OA 自动化 NETRS 性能分析及利用 L 进行性能测试的方案

.NETRemotingServer 性能分析及利用 Loadrunner 进行性能测试的方案

1 概述

.NETRemoting 被誉为管理应用程序域之间的 RPC 的首选技术。应用程序域是公共语言运行库的隔离单元，它们是在进程内创建并运行的。这与 CLR 和非 CLR 托管的进程之间的进程间通信（互操作）不同。后一种类型的 RPC 通信（特别是 Web 上的）一般被认为是 Web 服务领域的问题。遗憾的是，这种看似清楚的区分，却由于可以在 IIS 下集成 .NetRemoting 服务器而变得模糊，“通过在 IIS 中集成 .NETRemoting 对象，可以将其作为一种 Web服务提供……”

Remoting ，简而言之，我们可以将其看作是一种分布式处理方式。从微软的产品角度来看，可以说 Remoting 就是 DCOM 的一种升级，它改善了很多功能，并极好的融合到.Net 平台下。Microsoft®.NETRemoting 提供了一种允许对象通过应用程序域与另一对象进行交互的框架。这也正是我们使用 Remoting 的原因。为什么呢？在 Windows 操作系统中，是将应用程序分离为单独的进程。这个进程形成了应用程序代码和数据周围的一道边界。如果不采用进程间通信（ RPC ）机制，则在一个进程中执行的代码就不能访问另一进程。这是一种操作系统对应用程序的保护机制。然而在某些情况下，我们需要跨过应用程序域，与另外的应用程序域进行通信，即穿越边界。其主机与客户端的主要任务如下：

主机任务

·设计服务，选择应用程序域、激活模式、通道、端口和发布。

·实现 Remoting 主机应用程序域（例如 IIS/系统服务）。

·配置主机激活、通道和协议设置。建议使用配置文件，可以通过调用 RemotingConfiguration.Configure 加载。

·发布接口，供客户端使用（有关详细信息，请参阅下文中的“接口发布选择”）。

客户端任务

·设计客户端，选择应用程序域和激活模式。

·考虑是否需要注册通道和端口。

·获取远程类型元数据。

·实现客户端应用程序域。

·配置客户端激活模式和其他类型的信息，如应用程序名称、通道和对象 URI 等。建议使用配置文件，可以通过调用 RemotingConfiguration.Configure 加载。

2Remoting 解决方案的过程中可能会遇到的错误情况

在任何情况下，都应该记住要使用标准的设备使用和监视方法。事件记录仍是非常有价值的信息资源，就象网络监视器工具一样，网络监视器可以专门用于详细查看客户端/服务器的 Remoting 会话。中间层的 Remoting 服务器仍可以使用 VisualStudio.NET 提供的标准调试工具进行调试，例如，对于由 IIS 集成的 Remoting 服务器，可以通过向 ASP.NET

辅助进程附加调试会话（VisualStudio.Net|Debug[调试]|Processes[进程]|Attach[附加]）来设置断点（如果资源可用）。但 Remoting 的错误很独特，下面列出了一些。请注意，所有错误都已使用 .NETFrameworkSDK 提供的 BasicRemotingHelloSample 的各版本进行了复现，服务器和客户端也已在单机上运行。故障现象与在网络链接上的相同，只是由于 HTTP/TCP 的超时设置不同，需要相当长的时间才能出现错误。

2.1 丢失 MarshalByRef

由于 Remoting 要通过引用以用于给定的类，该类必须只做一件事，就是继承 MarshalByRefObject 。假设开发人员忘记做这项工作，我们将得到一个 System.Runtime.Remoting.RemotingException 类型的异常，说明我们有一个“丢失的 MarshalByReference”.

是否能正确捕获和处理这个 RemotingException 将取决于程序员。（想想这个开发人员忘记了他应记住的唯一一件事。）

解决方法是：记住继承 MarshalByRefObject ！

2.2 众所周知的服务器激活的错误服务器端点

对于服务器激活，Remoting 服务器将其侦听处声明为端点。该端点一般包括一个对象 URI（远程对象的众所周知名称），一个协议和一个端口号。当然，所有这些都可能配置错误。

2.3 错误的 URI

由服务提供的 BasicRemotingHelloSample 的 URI 是 HelloService .soap ，如相关的 web.config 文件中所指定：

```
<configuration>
<system .runtime.
remoting>
<application>
<service>
<wellknownmode="SingleCall"type="Hello.HelloService,Hello"
objectUri="HelloService .soap"/>
</service>
```

</application>

</system.runtime.remoting>

</configuration>

此服务是 IIS 集成的。IIS 集成要求 URI 带有后缀 .rem 或 .soap ，我们在服务器上使用 .rope 。在此实例中，我们将再次收到 RemotingException ，这次显示的文本是“对象 </Hello.soap>在服务器上已断开或不存在” 。

