为RMI实现类Jini的发现机制 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/273/2021_2022__E4_B8_BARMI_E5_AE_9E_E7_c104_273301.htm 如果你从事过Jini开发，你会知道Jini客户端是不需要知道服务的位置的；它们简单地通过发现机制来获得一个代理以访问它们需要的服务。相反，在RMI（远程方法调用）中，你必须知道你想访问的服务器的URL。 在本文中，我们将向你展示怎样为RMI实现一个类Jini的发现机制，这将使得一些客户端从必须知道RMI服务器URL的麻烦中解脱出来。 你可能首先会想，为什么要这么麻烦；为什么不干脆用Jini？我们也同意这样的逻辑，特别是对新的系统来说。不管怎样，已经有许多基于RMI的系统存在，并且在Jini被Java开发的主流接受以前，我们仍然要提供更优雅的RMI解决方案。事实上，我们在这儿描述的工作，是这样的需求的结果：开发一项Jini服务使它同时可以作为一个独立的RMI服务器运行，但使用类Jini的发现机制。 本文主要是针对没有用过Jini的RMI开发者。通过深入观察Jini内部的运作，我们希望你能开始了解Jini的机制有多么强大。我们当然不是希望你重新实现Jini，但这篇文章能帮助你理解这些机制是怎样运作的。甚至可能帮助你说服你的经理或部门头头，该考虑将Jini作为一项可行的分布式系统技术。 我们不会太深入Jini的发现机制，所以如果你对此不是很熟悉，我们建议你快速浏览一下Bill Venners的"Locate Services with the Jini Lookup Service."（ http://www.javaworld.com/javaworld/jw-02-2000/jw-02-jiniology.html） RMI基础和Jini查找 在RMI中，客户端必须知道它所要

连接的服务器的位置。RMI服务器的地址是URI的形式rmi://:/，其中端口号是rmiregistry用来侦听请求的端口。例如：Translator service =(Translator)Naming.lookup("rmi://theHost/SpanishTranslator"). 在Jini中，客户端通过一个Jini工具类来找到服务，比如ServiceDiscoveryManager。在下面的例子中，我们创建了一个ServiceTemplate的实例，该实例包含一个类列表；在我们的例子中，是我们要匹配的类Translator.class： Class [] classes=new Class[]{Translator.class}. ServiceTemplate tmpl=new ServiceTemplate(null,classes,null). ServiceDiscoveryManager lmgr=new ServiceDiscoveryManager(null,null). ServiceItem serviceItem =lmgr.lookup(tmpl,null). Translator service=serviceItem.service. 正如我们从例子中可以看到，ServiceDiscoveryManager用lookup()方法来查找任何与ServiceTemplate匹配的可用的服务。你还可以在服务查找中使用任何数字或属性；在这里我们出于保持简单和精练的考虑而没有这样做。 比较两种查找机制，你会注意到在Jini版本中没有指定服务的位置。值得一提的是，如果必要，你也可以指定一个查找服务的位置，但不是你想要访问的实际服务的位置。Jini模型的强大之处是，我们不需要知道或关心服务位于何处。 比较了RMI和Jini的发现机制之后，现在我们可以考虑怎样用类Jini的风格来访问一个RMI服务器。 位置中立的RMI查找 理想地，我们考虑查找Translator所发现的第一个匹配的实例。 Translator service =(Translator)RMIDiscovery.lookup(clazz,id). 在这里clazz是RMI服务的接口，id是区分实现clazz接口的不同服务器实例的唯

一字符串标识。例如，要找到一个西班牙语翻译器，我们用下面的代码： Class clazz=Translator.class. String id="Spanish". 现在我们对如何使用RMI发现机制有了一个更好的主意，我们来研究一下怎样实现它。在我们尝试实现我们“ 简陋的 ”RMI发现机制以前，先来看看Jini是怎样做的，再把这些原理/概念适用到RMI服务器和客户端上。 发现机制 Jini的基本发现机制联合使用多播UDP（用户数据报协议）（multicast UDP 见文后的Resources）和单播TCP/IP。简单来说，这意味着客户端发出一个多播的请求数据包，然后数据包被监听它的查找服务拾取。然后查找服务用单播连接连回客户端，并把查找服务的代理串行化成流通过此连接发送出去。此后客户端就可以和查找服务（的代理）交互以定位它需要的服务。 发现机制实际上比这要复杂得多，但我们只对其中多播UDP和单播TCP/IP的关键概念感兴趣。 我们并不打算实现一个等同的独立运行的RMI查找服务。相反我们将实现一个简单的多播监听器/单播分发器（multicast listener/unicast dispatcher）供RMI服务器使用，实际上我们使得每个RMI服务器作为它自己的查找服务。在客户端，我们为服务器端socket写个配对物一个多播分发器/单播监听器（multicast dispatcher/unicast listener）。 