文章编号:1001-5132（2007）01-0046-05

Java RMI的企业级应用及框架设计

赵永标

（宁波大学 理学院，浙江 宁波 315211）

摘要：对 RMI 在企业级应用进行了探讨，分析了 Java RMI 在企业级应用中存在的缺陷，并设计一个框架作为运行容器的方案来解决这些缺陷，该方案能有效地应对企业级的应用.

关键词：RMI；企业级应用；缺陷；框架

中图分类号：TP311.52 文献标识码：A

Java RMI (Remote Method Invocation，远程方法调用)是用Java语言在JDK1.1 中实现的，它大大增强了Java开发分布式应用的能力[1,2]. Java的独特优势体现在它强大的开发分布式网络应用的能力上，RMI是开发百分之百纯Java网络分布式应用系统的核心解决方案之一. RMI目前使用Java远程消息交换协议JRMP(Java Remote Messaging Protocol)进行通信，而JRMP是专为Java的远程对象制定的协议. 因此，Java RMI具有Java的“ Write Once, Run Anywhere ”的优点，用Java RMI开发的应用系统可以部署在任何支持JRE(Java Runtime Environment，Java运行环境)的平台上.

Java RMI 技术给应用程序开发者提供了一种可以调用远程对象方法的抽象功能，隐藏了低级通信的过程，提供了在 Java 技术领域开发分布式系统的基础技术.

1 RMI 在企业级应用的一些缺陷

企业级分布式系统要求整个系统架构在高容量、高安全性、高稳定性和具有强大的灵活性的基础之上，这在很大程度上需要在系统架构的层次上予以支持[3]. 基于对RMI技术内涵的理解和实际的工程开发经验，可以得出：在开发具有高要求的应用时，RMI为我们提供了技术基础，但它所包含的或者它所反映出的一些JVM的基础性服务由于存在 2 个缺陷，仍然不足以成为一个成熟的应用程序服务器.

1.1 不能妥善管理 RMI 连接的问题

这里的 RMI 连接不是指客户端与服务器端的连接过程，而是需要假定在分布式系统可以正常运行的基础上探讨更深层次的问题. 因为如代理类和框架类通信类的基础连接技术问题，RMI 内含的基础服务器可以通过抛出异常来解决. RMI 连接的问题可以通过以下案例来说明.

如果客户端调用服务器端，需要访问服务器端的一个私有对象，这意味着处理这个对象的任何方法都是同步的，因此调用的方法，不一定可以及时获得对处理对象的处理权. 甚至，如果其他方法在处理这个对象的过程中发生了意外堵塞，所调用的

收稿日期：2006-09-13. 宁波大学学报（理工版）网址：http://3xb.nbu.edu.cn
作者简介：赵永标（1980－），男，山东鱼台人，在读硕士研究生，主要研究方向：自动推理与软件设计. E-mail: g04c07010101@email.nbu.edu.cn

远程方法有可能永远也无法获得对该对象的处理权，从而导致 RMI 连接线程进入了非正常状态，那么不仅原始的用户被阻塞，而且 RMI 连接线程会永远的挂起. 因为基础的 RMI 的服务器没有机制对这一情况做出处理，也不会抛出异常，因此 RMI 连接线程也没有自己自动终止的理由. 需要解决这一问题，进而提供错误记录和恢复机制. 这一类问题可能遇到的时候不多，但却是需要在系统级解决的问题之一.

1.2　不能妥善管理服务器端逻辑处理的问题

此问题来源于RMI这种分布式技术的本质：RMI技术所进行的任何的远端方法调用，都是属于异步处理的范畴[4]. 因为应用程序逻辑与接口位于不同的Java虚拟机(JVM)，需要发送此请求并接收来自另一 JVM 的回答. 众所周知，这种完全异步的环境缺乏从客户端管理服务器端逻辑处理的功能，这需要在RMI的服务器端予以解决.

