服务.

1.2 RFC119

OSGi 联盟在发展过程中为了满足 OSGi 的分布式的扩展, 制定了 RFC119[2]规范, 其目的是给出一个通用的解决方案以支持最基本的分布式处理特性和功能, 包括服务发现和与外部环境的交互, 并使得熟悉 OSGi 的开发人员不需要太多额外的学习就可以轻松的开发分布式 OSGi 应用. 为有效支持分布式处理, RFC119 规范对 OSGi 框架和编程模型进行了相应的改变, 在 OSGi 框架中增加了服务钩子能力以达到对服务使用信息的监控; 在编程模型上, 增加远程服务属性标识和配置分布式应用所需要的元信息; 在 OSGi 框架上引入了两个额外的系统级模块: DSW(Distribution Software)和 Discovery Service, 其中, DSW 是一个软件实体, 通过使用多种已有的分布式软件系统, 使得 OSGi 平台能够支持对其他地址空间或其他机器服务的绑定和注入; Discovery Service 也是一个软件实体, 通过使用多种已有的服务发现系统, 使得 OSGi 平台能够支持跨地址空间、跨机器节点服务的注册和查找.

1.3 Web 应用

Web 应用是 Web 服务器的动态扩展, Web 应用由 Web 构件(包括 Servlet 和 JSP)、Bean 构件、静态资源文件(例如: HTML、JS、CSS、图片、Flash 等)以及客户端 Applets 等元素构成. Web 应用一般为 B/S 结构, 就是基于浏览器/服务端的结构, 用户只需要有浏览器就能访问应用程序, 而不需要安装别的客户端软件.

2 分布式Web应用结构

2.1 基于 OSGi 的 Web 应用技术路线

假设一个应用有 A、B、C、D 四个功能模块和节点 1、节点 2、节点 3、节点 4 四个节点.

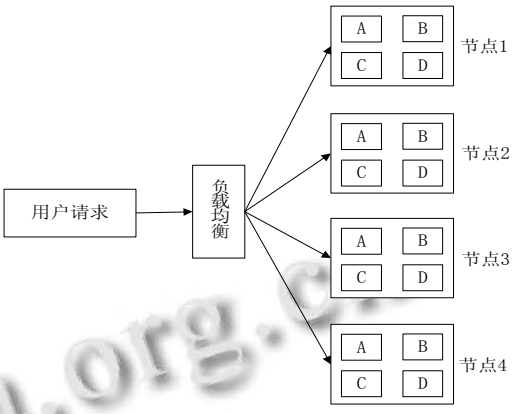

图 1 传统 Web 应用集群部署图

传统集群部署的 Web 应用, 会在每个节点上都部署一套完整的应用, 如图 1 所示. 当一个请求访问应用程序的时候, 首先会访问代理节点. 代理节点根据一定的负载均衡策略把请求转发到具体处理的节点, 由具体的节点进行处理并返回结果.

本文所述的分布式 Web 应用基于 OSGi 规范, 采用在 Servlet 容器中嵌入 OSGi 框架的方式实现[3]. 如图 2 所示, Web 应用在启动的时候会初始化桥接 Servlet, 桥接 Servlet 初始化时会启动 OSGi 框架. OSGi 框架负责启动框架内的各个功能模块. 功能模块在启动的时候会注册动态 Web 资源(Servlet、Listener、Filter、JSP 等)和静态 Web 资源(HTML、JavaScript 等). 各个功能

模块启动完成后，整个 Web 应用启动完成.

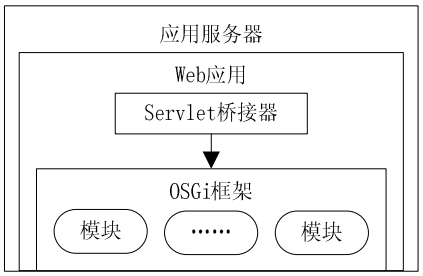

图 2 Web 应用模块化结构

基于 OSGi 的 Web 应用，各个功能模块在物理上隔离，具备了分布式部署的条件.

