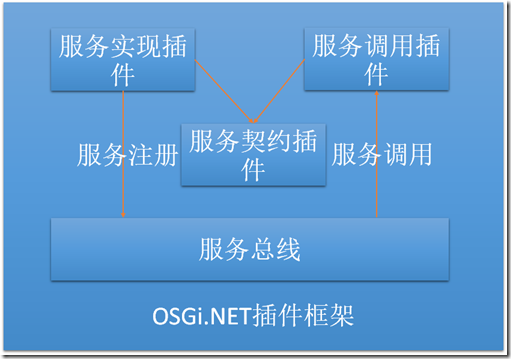
随笔-64  文章-0  评论-3202

[**分享在Linux下使用OSGi.NET插件框架快速实现一个分布式服务集群的方法**](http://www.cnblogs.com/baihmpgy/p/3765525.html)

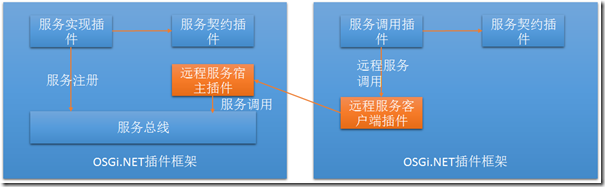
在这篇文章我分享了如何使用分层与模块化的方法来设计一个分布式服务集群。这个分布式服务集群是基于DynamicProxy、WCF和OSGi.NET插件框架实现的。我将从设计思路、目标和实现三方面来描述。

1 设计思路

首先，我来说明一下设计思路。我们先来看看目前OSGi.NET插件框架的服务。在这里，服务不是远程服务，它是轻量级的服务，由接口和实现类组成，如下图所示。服务契约插件定义了服务接口，服务实现插件向服务总线注册服务，服务调用插件利用服务契约（接口）从服务总线获取实现的服务并调用，服务实现插件和服务调用插件都依赖于服务契约，但二者并未有依赖。服务是插件间松耦合的调用方式。

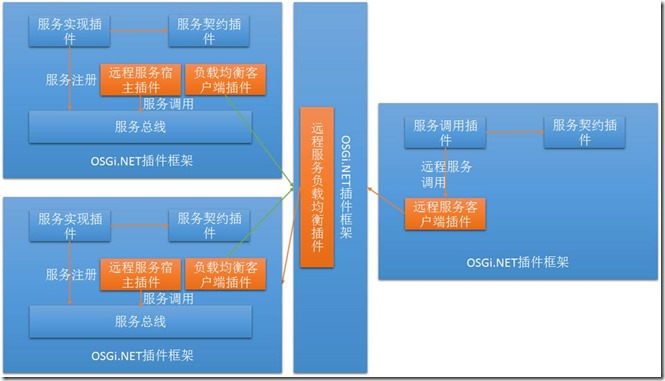
[](http://images.cnitblog.com/blog/21322/201406/031228596143550.png)

我们希望在不更改现有的通讯机制的情况下，将以前定义的在同一个框架下的服务能够直接不更改代码情况下，变成远程服务。此时，基于远程服务的通讯方式变成如下图所示的方式。

[](http://images.cnitblog.com/blog/21322/201406/031229005994149.png)

这时候，在不更改服务定义、服务注册的代码下，在OSGi.NET框架中安装一个远程服务宿主插件，它直接将服务总线的服务暴露成远程服务；OSGi.NET插件框架安装一个远程服务客户端插件，就可以使用服务契约来获取并调用远程服务。

接下来，我们希望能更进一步在不需要更改服务定义和注册代码情况下，来实现透明的集群支持。

[](http://images.cnitblog.com/blog/21322/201406/031229020366377.jpg)

在这里，我们引入了负载均衡插件。首先，一个OSGi.NET插件框架安装了远程服务负载均衡插件，它管理所有远程服务的负载状况，并为集群提供了统一的访问和负载均衡支持；接着，所有安装了远程服务宿主插件的OSGi.NET框架，会安装一个负载均衡客户端插件，它用于将远程服务注册到负载均衡器；服务调用端安装了远程服务客户端插件，它通过负载均衡器来调用远程服务。