请确保各个 URI 相互匹配！当 IIS 集成 Remoting 服务器时，还要确保 URI 以 .rem 或 .soap 结尾。

2.4 不匹配的协议/端口

为了进行此项测试，我们切换到控制台集成的服务器，以下是该服务器的配置文件：

<configuration>

<system .runtime.remoting>

<applicationname="RemotingHello">

<service>

<wellknownmode="SingleCall"type="Hello.HelloService,Hello"

objectUri="HelloService .soap"/>

</service>

```
<channels>
<channelref="http"port="8000"/>
```

</channels>

</application>

</system.runtime.remoting>

</configuration>

假设我们要在服务器端将协议更改为 TCP ，而使客户端保留 HTTP 。

我们将再次收到 RemotingException ，这次的文本是“底层连接已关闭：接收时出现意外错误” 。

端口设置错误也会导致上述异常，唯一的不同是这种情况下，要用较长的时间才会出现错误。服务器和客户端之间的端口和协议必须匹配。

2.5 丢失 URI

另一种可能性是远程服务器没有运行，例如，服务器由 IIS 集成，而虚拟应用程序或相关的程序集丢失。再次使用 BasicHelloRemoting 服务器，我们需要虚拟应用程序 RemotingHello 能够运行。如果不能运行，我们将收到未处理的异常（取决于调用代码），但这次的异常将是：“无法加载类型 clr:Hello.HelloService,Hello” 。

在这些情况下，请确保虚拟应用程序在运行，而且所需的程序集正确地放置在相关的 bin 子文件夹中。

总而言之，客户端必须正确地引用服务器定义的端点以便激活服务器，这意味着，端口、协议和 URI 的定义必须相互匹配。这太容易出错了。因此，如果服务器的位置定义为：

```
<service>
<wellknownmode="SingleCall"type="Hello.HelloService,Hello"
objectUri="HelloService.soap"/>
```

</service>

那么，客户的设置必须为：

<clienturl="http://localhost/RemotingHello">

<wellknowntype="Hello.HelloService,Hello"

url="http://localhost/RemotingHello/HelloService .soap"/>

</client>

其中，URL 表示集成 Remoting 服务的 IIS 虚拟应用程序，类型表示类和程序集名称。

3Remoting 的特点

3.1 优点

他的优点是用户既可以使用 TCP 信道方式进行二进制流方式通信，也可以使用 HTTP 信道进行 SOAP 格式的性通信

效率相对 WebService 要高不少；但是它的缺点也很明显， .netremoting 只能应用于 MS 的 .netframework 之下。

从性能上来讲 Remoting 的效率和传统的 DCOM 、COM+ 的性能很相近。

3.2 缺点

这种三层设计的缺点与使用 XMLWebservice 的三层设计的缺点相同。

所有业务规则均包含在前端代码中。因而，如果需要更改业务规则，则必须更新全部客户端。除非能够进行自动更新，否则这种维护工作将十分繁琐。当然，如果使用 SQLServer ，则可以将某些业务规则放到存储过程中，从而减少维护的时间和成本。

所有字段名称均在源代码或控件属性中硬编码。如果更改字段名称，则必须查找和替换应用程序中所有该字段的名称。如果使用了数据绑定，还必须检查所有窗体并更改属性。

通过网络从一个组件向另一个组件传输数据比直接连接数据库要慢。在 Intranet 方案中，.NETRemoting 的性能比 XMLWebservice 要好。而在 Internet 方案中，一般不使用 .NETRemoting 。

建立这种应用程序比建立两层应用程序或使用 XMLWebservice 的应用程序要复杂一些。

必须使用比 TCP 速度慢的 HTTP 。另外，IIS 可能循环执行 ASP.NET 辅助进程，这将破坏所有 Singleton 的状态。对您来说，这可能是问题也可能不是问题，要取决于您的设计需

要，因为客户端的下一个调用将重新启动 Singleton 。您可以将 IIS 配置为不循环执行辅助进程，但这种能力很有限，特别是在 IIS5 中，而且可能造成更进一步的影响。这里最根本的意思是，如果要求远程服务器的安全性，那么无疑要使用 IIS 集成。

4 调试小技巧

编写 Remoting 程序，通常分为三部分：远程对象、服务端程序、客户端程序。如果不考虑元数据的安全性，我们会把远程对象的 dll 生成相同的两份，分别放到服务端和客户端。Remoting 在客户端的调用是很简单的，但调试起来就没有那么容易。因为客户端和服务器端分别属于不同的应用程序域，无法设置断点进行单步调试。如果了解 NUnit ，大家会知道 NUnit 也是不支持分布式应用程序的调试的，至少是支持得不够好。

所以，在实际做项目的过程中，我更倾向于先调用本地的对象，等调试成功后，再打开

Remoting 服务，调用远程对象，验证是否正确。举例来说，我要提供访问数据库的远程对象。我会先让该对象在本地运行，并调用其方法。如果一切正常，说明数据库的配置和连接均是正确的。然后再将该调用替换为远程对象。如果程序出错，则可以肯定是 Remoting 提供的服务出错了，或者是远程对象未按照 Remoting 的规定，没有派生

MarshalByRefObject ，或者未提供序列化特性。

最初使用了最愚蠢的办法，就是写两行调用，一个调用本地对象，一个调用远程对象。然后根据实际的情况，酌情考虑注释某一行代码。如此这般用了一段时间，终于觉得麻烦，迫使改变方法了。其实很简单，就是为客户端程序的主类中，多写一个构造函数而已，呵呵：）

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。
如要下载或阅读全文，请访问：
https://d.book118.com/356143044005011004