

## 基于动态代理的 Java 远程调用框架的研究

韩 勇 沈备军

(上海交通大学软件学院 上海 200030)

**摘 要** 随着网络技术的发展,越来越多的项目采用了分布式的系统架构。对于这些把系统中的任务分配到不同计算机的架构来说,采用哪种方式在系统中不同的计算机之间进行通信十分重要。提出了一种基于动态代理的 Java 远程调用框架。利用此框架可以使远端的 Java 应用程序之间的调用更加简单、灵活、高效,同时也可以通过加入插入式代码的方式扩充对远程调用的自定义管理。

**关键词** 远程调用 RMI Java 动态代理

### ON JAVA REMOTE INVOCATION FRAMEWORK BASED ON DYNAMIC PROXY

Han Yong Shen Beijun

(School of Software, Shanghai Jiaotong University, Shanghai 200030, China)

**Abstract** With the development of network technology, a growing number of projects adopt distributed system framework. For those frameworks which assign tasks in the system to different computers, it matters greatly what method will be adopted to realize the communications among computers. This paper presents a new Java remote invocation framework based on dynamic proxy, which will make remote Java application invocation simpler, more flexible and more efficient, and also expand user-defined management of remote invocation by adding inserting codes.

**Keywords** Remote invocation RMI Java dynamic proxy

## 0 引 言

随着互联网行业的飞速发展,大部分的网站都要面对情况更多变、访问更频繁的网络应用需求,采用何种架构来组织这些服务是必须要认真面对的问题。因为单个计算机的运算能力的限制以及网络通信能力的大幅增强,大型的提供服务的网站选择分布式技术作为基础的组织架构成为必然。

虽然如此,我们还应看到分布式架构可能对系统造成的不良影响。因为不同计算机之间交互的效率比之本地调用的效率会有较大差别,所以分布式架构会对系统调用效率上有较大影响,由此也引出了数据交互粒度、对象设计方式等其他方面的约束。因此选择一个适合系统本身的分布式架构就显得格外重要了<sup>[1,2]</sup>。

不同的系统平台有不同的组建分布式架构的方案,在基于 Java 平台的系统中,也存在着可以为分布式架构服务的多种远程调用方案。本文提出了一种新的运行于 Java 平台系统之上的分布式调用框架。

## 1 Java 远程调用和动态代理相关技术

### 1.1 RMI

远程方法调用 RMI(Remote Invocation)是在 JDK1.1 中为了封装远程调用过程而引入的,是最重要的 Java 远程调用技术之

一<sup>[3]</sup>。但是利用 RMI 进行远程调用存在一些不足之处,主要体现在几个方面:第一,需要暴露的业务类需要实现 Remote 接口;其次,而业务类中每个需要被远程调用的方法都要抛出 RemoteException 异常;最后,在 RMI 调用中,注册和暴露远程对象的过程还较为繁琐。

### 1.2 EJB

EJB 是一个服务器端的构件架构,它可以简化使用 Java 来构建企业级分布式应用构件的过程<sup>[4]</sup>。通过使用 EJB,可以编写出可扩展的、可靠的、安全的应用,并且不需要编写复杂的分布式构件框架。EJB 被设计为可以支持在任何 Java EE 应用服务器之间进行移植和重用。

但是,EJB 在实际开发中存在以下不足:

1) EJB 容器的成本相当高昂,许多项目并不需要如此高端的服务。

2) EJB2.0 或者更早版本的业务对象不是纯粹的 Java 对象,这种对 EJB 容器过重的依赖和难以测试的特性引起了较大的争议。不过 EJB3.0 已经对此做出了很好的改进。