这个思路可以简单描述如下：

A）本地服务 = OSGi.NET插件框架 + 服务契约插件 + 服务实现插件 + 服务调用插件；服务实现和服务调用在同一个OSGi.NET插件框架内，在同一个进程。

B）远程服务实现 = OSGi.NET插件框架 + 服务契约插件 + 服务实现插件 + 远程服务宿主插件，远程服务调用 = OSGi.NET插件框架 + 服务契约插件 + 远程服务客户端插件；服务实现和服务调用可以在不同的OSGi.NET插件框架，在不同进程内。

C）负载均衡器 = OSGi.NET插件框架 + 远程服务负载均衡插件，负载均衡远程服务实现 = OSGi.NET插件框架 + 服务契约插件 + 服务实现插件 + 远程服务宿主插件 + 负载均衡客户端插件，远程服务调用 = OSGi.NET插件框架 + 服务契约插件 + 远程服务客户端插件； 负载均衡器、远程服务、服务调用均可以在不同OSGi.NET插件框架和不同进程，远程服务可以在多台机器中，注册到负载均衡器。

2 设计目标

远程服务和负载均衡的实现基于模块化思路，如下图所示。

[](http://images.cnitblog.com/blog/21322/201406/031229033956161.jpg)

（1）不更改本地服务的定义、注册和使用方法；

（2）在本地服务的基础上，安装远程服务宿主插件，服务就暴露成远程服务；

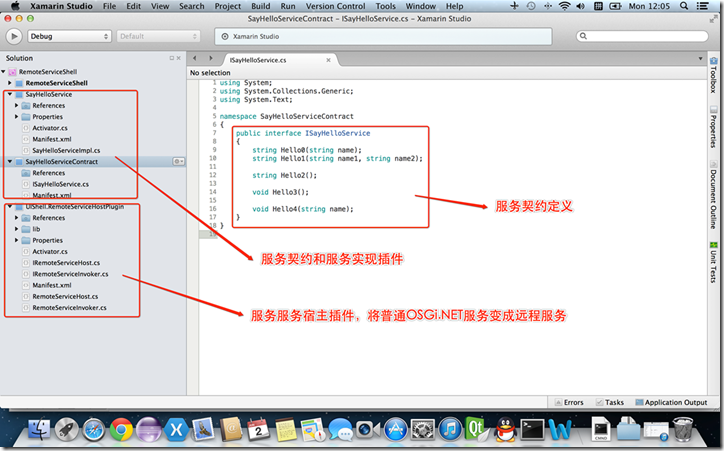
（3）在远程服务的基础上，安装负载均衡客户端插件和负载均衡器，远程服务就支持集群及负载均衡。