2.2 分布式 Web 应用结构

基于图 1 中的 Web 应用，假定 A 模块的负载最大(并发请求 400)，C 模块的负载次之(并发请求 200)，B 和 D 模块的负载最小(并发请求 100)，每个模块运行占用内存 100M，每个节点支持的并发能力为 200. 传统解决方案按照图 1 的结构，将每个模块进行集群部署，为了解决负载最大的 A 模块和负载次之的 C 模块，而将 B 和 D 模块进行重复部署额外占用内存空间.

根据四个模块的不同负载情况，采用分布式的 Web 应用结构能够更合理利用资源. 基于上述假定，对 A、B、C、D 模块采用图 3 方式进行分布式部署，每个模块的并发请求能力是 100，每节点的并发为 200，在满足用户请求和节点并发能力要求下，四个节点各减少 200M 内存，共计 800M 内存. 当 Web 应用功能模块足够多时，资源利用率更加明显.

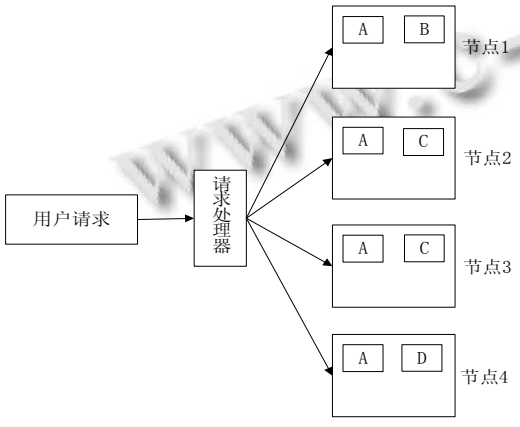

图 3 基于 OSGi 的分布式 Web 应用部署图

这种应用结构在一定程度上能节约磁盘和内存资源，节约的资源可以用来按需部署功能模块，或者为其它应用程序及系统进程服务.

2.3 请求处理

随着 Web 应用结构的改变，应用的请求方式也与传统方式有所不同. 传统方式下，基于 F5 硬件或者 Nginx 软件等成熟的负载均衡解决方案即可实现用户请求到功能模块的处理，而在分布式结构下，外部请求访问应用时，将用户请求转发到具体的分布式节点需要对 URL 特殊处理及请求计数等方式实现.

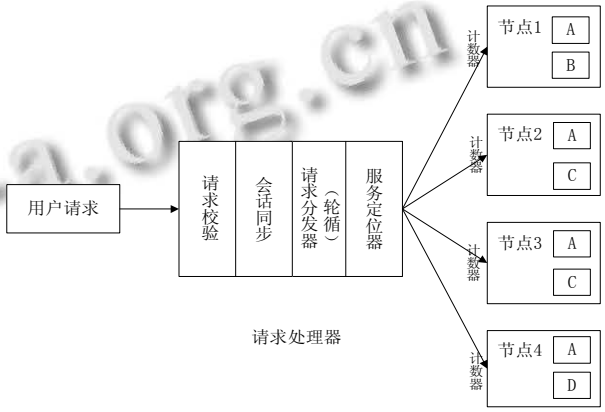

图 4 请求处理器结构图

请求处理器是一个独立的程序，类似于传统的负载均衡器，在功能方面如图 4 所示，主要包含请求校验、会话同步、请求分发器和服务定位器.

请求处理的具体流程图如图 5 所示.

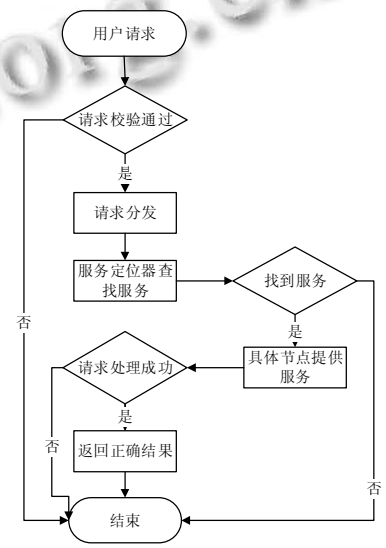

图 5 请求处理图

(1) 请求处理器接收到用户 URL 请求后，先通过

请求校验是否是系统合法 URL.