3) 对于许多开发者来说,EJB 仍然过于复杂。这与当初 EJB 设计的初衷有较大的偏差。用户在许多场合也许更愿意使用更简单的轻量级的框架来实现需要的功能。

### 1.3 Web service

Web service 中以服务(service)的方式对外暴露接口,每个

基于 WSDL 的 Web service 都代表着一个服务(service),每个服务中又包含了一个或者多个服务端点(service endpoint)。每个服务端点都对应于一个远程服务接口。这里面,服务端点的概念与 RMI 中的远程接口概念相类似。具体细节可以参考 W3C 网站关于 Web service 的规范部分<sup>[5]</sup>。

由于 SOAP 协议以 XML 格式为基础,所以需传输的内容有一定的冗余度,并且因为 Web service 常以 http 协议作为通信协议,比起 RMI 来通信效率也会低一些。此外,提供服务的一方也通常需要在 http 服务器下面才能运行。

#### 1.4 动态代理

代理(Proxy)是 GOF(Gang of Four)设计模式里属于对象结构模式中的一种模式<sup>[6]</sup>,其目的是向其他对象提供一种代理以控制对这个对象的访问。

动态代理类是这样一类,可以在运行时,创建它的实例的时候才指定它实现的接口。这样就避免了预先要做多个代理类的麻烦,而可以把这些变化及情况在 invoke()方法中分开<sup>[7]</sup>。

## 2 基于动态代理的Java远程调用框架的设计和实现

针对上述 Java 平台上远程调用方案的一些不足,本文提出了一个基于 RMI 和动态代理的远程调用框架,命名为 Summer。本框架旨在实现以下目标:可以使普通的 Java 对象被远程调用,而无需为此实现与业务逻辑无关的接口;服务端不需要 JRE 之外的其他平台和框架就可以提供远程调用的功能;便于为远程方法添加“拦截器”以扩展远程方法的功能;框架有较好的通信效率。

### 2.1 Summer 的架构

Summer 架构的主要构成部分分为三个包,服务端包 SummerServer、客户端包 SummerClient 以及通信包 SummerRmi,如图 1 所示。其中 SummerServer 和 SummerRmi 的一部分部署在服务端,SummerClient 和 SummerRmi 的一部分部署在调用远程服务的客户端。客户端还应该部署要远程调用的业务对象的接口定义。

SummerClient 包主要负责封装客户端的各种请求,包括获取远程对象、调用远程对象方法以及释放远程对象在本地的引用(根据调用模式的不同,释放远程对象也可能不需要客户端调用)。主要内容包括负责创建本地业务对象的工厂类 RemoteServiceFactory、用以创建动态代理对象和实现方法的 ProxyHandler 类。

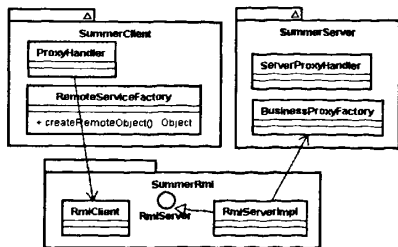


图1 Summer框架逻辑架构图

SummerServer 负责解析客户端传递过来的远程调用指令,并且创建以及维护服务端的业务对象列表以及对对象列表中的对象进行调用。主要包括管理服务端代理业务对象的工厂类

BusinessProxyFactory、用以创建服务端动态代理对象和实现代理方法的 ServerProxyHandler 类。

SummerRmi 在客户端与服务端之间进行通信,传递客户端的各种指令到服务器端。SummerRmi 中的通信部分以 RMI 实现。主要包括:

1) 在服务端端的用于通信的 Rmi 接口 SummerRmi 以及接口的实现类 RmiServerImpl;

2) 在客户端的用于调用 SummerRmi 的 RmiClient 类,同时 SummerRmi 接口也应该部署于客户端。

### 2.2 Summer 原理

Summer 框架的基本思路是这样,为了避免业务对象因为要被远程调用而实现额外的接口,框架在客户端和服务端都利用了动态代理机制来中转实际的调用。服务端采用动态代理的另外一个用途是便于插入横切性的代码。在客户端和服务端之间,框架对 RMI 进行了封装来传输远程调用的方法和返回值。下面分别介绍服务器端和客户端的工作原理。