以一个简单的服务ISayHelloService为例，下面将描述如何通过以上方式来实现远程服务和服务集群。

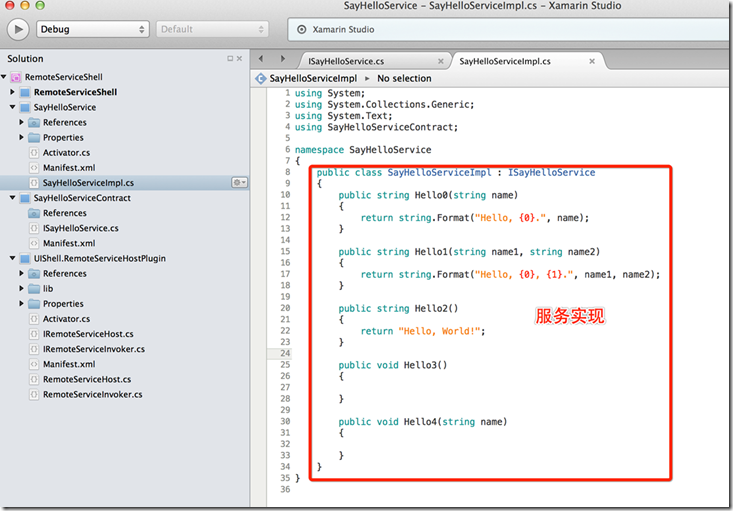
2.1 远程服务示例

2.1.1 远程服务注册及实现

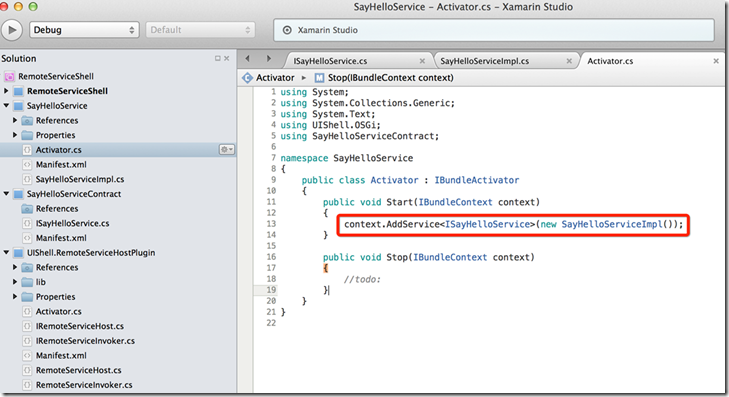
如下图所示，SayHelloServiceContract插件定义了一个ISayHelloService服务接口，SayHelloService定义了SayHelloServiceImpl服务实现，在OSGi.NET插件框架安装了UIShell.RemoteServiceHostPlugin插件，这样我们就将本地服务暴露成远程服务了。

[](http://images.cnitblog.com/blog/21322/201406/031229087394729.png)

下图是SayHelloServiceImpl服务的实现和服务注册。

[](http://images.cnitblog.com/blog/21322/201406/031226019276167.png)

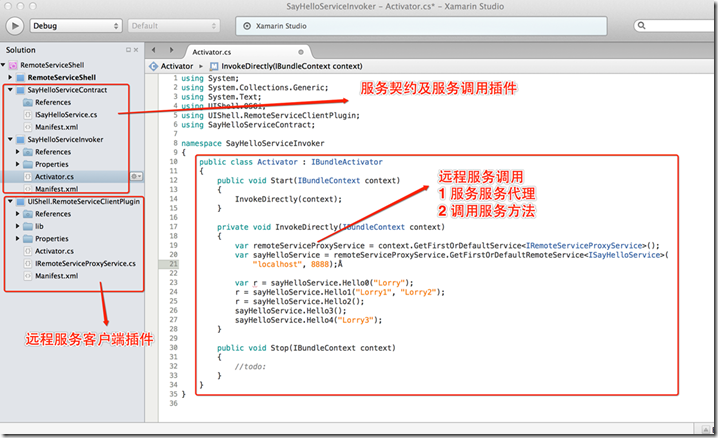
下图则是服务注册。

[](http://images.cnitblog.com/blog/21322/201406/031226069277306.png)

你可以发现，为了支持远程服务，我们仅仅是安装了一个远程服务宿主插件，而没有更改服务的实现和注册方法。

2.1.2 远程服务调用

远程服务调用如下所示，在OSGi.NET插件框架安装了远程服务客户端插件。服务调用插件使用服务契约，来调用远程服务。

[](http://images.cnitblog.com/blog/21322/201406/031226134428416.png)