(2) 通过请求校验后，携带用户会话信息交给请求分发器. 普通应用的访问路径为: http://ip:port/应用上下文，分布式结构下的访问路径为: http://ip:port/应用上下文/模块上下文. 假定 A、B、C、D 四个功能模块的模块上下文为 a, b, c, d. 要访问 A 功能模块，则访问路径为: http://ip:port/app/a/，后面再拼接上具体要访问的资源，比如 index.jsp 等. 请求分发器中会记录记录各节点所含的功能模块的请求数量，每进来一个请求，该节点被请求的功能模块的请求数加 1. 当下一个请求到达请求处理器时，请求分发器会通过计数轮循算法方式确定由四个节点中的哪个来处理，从而保证负载均衡地分发到不同节点上[4].

(3) 服务定位器存放各个功能模块和其注册服务的映射关系，具体的服务转发和处理由服务定位器完成.

2.4 分布式服务设计

如果各个功能模块有依赖关系，在传统 Web 应用结果下，各模块处于同一个 JVM 进程内，OSGi 框架提供了一种本地服务注册机制，使得模块通过服务进行通信，而在分布式结构下，一个分布式节点中的功能模块可能依赖别的分布式节点的功能模块，它们不在同一个 Java 虚拟机中，需要解决各个分布式节点之间远程服务的发布与使用问题.

OSGi 联盟提出的 RFC119 规范用于解决 OSGi 的分布式问题，本文以 RFC119 为标准，通过远程 OSGi 服务的方式来实现分布式节点之间的通讯，使一个功能模块能够使用别的节点的功能模块发布的服务[5]. 分布式服务交互方式如图 6 所示.

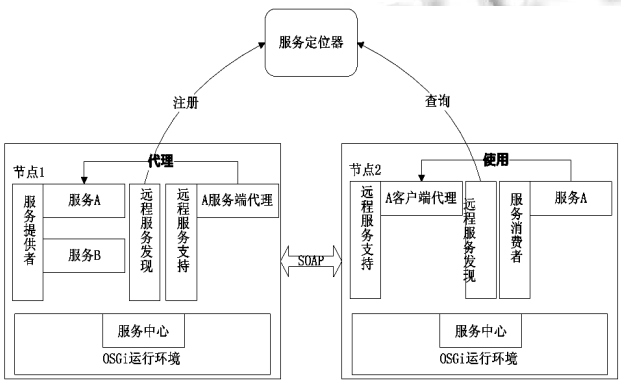

图 6　分布式服务交互图

分布式服务交互由以下关键模块协作实现:

(1) 服务定位器不仅用来处理用户请求，还用来处理各个分布式节点之间的服务发布与使用. 因为多个分布式节点可能注册同一个服务，服务消费者在使用该服务的时候由服务定位器根据负载均衡算法来决定具体使用哪个模块提供的服务.

(2) 远程服务支持模块是对 OSGi RFC119 规范中 Distribution Software 的具体实现. 当模块注册服务到服务中心的时候，如果发现该服务有远程服务的标识，则远程服务支持模块会对该服务的生命周期进行监控，包括服务的注册、查询和注销. 在监控过程中，远程服务支持模块会根据监控情况动态生成服务的服务端代理和客户端代理. 远程服务支持模块在生成服务端代理的过程中，会使服务端代理具备远程服务对象请求调用的能力，当远程服务对象请求请求服务端代理的时候，该请求会转化为 Java 调用请求. 当有 Java 调用请求使用客户端代理的时候，与服务端代理相反，远程服务支持模块会把 Java 调用请求转化为远程服务对象请求.

(3) 远程服务发现模块是对 OSGi RFC119 规范中 Discovery Service 的具体实现. 远程服务发现模块用于把服务和模块的映射关系注册到服务定位器中，并且用于查询服务定位器中服务和模块的映射关系.

在实现分布式服务的交互功能中，远程服务支持模块和远程服务发现模块的交互流程见图 7.