#### 2.2.1 客户端的工作原理

客户端的工作原理如图 2 所示。

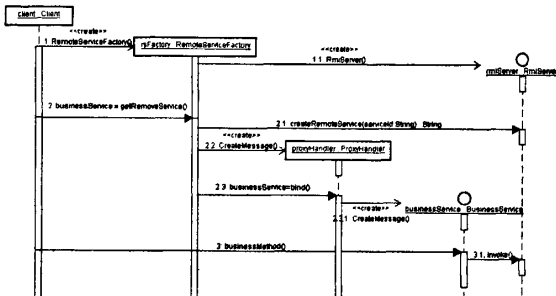


图2 客户端的工作原理

客户端通过三个方法对远程服务进行调用:

方法 1 用户创建一个远程对象工厂。

方法 2 通过远程对象工厂得到远程对象。该过程名是 getRemoteService,过程有一个参数来定义远程服务对象的 id,Server 端将根据 id 从配置文件里读取该 id 对应的对象类型,以创建相应的对象返回客户端。

方法 3 用户通过远程对象调用远程对象的方法。

用户在调用远程框架之前,会先创建远程服务工厂 RemoteServiceFactory 对象,这个类的构造过程中会创建一个用于与 Server 端进行交互的 RmiClient,同时该 RmiClient 与 Server 端建立连接。

在方法 2 中,客户端开始请求一个远端对象,RemoteServiceFactory 根据客户端的请求通过 Rmi 向 Server 端发送该请求,Server 端准备好对象后会将该对象的 instanceId 传回。RemoteServiceFactory 会通过动态代理处理类 ProxyHandler 创建出一个实现了业务对象的动态代理对象,在 ProxyHandler 类的 invoke 方法中,会封装所有到动态代理对象的请求,并通过 Rmi 传递到服务器端去,并再通过 Rmi 返回服务端该方法的返回值。Rmi 封装的输入参数的内容是被调用的对象的 instanceId、方法、方法名。

之后,用户就可以直接使用本地的这个动态代理对象来进行操作了。

可以看出,在调用过程中,客户端不需要创建 RMI 远程对

象和寻找 RMI 远程接口,服务器端的远程业务对象类也无需实现 Remote 接口,远程对象只是一个普通 Java 对象即可。

### 2.2.2 服务器端的工作原理

在服务器端,服务程序的 classpath 需要包含 SummerServer 包以及 SummerRmi 包。此外用户只需要将要提供远程服务的对象配置到远程对象配置文件中就可以运行服务程序开始接收客户端的远程调用请求了。该文件可位于服务程序的 class 文件夹下。配置文件的格式采用 xml 格式,配置文件代码片断如下:

```
<? xml version="1.0" encoding="UTF-8"? >
<beans>
  <bean id="addition"
    interface="com.summer.demobusinesslogic.Addition"
    class="com.summer.demobusinesslogic.impl.AdditionImpl">
  </bean>
  <bean id="addition2"
    interface="com.summer.demobusinesslogic.Addition"
    class="com.summer.demobusinesslogic.impl.AdditionImpl2">
  </bean>
</beans>
```

用户在 beans 节点内可以配置要远程调用的类,每个类配置在一个 bean 节点内。客户端创建对象时将会指定 bean 节点里面的 id 属性,从而创建该 id 对应的远程对象。而这里面提供给远程调用的类,无需继承任何远程接口。

服务器端的工作原理如图 3 所示,在客户端可以访问服务端之前,服务器端需要先启动。服务器端启动时,建立一个 RmiServer 以供客户端调用。之后建立业务对象代理工厂,用以生成实现业务接口的动态代理对象。