调用远程服务的步骤为：

1 使用远程服务客户端插件的IRemoteServiceProxyService来获取远程服务；

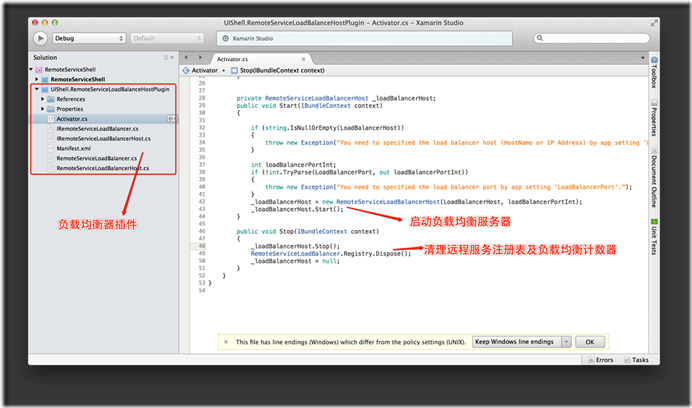
2 直接调用远程服务的方法。

因此，你可以发现，远程服务的定义和使用都非常的简单。接下来我们再看看负载均衡远程服务的使用。

2.2 负载均衡远程服务示例

2.2.1 负载均衡器

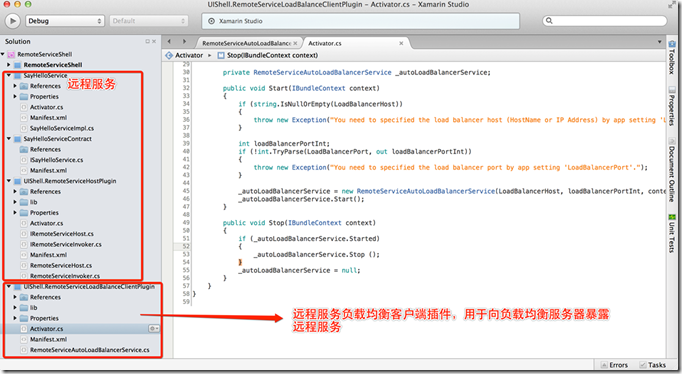
负载均衡器相当于远程服务注册表，它用于注册暴露远程服务的所有机器以及每一个机器每一个服务的负载均衡状况，并提供负载均衡支持。下图是负载均衡器的实现。

[](http://images.cnitblog.com/blog/21322/201406/031226192868898.png)

负载均衡器向外暴露一个固定的IP地址和端口号，用于注册远程服务，并提供负载均衡。

2.2.2 负载均衡远程服务

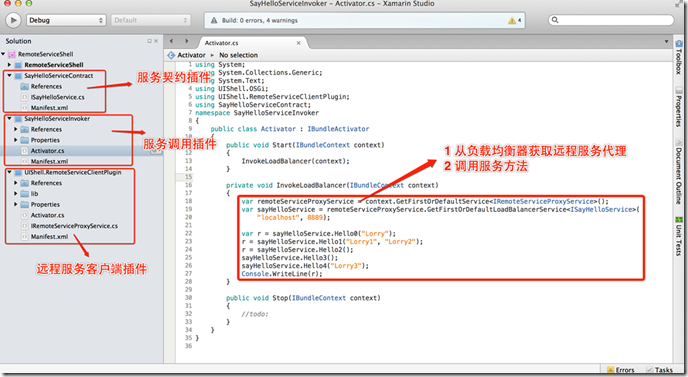
支持负载均衡的远程服务需要安装一个负载均衡客户端插件，如下所示。负载均衡客户端插件用于将远程服务注册到负载均衡器，从而，负载均衡器可以来管理远程服务的负载情况，当发生故障时可以实现负载转移和实现负载均衡。

[](http://images.cnitblog.com/blog/21322/201406/031226254117593.png)

这里，远程负载均衡器客户端插件会连接到负载均衡服务器，向其注册本机器的远程服务。

2.2.3 负载均衡远程服务调用

调用负载均衡远程服务与直接调用远程服务方法类似，如下图所示。它使用GetFirstOrDefaultLoadBalancerService接口来访问负载均衡器，获取经过负载均衡的远程服务。

[](http://images.cnitblog.com/blog/21322/201406/031226317862746.png)