客户端调用远程对象方法有 2 种：同步请求和异步请求. 如果需要服务器端返回一个处理结果，那这个结果可以作为服务已经正确执行的一个证明，这是同步请求. 但调用的远端方法，有可能只是引发了一个远程对象外部逻辑或者线程的执行，这是异步请求，RMI 的基础服务没有对这种执行情况的监督机制. 在这种放任自流的情况下执行企业级应用，无异于为整个系统的稳定性和安全性埋下了一个极大的隐患. 解除这一隐患需提供服务器端逻辑处理管理机制，提供错误恢复功能.

妥善管理服务器端逻辑处理的问题，还意味着必须提供机制来解决服务器端大容量处理的问题. 如果仅仅为每个请求创建一个新的应用程序线程，创建或销毁的开销和线程的数量将给服务器带来严重的压力，并且 JVM 最后将把资源用完. 这也是需要在系统架构级解决的问题之一.

1.3　解决问题的设想

考虑到这些需要在系统级解决的问题，加上EJB(Enterprise JavaBeans，企业组件)技术的成功，甚至EJB系统的开发，比普通的RMI开发简单. 这是基于这样一个事实：EJB运行于容器内. 这一容器，实质上就是一个保障EJB正常运行的框架(Framework)，它用于管理持久性、消息传递、线程管理、日志记录、事件队列、用户界面等等[5]. 利用框架是在系统架构级解决重大问题的妥善方案.

无疑，RMI 客户端和服务器端的处理逻辑管理的功能，管理连接线程的问题，日志管理的功能，任务队列和多线程的问题，还有错误恢复的功能，如果需要在系统级得到妥善解决，比较好的解决方案是让客户端的逻辑和服务器端的逻辑都在统一的容器内运行. RMI 框架正是要设计成这样一个容器.

2　Java RMI 框架的设计思路

2.1　建立 RMI 框架的理论研究

2.1.1 统一异步请求和同步请求

在同步请求中，RMI 连接线程调用应用程序线程，等服务器返回结果后再中断该连接. 异步请求的事件流程和同步请求的事件流程是一样的，除了异步请求没有要求 RMI 连接线程等待完成以外. 在唤醒了应用程序线程后，RMI 连接线程返回给客户机一个“已调度”的消息. 异步请求作为结果看起来也许很简单，但是存在一个隐藏的困难. 来自应用程序处理类的返回数据会发生什么呢？它的返回数据到哪里去了并不该作为应用程序关注的问题. 开发者必须能够为同步的或是异步的请求使用相同的应用程序逻辑. 所以，需要一个异步请求的代理，该代理会进行一些后续处理. 这样，在增加了代理类后，对服务器来说，同步请求和异步请求的处理机制就一样了.

2.1.2 保证各个线程间的互相通信

服务器需要开辟一块公共区域(见代码段 1)供各个线程访问，服务器启动时实例化该公共内存，并且在服务启动即建立一个该对象的引用，因为该

引用始终是活动的，而且服务器是持久的，所以JVM 不会垃圾收集该对象. 仅因为在线程间是公共内存，并不意味着线程可以没有任何限制的访问或修改此内存. 修改在多线程中使用的变量的关键是同步语句或同步方法修饰符[6]. 在Java语言中，有共享的主内存和线程内存，JVM 从共享主内存给每个线程它所需的一个变量的副本，并且将变量写回共享的主内存以供其他的线程使用，这是Java的内存模型.

代码段 1　FrameWorkBase 部分代码：

```
public final class FrameWorkBase {
// 公共内存区域,所有的public成员都是static的，保证了数据的唯一性.
// 访问时，必须对该内存块进行同步化
// 队列表
public static FrameWorkMain mainTable = null;
// 提供给客户端的方法
public static FuncTable funcTable = null;
// 异步请求的队列
public static AsyncTable asyncTable = null;
// 同步请求的队列
public static SyncTable syncTable = null;
// 该框架的远程接口
public static FrameWorkInterface fwi = null;
```

RMI 服务器是持久的，这意味着服务器所创建并且还没释放(活动引用)任何对象会为服务器的生命期内保留. 当服务器启动时，它获得一个公共内存类的新的实例，并且分配一个带有该引用的私有的、静态的域. RMI 实现类也获得对公共内存类的引用. 通过这种方式，所有的运行在服务器的线程，包括 RMI 连接线程和应用程序线程，都可以访问该公共的内存.