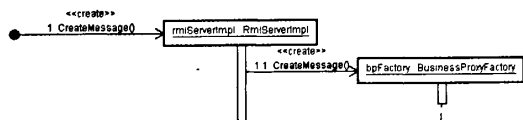


图3 启动服务的时序图

图4是服务端接收客户端指令创建业务对象并调用业务对象的过程。

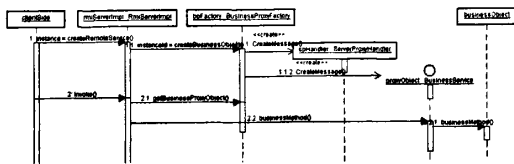


图4 客户端创建和调用业务对象的过程

创建业务对象时,客户端会通过 Rmi 的客户端调用 Rmi 服务端的 createRemoteService 方法,该方法中会传递要创建的业务对象的 beanId。RmiServer 会将此调用传递到业务对象工厂中,工厂对象会在配置的 xml 文件中找到 beanId 对应的业务类。并通过动态代理机制创建实现了这个业务接口的对象。最后将创建的对象的 instanceId 传递回客户端。在创建动态代理对象的同时,再创建一个 beanId 对应的业务对象,并将该业务对象绑定到动态代理对象中。

当 RmiServer 收到客户端通过 Rmi 的 invoke 方法传递的调用请求后,将会根据被请求的远程对象 instanceId、方法以及参数找出 instanceId 所对应的动态代理对象,并调用该对象的对应方法。动态代理对象接到方法后,会调用真实业务对象的对应方法,并在该调用方法的前后依据配置文件查找是否有插入代

码需要执行,如果有则执行之。

下面过程通过一段实例 xml 介绍如何向服务对象中加入横切代码以增强远程服务方法。

```
<? xml version="1.0" encoding="UTF-8"? >
<beans>
  <bean id="addition"
    interface="com.summer.demobusinesslogic.Addition"
    class="com.summer.demobusinesslogic.impl.AdditionImpl">
    <interceptor_ref method="add" check_flag="before">
      interceptor_name="interceptor"
      inerceptor_method="checkUser"
    </interceptor/>
    <interceptor_ref method="add" check_flag="after">
      interceptor_name="interceptor"
      inerceptor_method="writeLog"
    </interceptor/>
  </bean>
  <bean id="interceptor">
    interface="com.summer.demobusinesslogic.Interceptor"
    class="com.summer.demobusinesslogic.impl.InterceptorImpl">
  </bean>
</beans>
```

根据如上的配置文件,addition 为提供远程服务的 bean。在 addition 的配置中可以看到,有两个拦截点,其中一个是在 add 方法前调用 Interceptor 的 checkUser 方法,用以检查用户的权限,另外一个在 add 方法之后调用 Interceptor 的 writeLog 方法,其中 Interceptor 是另外一个处理检查权限和写 log 的 bean。

通过这样的方法,用户可以灵活地在服务代码里面插入各种横切代码,从而对远程服务进行增强。

## 3 框架应用实践

Summer 框架已于 2008 年 8 月开发完成,由于和游典网(<http://www.tripdict.com>)的部分网站架构需求相吻合,故将本框架在游典网项目中进行了应用实践。

游典网是一个旅游爱好者交流平台网站,网站中提供了许多与旅游相关的服务。这些服务因为各种原因通过不同的 Server 提供服务。现以天气预报服务为例说明框架的应用。

该网站的天气预报获取服务由一个独立的服务器提供,当页面请求某个 IP 地址的天气时,则由该服务器查找该 IP 地址对应的天气,并返回给应用程序服务器。应用该框架后,提供天气预报服务的接口无需针对 RMI 做改变,创建的对象也仍然是普通的 Java 对象。在提供服务之前,插入了调用权限检查的代码,在提供服务之后,如果发现了无法匹配到天气的 IP 地址,插入的后处理代码进行了记录。这些插入的代码在提供了这些拦截式服务的同时,无需对业务代码和远程框架作出任何改动。