你可以发现，调用负载均衡远程服务的方法也非常简单。下面，我来介绍一下如何实现。

3 设计实现

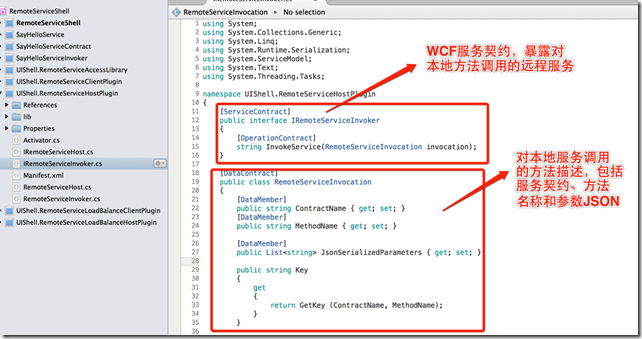
首先，我们先来看看远程服务的实现。

3.1 远程服务的实现

远程服务的实现可以归结为以下几点：（1）远程服务宿主插件用于暴露一个WCF服务，这个WCF服务相当于本地服务的Bridge，即客户端对远程服务的调用先中专到这个WCF服务，再由WCF服务来调用本地服务，然后返回给客户端；（2）客户端使用DynamicProxy为服务契约生成一个代理，对这个代理的方法调用将会被拦截，然后调用远程服务宿主插件的WCF服务，将调用结果再返回。

3.1.1 远程服务宿主插件实现

该插件首先定义了一个IRemoteServiceInvoker的WCF服务接口，这个接口及参数的定义如下。

[](http://images.cnitblog.com/blog/21322/201406/031226373024327.png)

它的作用就是通过调用这个WCF远程服务来调用OSGi.NET框架本地服务，达到将本地服务暴露成远程服务的目的。

这个WCF的实现代码如下所示，其目的就是对WCF的调用转换成对本地服务方法的调用并返回。

[复制代码](javascript:void(0);)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Text;

5 using System.Threading;

6 using System.Reflection;

7 using System.Threading.Tasks;

8 using fastJSON;

9 using Fasterflect;

10 using UIShell.OSGi.Utility;

11

12 namespace UIShell.RemoteServiceHostPlugin

13 {

14 public class RemoteServiceInvoker : IRemoteServiceInvoker

15 {

16 public static ReaderWriterLock Locker = new ReaderWriterLock();

17 public static Dictionary<string, System.Tuple<MethodInfo, Type[], MethodInvoker>> InvokerCache = new Dictionary<string, System.Tuple<MethodInfo, Type[], MethodInvoker>> ();

18

19 public string InvokeService(RemoteServiceInvocation invocation)

20 {

21 AssertUtility.NotNull(invocation);

22 AssertUtility.ArgumentHasText(invocation.ContractName, "service contract name");

23 AssertUtility.ArgumentHasText(invocation.MethodName, "service method name");

24

25 var service = Activator.Context.GetFirstOrDefaultService(invocation.ContractName);

26 string msg = string.Empty;

27 if(service == null)

28 {

29 msg = string.Format ("Remote Service '{0}' not found.", invocation.ContractName);

30 FileLogUtility.Warn (msg);

31 throw new Exception(msg);

32 }

33

34 System.Tuple<MethodInfo, Type[], MethodInvoker> invokerTuple;

35 using (var locker = ReaderWriterLockHelper.CreateReaderLock (Locker))

36 {

37 InvokerCache.TryGetValue (invocation.Key, out invokerTuple);

38 }

39

40 if (invokerTuple == null)

41 {

42 Type serviceType = service.GetType ();

43

44 var serviceMethodInfo = serviceType.GetMethod (invocation.MethodName);

45 if (serviceMethodInfo == null)

46 {

47 msg = string.Format ("The method '{1}' of the remote service '{0}' not found.", invocation.ContractName, invocation.MethodName);

48 FileLogUtility.Warn (msg);

49 throw new Exception (msg);

50 }

51

52 if (invocation.JsonSerializedParameters == null) {

53 invocation.JsonSerializedParameters = new List<string> ();

54 }

55

56 var parameterInfos = serviceMethodInfo.GetParameters ();

57 if (invocation.JsonSerializedParameters.Count != parameterInfos.Length)

58 {

59 msg = string.Format ("The parameters count is not match with the method '{0}' of service '{1}'. The expected count is {2}, the actual count is {3}.", invocation.MethodName, invocation.ContractName, parameterInfos.Length, invocation.JsonSerializedParameters.Count);

60 FileLogUtility.Warn (msg);

61 throw new Exception (msg);

62 }

63

64 var parameterTypes = new Type[parameterInfos.Length];

65 for (int i = 0; i < parameterInfos.Length; i++)

66 {

67 parameterTypes [i] = parameterInfos [i].ParameterType;