把对一个对象的访问/改变放进一个同步块或同步方法可以完成 3 个功能：防止其他线程(在同一个对象上同步的)的执行；当线程访问变量时，从共享的主内存读取该变量的当前值到线程存储器；当线程改变了变量时，将线程存储器的变量的新值在块或方法的末尾写到共享的主内存. 所以，为保证完整性和确保所有的线程都可以访问变量的最新值， 就需要同步该内存区域.

2.2　RMI 框架的设计

实用的解决方案是从应用程序处理中分离 RMI 连接活动. 可以通过在服务器端创建一个应用程序队列和线程结构来做到这一点. 这种工作方式具有很高的可靠性，完全以任务为重点，而且该结构适用于任何应用程序. 现在，目标是设计一个应用程序队列和应用程序线程环境以便 RMI 连接线程和应用程序线程可以彼此对话；应用程序环境可以知道客户机超时和恢复；以及不会出现线程过载问题. 当有太多的应用程序线程在执行以至于 JVM 不能再承担更多的线程时，或是当很多的应用程序线程引起太多的资源竞争以至于环境有效的死锁时，这样的过载就可能会发生. 应用程序线程创建／销毁的开销不能使应用程序陷入泥沼.

作为 RMI 框架，要具备基本的 RMI 要素：

(1) 一个接口继承了java.rmi.remote，任何对象要想参与和另一个Java对象的远程会话，就必须直接或间接地实现该接口[7]. 对于该框架，还需要提供至少 2 个方法.

处理同步请求的 syncRequest()方法；

处理异步请求的 asyncRequest()方法.

(2) 实现 FrameWorkInterface 接口的具体实现类(见代码段 2).

代码段 2　FrameWorkImpl 部分代码：

```
public final class FrameWorkImpl
        extends UnicastRemoteObject
        implements FrameWorkInterface {
private FrameWorkBase fwb;  //公共内存区域
// 含参构造方法，传递公共内存区域的引用
public FrameWorkImpl (FrameWorkBase
reference_to_fwb)  throws RemoteException {
// 得到公共内存区域的引用
```

fwb = reference_to_fwb.

(3) 创建 RMI 服务器环境的启动类，它包含用于建立持久环境的逻辑(见代码段 3).

代码段 3　FrameWorkServer 部分代码：

```
public final class FrameWorkServer {
private static FrameWorkBase fwb = null;
// 服务启动
public static void main(java.lang.String[] args){
    // 初始化公共内存区域
    fwb = new FrameWorkBase();
    FrameWorkImpl fwi=new FrameWork Impl(fwb).
```

除了 RMI 的基本要素外，还要包含如下内容：

(1) 定义优先级等待列表、应用程序线程、应用程序处理类的列表类(见代码段 4).

代码段 4　WorkUnitHeader 部分代码：

```
public final class WorkUnitHeader {
private String  work_unit_name;  // 应用程序处理类的名称
private WaitLists waitlist;  // 等待列表
private int nbr_of_waitlists;  // 等待列表的最大个数
// 最大等待时间，如果在该时间内没有处理完毕，则抛出异常
private int max_wait_time;
private int nbr_of_threads;  // 处理线程的个数
public WorkUnitInterface work_unit;  // 应用程序处理类要实现的接口.
```

(2) 从等待列表取得请求并调用应用程序处理类的线程类(见代码段 5).

代码段 5　WorkUnitThread 部分代码：

```
synchronized (this) {  // 同步化
while ( ! posted) {  // 等待请求完成或者超时
    try {
        wait(time_wait);
    } catch (InterruptedException e) { }
    time_now = System.currentTimeMillis ();
    time_wait -= (time_now - start_time);
    if  (time_wait < 1) {  // 超时
    break;
    } else {
    start_time = time_now;
    }
}
}
```

(3) 执行应用程序逻辑的应用程序处理类 (见代码段 6).