通过本框架的使用,游典网成功地部署了远程的天气预报服务,对系统负载进行了有效分割。应用结果表明,本框架在保证了远程调用高效性、简单性的基础上还提高了可扩展性,对应用系统具有良好的实践效果。

## 4 总结和展望

本文针对常用的 Java 远程调用方案的一些不足之处,提出

(下转第 144 页)

的基于语义匹配的服务选择方法进行比较,针对候选服务数分别为 12、24、36、48、60 的服务选择流程进行模拟,每个候选服务在其 QoS 多维属性中,挑选各个维度的三个特征属性进行匹配。三个方法分别运行 10 次,对比结果如图 3 所示。

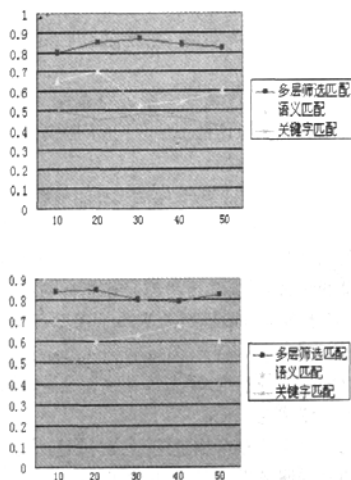


图3 查准率及查全率对比效果图

这些方法的平均查准率分别为 43%, 55%, 86%; 平均查全率分别为 49%, 67%, 83%。

实验结果表明,传统关键字匹配由于没有考虑语义功能,具有很大的局限性;Seog-Chan Oh 提出的基于语义匹配的服务选择方法性能随着测试集波动较大,这与它只涉及服务功能而没有涉及服务的 QoS 约束有关;多层匹配筛选模型在各个测试集合里面一直保持较好的性能,在多维 QoS 指标的支持下,找到满足语义需要的服务,较好地满足了用户的需求。

### 3 结 论

Web 服务目前在被各个领域越来越广泛的应用,在 Web 服务的海量信息中如何更准确、更快速找到和定位所需服务也变得日益重要。本文结合逐层筛选的思想引入一个改进的多层匹配筛选模型。经过实验验证,多层匹配筛选模型在各个测试集合里一直保持较好性能,为 Web 服务发现提供了一种有效的方法。由于在此模型中的第一层及第二层需要对权值进行计算,因此时间复杂度比较高,我们将在今后的工作中进行一定的改进。

### 参 考 文 献

- [1] 岳昆,王晓玲,周傲英. Web 服务核心支撑技术研究综述[J]. 软件学报,2004,15(3):428-442.
- [2] Maximilien E M, Singh M P. A Framework and Ontology for Dynamic Web Service Selection[J]. IEEE Internet Computing, 2004, 9(10):84-93.
- [3] Yu T, Lin K J. Service selection algorithms for Web service with end-to-end QoS constraints[C]//The IEEE Int Conf on E-Commerce Technology (CEC 2004). New York, IEEE Press, 2004:129-136.
- [4] 李春梅,蒋运承. 具有 QoS 约束的语义 Web 服务发现的研究[J]. 计算机科学,2007,34(6):116-121.

- [5] 段初,张卫华,张波. 一种基于语义的 UDDI 服务发现机制[J]. 微电子学与计算机,2007,24(9):206-208.
- [6] 张佩云. 基于语义匹配和 QoS 的 Web 服务混合选择方法[J]. 武汉大学学报:信息科学版,2008,33(5):537-541.
- [7] 郑晓霞,王建仁. 基于 QoS 的 Web 服务发现模型研究[J]. 情报科学,2007,25(2):249-253.
- [8] Rvdriguez A, Egenhofer M. Determining Semantic similarity Among Entity Classes from Different Ontologies[J]. IEEE Internet Computing, 2004, 9(10):84-93.
- [9] 郭得利,任彦,陈洪辉,等. 一种基于 QoS 约束的 Web 服务选择和排序模型[J]. 上海交通大学学报,2007,41(6):870-875.
- [10] Zeng L, BenatSalla B. Qos aware middle are for Web service composition[J]. IEEE Transactions on Software Engineering, 2004, 30(5):311-327.