68 }

69

70 try

71 {

72 var methodInvoker = serviceType.DelegateForCallMethod (invocation.MethodName, parameterTypes);

73

74 invokerTuple = new System.Tuple<MethodInfo, Type[], MethodInvoker> (serviceMethodInfo, parameterTypes, methodInvoker);

75

76 using (var locker = ReaderWriterLockHelper.CreateWriterLock (Locker))

77 {

78 if (!InvokerCache.ContainsKey (invocation.Key))

79 {

80 InvokerCache [invocation.Key] = invokerTuple;

81 }

82 }

83 }

84 catch(Exception ex)

85 {

86 msg = string.Format ("Failed to create delegate method for the method '{0}' of service '{1}'.", invocation.MethodName, invocation.ContractName);

87 FileLogUtility.Warn (msg);

88 FileLogUtility.Warn (ex);

89 throw new Exception (msg, ex);

90 }

91 }

92

93 var paramters = new object[invokerTuple.Item2.Length];

94

95 for(int i = 0; i < invokerTuple.Item2.Length; i++)

96 {

97 try

98 {

99 paramters[i] = JSON.ToObject(invocation.JsonSerializedParameters[i], invokerTuple.Item2[i]);

100 }

101 catch(Exception ex)

102 {

103 msg = string.Format ("Failed to unserialize the '{0}'th parameter for the method '{1}' of service '{2}'.", i + 1, invocation.MethodName, invocation.ContractName);

104 FileLogUtility.Warn (msg);

105 FileLogUtility.Warn (ex);

106 throw new Exception (msg, ex);

107 }

108 }

109

110 try

111 {

112 return JSON.ToJSON(invokerTuple.Item3(service, paramters));

113 }

114 catch(Exception ex)

115 {

116 msg = string.Format("Failed to invoke the method '{0}' of service '{1}'.", invocation.MethodName, invocation.ContractName);

117 FileLogUtility.Warn (msg);

118 FileLogUtility.Warn (ex);

119 throw new Exception (msg, ex);

120 }

121 }

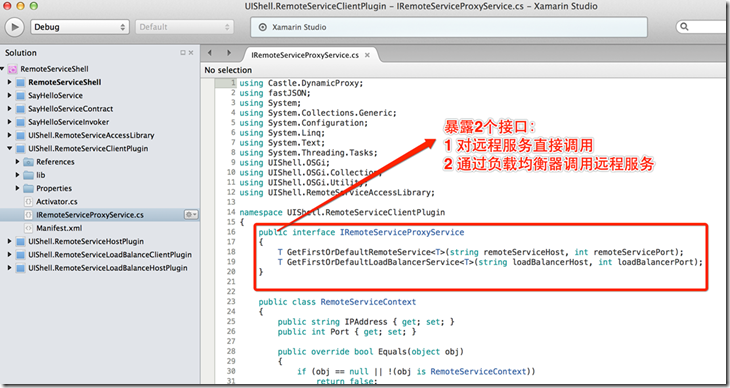
122 }

123 }

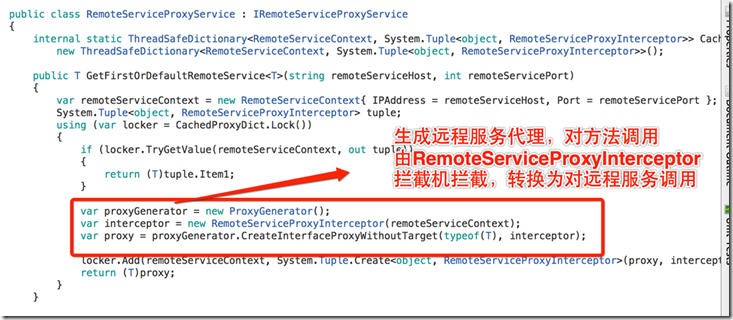
[复制代码](javascript:void(0);)

3.1.2 远程服务客户端插件的实现

接下来，我们看看远程服务客户端插件的实现。它定义了一个IRemoteServiceProxyService服务，暴露了两个接口分别用于对远程服务和负载均衡远程服务的调用。

[](http://images.cnitblog.com/blog/21322/201406/031226442399608.png)