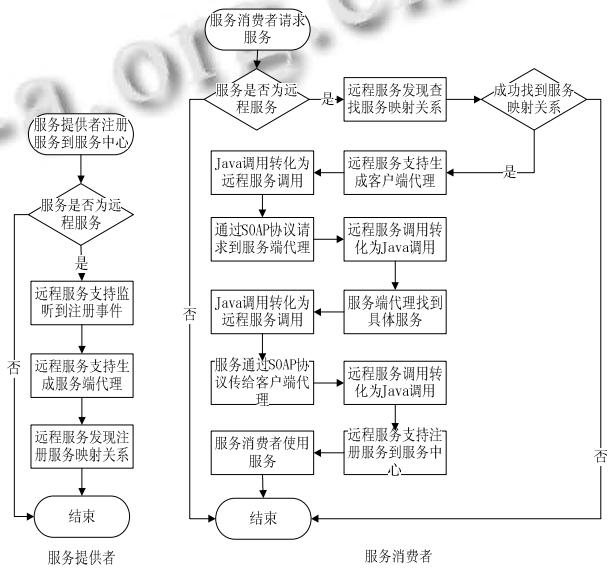

图 7　分布式服务流程图

服务提供者发布服务的时候，首先注册服务到本

地服务中心，如果该服务是远程服务，远程服务支持模块会监听到该服务的注册事件，并把该服务转化为服务端代理，服务端代理是提供给服务消费者使用的，同时远程服务发现模块会注册服务和模块的映射关系到服务定位器.

服务消费者使用 Java 调用请求一个服务，会首先去本地服务中心去寻找服务，找不到服务则去服务定位器寻找远程服务，如果找到服务的映射关系，远程服务支持模块会把 Java 调用请求转化为远程服务对象请求. 远程服务对象请求通过远程服务支持模块访问到服务端代理，这时再把远程服务对象请求转化为 Java 调用请求，Java 调用请求访问服务端代理时，服务端代理转化为具体服务. 远程服务支持模块再把 Java 调用请求转化为远程服务对象请求，具体服务在该请求下通过 SOAP 协议[6]转送给客户端代理. 这时，服务消费者所在的远程服务支持模块会再把远程服务对象请求转化为 Java 调用请求，该请求把客户端代理转化为具体服务，并把服务注册到服务消费者所在节点的服务中心. 这样节点内的模块能像使用本地服务一样透明的使用远程服务[7].

2.5 分布式服务的实现

当服务提供者提供一个服务的时候， 需要配置的参数如表 1.

表 1 服务提供者配置

服务 id	唯一标识一个服务的编码
服务接口	服务接口类
服务实现类	实现服务接口的类
模块白名单	可以使用该服务的模块的 id 集合
IP 白名单	可以使用该服务的 IP 集合
服务地址	服务远程访问地址

具体参考配置如下:

```
<service:provide id="userManagerService"
interface="com.test.user.IUserManager"
class="com.test.user.UserManagerImpl"
includeBundle="module1,module2"
excludeIp="192.168.1.1,192.168.1.2"
remoteAddress="/user"/>
```

能使用“userManagerService”服务的服务消费者为模块白名单和 IP 白名单的交集.

当 remoteAddress 属性有值时，远程服务支持模块会把该服务作为远程服务来处理，将该服务注册到 OSGi 框架的服务中心.

当服务消费者调用一个服务的时候，需要的配置如表 2.

表 2 服务消费者配置

消费 id	唯一标识一个服务消费的编码
服务接口	服务接口类
服务 id	服务提供者提供服务的 id
服务地址	服务提供者提供服务的远程访问地址

具体参考配置如下:

```
<service:require id="userManager Require "
inteface=" com.test.user.IUserManager "
serviceId="userManagerService"
remoteAddress="http:
//192.168.1.3:7001/test/module3/services/user"/>
```

当服务消费者模块启动的过程中，读取到服务消费者相关配置，会先使用客户端代理服务作为服务，代理服务会连接到服务中心，如果代理服务通过服务中心发现远程的服务提供者提供的服务是可用的，就会自动转化为服务提供者提供的服务.