### (上接第 138 页)

并实现了一种基于动态代理的 Java 远程调用框架,并对框架进行了实践和检验。实践表明,通过利用本框架对远程服务对象进行配置,具有如下好处:

- 1) 服务提供商无需购买昂贵的 EJB 服务器来进行远程调用。
- 2) 远程服务对象和其对应的接口无需针对其他框架做任何修改,仍然可以保持普通 Java 对象原貌。
- 3) 在服务端远程业务对象被调用的同时,可以通过插入式编码加入针对业务方法的拦截器,从而可以灵活地增强业务方法。

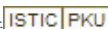
同时通过本框架的实际使用,我们也发现了框架的一些不足,下阶段计划在以下方面进行改进:

- 1) 提高传输的效率。现在用的是 Java 默认的序列化方法来传递方法和对象,而 Java 的默认序列化方法在压缩要序列化的内容方面还有提高的潜力<sup>[8]</sup>。可以考虑针对要传输的内容进行压缩,从而减少网络的交互量,当然,是否需要压缩,也要看传输的内容是否较多,可以通过配置来决定。
- 2) 增加一些异步传输的机制。有些远程调用适合用异步方式来完成,尤其是相对于一些需要耗时较多的资源来说。
- 3) 增强数据安全性。在某些应用场合,传输的内容需要很高的安全等级,可以考虑对传输的内容在传输的前后进行加解密的工作。同样,是否需要加解密也应该在客户端进行配置。

### 参 考 文 献

- [1] Martin Fowler. 企业应用架构模式. 王怀民,周兵,译. 机械工业出版社,2004.
- [2] Rod Johnson, Juergen Hoeller. J2EE Development without EJB. Java Eye, 译. 电子工业出版社,2005.
- [3] RMI Specification. <http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html>.
- [4] Roman E. 精通 EJB. 刘晓华,等译. 电子工业出版社,2003.
- [5] Web service specification. [www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl).
- [6] Erich Gamma, Richard Helm, Ralph Johnson, et al. Design Patterns-Elements of Reusable Object-Oriented Software. 李英军,马晓星,蔡敏,等译. 1994.
- [7] 透明. 动态代理的前世今生. 程序员,2005(1).
- [8] Christian Neester, Michael Philippsen, Bernhard Haumacher. A more efficient RMI for Java. ACM, 1999.

# 基于动态代理的Java远程调用框架的研究

作者: 韩勇, 沈备军, Han Yong, Shen Beijun  
作者单位: 上海交通大学软件学院, 上海, 200030  
刊名: 计算机应用与软件   
英文刊名: COMPUTER APPLICATIONS AND SOFTWARE  
年, 卷(期): 2010, 27 (6)

## 参考文献(8条)

1. Erich Gamma; Richard Helm; Ralph Johnson; et al. 李英军, 马晓星, 蔡敏 Design Patterns-Elements of Reusable Object-Oriented Software 1994
2. Web service specification
3. Roman E; 刘晓华 精通EJB 2003
4. RMI Specification
5. Rod Johnson; Juergen Hoeller; Java Eye J2EE Development without EJB 2005
6. Martin Fowler; 王怀民; 周兵 企业应用架构模式 2004
7. Christian Nester; Michael Philippsen; Bernhard Haumacher A more efficient RMI for Java 1999
8. 透明 动态代理的前世今生 2005(01)

本文链接: [http://d.g.wanfangdata.com.cn/Periodical\\_jsjyyyrj201006044.aspx](http://d.g.wanfangdata.com.cn/Periodical_jsjyyyrj201006044.aspx)