该服务的远程服务获取实现如下所示。

[](http://images.cnitblog.com/blog/21322/201406/031226522705987.png)

它仅仅时通过DynamicProxy创建远程服务代理类，此时，对代理类方法的调用会转换成远程服务调用。下面看看拦截机的实现。

[复制代码](javascript:void(0);)

1 class RemoteServiceProxyInterceptor : IInterceptor, IDisposable

2 {

3 private RemoteServiceContext \_remoteServiceContext;

4 private RemoteServiceClient \_remoteServiceClient;

5 public RemoteServiceProxyInterceptor(RemoteServiceContext context)

6 {

7 \_remoteServiceContext = context;

8 \_remoteServiceClient = new RemoteServiceClient(\_remoteServiceContext.IPAddress, \_remoteServiceContext.Port);

9 \_remoteServiceClient.Start();

10 }

11 public void Intercept(IInvocation invocation)

12 {

13 try

14 {

15 var jsonParameters = new List<string>();

16 foreach (var param in invocation.Arguments)

17 {

18 jsonParameters.Add(JSON.ToJSON(param));

19 }

20

21 var resultJson = \_remoteServiceClient.Invoke(invocation.Method.DeclaringType.FullName, invocation.Method.Name, jsonParameters);

22

23 if (!invocation.Method.ReturnType.FullName.Equals("System.Void"))

24 {

25 invocation.ReturnValue = JSON.ToObject(resultJson, invocation.Method.ReturnType);

26 }

27 else

28 {

29 invocation.ReturnValue = null;

30 }

31 }

32 catch(Exception ex)

33 {

34 FileLogUtility.Error (string.Format("Failed to invoke the remote service 'Remote Service: {0}, Method: {1}.'",

35 invocation.Method.DeclaringType.FullName, invocation.Method.Name));

36 throw;

37 }

38 }

39

40 public void Dispose()

41 {

42 if (\_remoteServiceClient != null)

43 {

44 \_remoteServiceClient.Stop();

45 \_remoteServiceClient = null;

46 }

47 }

48 }

[复制代码](javascript:void(0);)

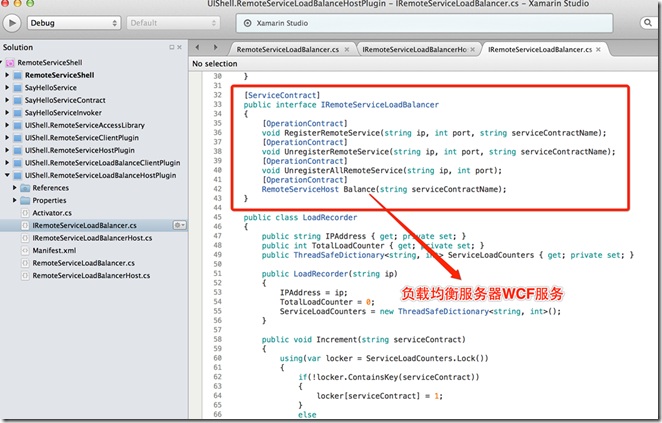
拦截机的代码很简单，对远程服务代理类方法的调用将直接转换成对远程服务宿主插件的WCF服务的调用。下面看看负载均衡远程服务的实现。

3.2 负载均衡远程服务的实现

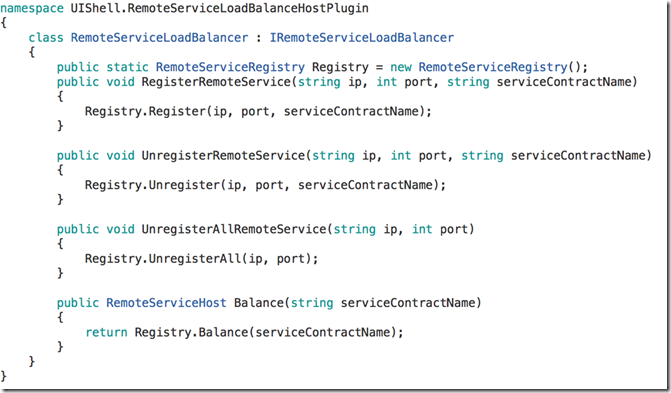
有了以上的技术，关于负载均衡远程服务的实现，就简单多了。负载均衡远程服务的目的就是将所有远程服务统一在负载均衡服务器进行注册，并实现负载的动态管理。因此，需要在远程服务基础上，创建一个负载均衡服务器和负载均衡客户端。负载均衡服务器用于管理所有远程服务及提供远程服务的机器，管理所有远程服务的负载情况，并实现负载均衡及故障转移；负载均衡客户端的目的时将远程服务注册到负载均衡器，并且当远程服务关闭时从负载均衡器卸载。下面，看看负载均衡器的实现。