代码段 6　WorkUnitInterface 部分代码：

```
// 应用程序处理类的公共接口，所有处理类都需要实现 doWork，该方法执行具体的业务逻辑.
public interface WorkUnitInterface {
public Object doWork(Object input, Frame WorkInterface fwi) throws java.lang.Throwable.
```

(4) 将客户机信息传递给 RMI 服务器的参数类(见代码段 7).

代码段 7　FrameWorkParm 部分代码：

```
public final class FrameWorkParm
implements java.io.Serializable {
private Object client_data;  // 从客户端传递过来的数据，封装成 Object 对象.
private String func_name;  // 客户端调用的具体的方法名
private int max_wait_time;
private int priority;  // 该请求的优先级.
```

图 1 表达了一个 RMI 框架的请求处理的流程，它包含了 RMI 框架设计的思路，只是有一些细节没有包括在内，这对 RMI 框架的设计思路没有影响. 在这个 RMI 的框架设计思路上，就可以不断的扩充和完善，使得 java RMI 更有效地进行企业级的应用.

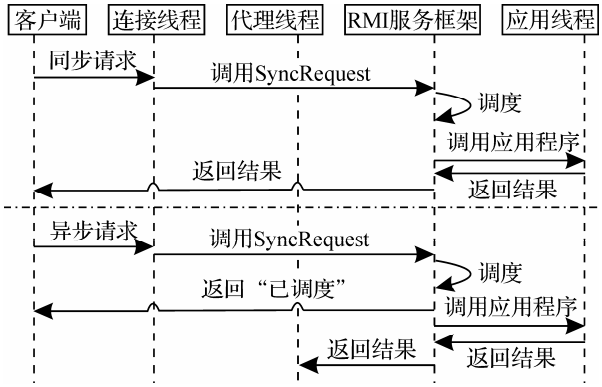

图 1 一个简单的 Java RMI 框架的请求处理流程

3 结语

为了更好地应用 RMI，分析了其作为成熟的应用程序服务器在企业级存在的一些缺陷，给出了一个 RMI 框架的设计方案，从而使得 RMI 能够更有效、更稳定、更安全地应对企业级应用.

致谢 在此向对本文的工作给予支持和提供建议的同事，尤其是向该框架的编码人员表示感谢.

参考文献：

[1] Sun Microsystems, Inc. Java Remote Method Invocation (Java RMI) [EB/OL]. [1997-02-18]. http://java.sun.com/products/jdk/rmi/.

[2] LI Zhi, REN Bo, WANG Chen. Study of Java-based distributed computing environment[J]. Computer Engineering and Design, 2004, 25(6):912-916.

[3] ZHU Jian-Zhong. Implementation of distributed computing based on RMI[J]. Computer Engineering, 2003, 29 (10):61-67.

[4] YU Jun, YU Ruitao. The theory and implement method of Java RMI[J]. Journal of Qingdao University, 1999, 14 (3):16-19.

[5] Edward Harned. A Java RMI server framework [EB/OL]. [2001-10-01].http://www-106.ibm.com/developerworks/java/library/j-rmiframe/.

[6] Qing Jun, Meng Dan, Gu Zhi-Ming. The application of Java RMI and multi-thread technology in cluster management system[J]. Computer Engineering and Application, 2004, 40(13):108-110.

[7] Wang Meiqing, Zheng Wenbo, Zheng Shouqi. Distributed computing system with Java RMI[J]. Mini-Micro System, 2000, 21 (9):908-910.

Java RMI’s Application in Enterprises and its Framework Design

ZHAO Yong-biao

(Faculty of Science, Ningbo University, Ningbo 315211, China)

Abstract: The RMI’s application in enterprises and its limitations are investigated in this paper. In order to work out a solution to these limitations, a scheme is proposed for designing a framework, which is to handle the Java RMI’s application in enterprises effectively.

Key words: RMI; enterprise application; fault; framework

CLC number: TP311.52 **Document code:** A

（责任编辑 史小丽）