3 实验与讨论

3.1 实验

实验对传统 Web 应用集群部署和基于 OSGi 的 Web 应用分布式部署两种方式分别进行测试，然后进行数据对比. 实验在四台服务器上进行，服务器的环境如表 3. 传统 Web 应用集群部署下，每台服务器各部署一个节点，应用包含 A、B、C、D 四个功能模块. 如图 1 所示.

表 3 服务器环境

CPU	Intel(R) Core(TM) i3-2328M 2.20GHz
操作系统	Win7 32 位
内存	4.00GB
应用服务器	Weblogic 92
数据库	Oracle 10g

各功能模块在用户访问高峰时段的并发数如表 4 所示，数据为模拟数据，并发测试工具采用 LoadRunner11.

表 4 各功能模块高峰时段的并发

模块名	并发数
A	400
B	100
C	200
D	100

分布式部署需要根据不同模块的并发数来调整模块在各个节点的分布, 所以根据表 2 所示的各模块的最大并发数, 分布式部署可以按照图 3 所示的方式. 模块 A 并发最大, 所以模块 A 四个节点都部署, B、D 模块并发最小, 只部署一个节点. C 则部署在 2 个节点上. 节点 3 和节点 4 总的负载很小, 故把节点 3 和节点 4 部署在一台服务器上, 节约出一台服务器, 仅使用 3 台服务器.

用 LoadRunner 按照表 2 的并发分别对这两种部署方式进行测试, 得出结果如表 5 所示.

表 5 两种方式对比表

模块名	并发数	集群部署耗时(s)	分布式部署耗时(s)
A	400	6.033	5.824
B	100	1.092	1.421
C	200	3.201	4.035
D	100	1.465	1.893

3.2 讨论

由实验数据可以看出, 减少了一台服务器后, 负载最大的模块 A 访问耗时基本没有改变. B、C、D 因为分配的资源减少而时间稍微增加, 但是增加后的耗时情况仍然在接受的范围内. 在实际的大型 Web 应用当中, 有的模块负载非常小, 部署在所有节点和部署在一两个节点上耗时基本差别不大.

生产环境下的 Web 应用, 部署的集群节点数和功能模块的数量都要比实验环境多, 采用基于 OSGi 的分布式部署方式节约资源的优势更加明显.

4 结语

本文所研究的应用结构把分布式 OSGi 思想引入到 Web 应用, 将应用按业务功能分成多个业务功能模块, 模块之间能够按照各自的实际负载情况进行分布式部署, 达到最大化的资源利用率, 从而实现在不增加资源甚至节省资源的情况下提升应用系统的性能, 来满足系统规模日益庞大、系统负载日益增加的情况. 当然实际的生产应用环境比实验环境更复杂, 需综合考虑硬件资源结构、各模块负载的动态变化、各模块在分布式节点上的分配方式、运维复杂度等多方面因素, 尤其是对分布式节点间模块更优化的分配方式研究是下一步的重点工作.

参考文献

1 OSGi Alliance. OSGi Service Platform Release 4 Enterprise Version4.2.http://www.osgi.org/Specifications/Home-Page. 2010-3-1.

2 OSGiAlliance.RFC119Specification. http://www.osgi.org/Specifications/HomePage.

3 刘丹,王宇飞,杨宁.一种基于 OSGi 的 Web 应用模块化架构设计.计算机系统应用,2014,23(1):62–67.

4 郭玲玲,范友贵,李春生.基于中间件的动态负载均衡策略的研究.计算机应用与软件,2010,27(9):94–96,107.

5 于庆梅,赵杰,黄冬梅,尹朝万,等.OSGi 平台服务动态更新的发布/订阅策略.计算机应用与软件,2012,29(6):17–20,32.

6 王建斌,胡小生,李康君,赵靓. REST 风格和基于 SOAP 的 Web Services 的比较与结合.计算机应用与软件,2010,27(9):297–300.

7 Alliance A, Apache CXF Project. http://cxf.apache.org/distributed-osgi.html.