3.2.1 负载均衡器实现

负载均衡服务器暴露了如下WCF服务。这个服务用于提供远程服务注册和卸载以及负载均衡请求。

[](http://images.cnitblog.com/blog/21322/201406/031227052398992.png)

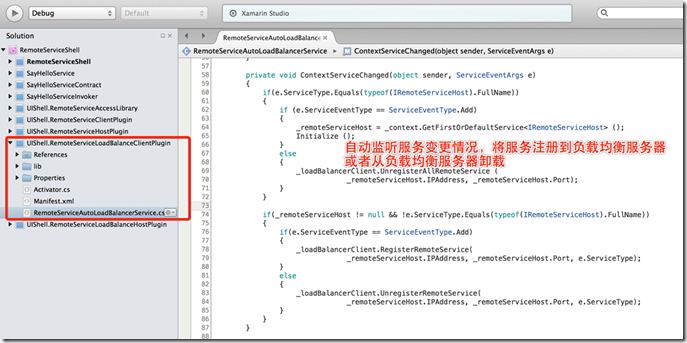
这个服务的实现如下所示。

[](http://images.cnitblog.com/blog/21322/201406/031227106927059.png)

它使用远程服务注册表来实现远程服务的管理和负载均衡的实现。

3.2.2 负载均衡客户端的实现

负载均衡客户端的目的时实现远程服务的注册与卸载，通过该插件将远程服务暴露到负载均衡服务器。这样服务调用者就可以通过负载均衡器来调用远程服务。

[](http://images.cnitblog.com/blog/21322/201406/031227190679867.png)

3.2.3 负载均衡远程服务调用

负载均衡远程服务的调用方式的实现和远程服务类似。它由远程服务代理服务的GetFirstOrDefaultLoadBalancerService接口来实现。

[](http://images.cnitblog.com/blog/21322/201406/031227264894834.png)

该接口的实现如下所示，主要时创建代理和方法拦截机。

[](http://images.cnitblog.com/blog/21322/201406/031227387867210.png)

这个方法调用拦截机会将方法调用转化为：（1）从负载均衡服务器获取均衡的远程服务主机；（2）直接调用该远程服务主机的服务，如果调用失败则尝试进行重新负载均衡。其实现如下所示。

[复制代码](javascript:void(0);)

1 class RemoteServiceLoadBalancerProxyInterceptor : IInterceptor, IDisposable

2 {

3 private string LoadBalancerHost

4 {

5 get

6 {

7 return ConfigurationSettings.AppSettings["LoadBalancerHost"];

8 }

9 }

10

11 private string LoadBalancerPort

12 {

13 get

14 {

15 return ConfigurationSettings.AppSettings["LoadBalancerPort"];

16 }

17 }

18

19 private LoadBalancerContext \_remoteServiceLoadBalancerContext;

20 private RemoteServiceClient \_remoteServiceClient;

21 private RemoteServiceLoadBalancerAccessClient \_remoteServiceLoadBalancerAccessClient;

22 private bool \_initialized;

23

24 public RemoteServiceLoadBalancerProxyInterceptor(LoadBalancerContext context)

25 {

26 \_remoteServiceLoadBalancerContext = context;

27

28 if (string.IsNullOrEmpty(LoadBalancerHost))

29 {

30 throw new Exception("You need to specified the load balancer host (HostName or IP Address) by app setting 'LoadBalancerHost'.");

31 }

32

33 int loadBalancerPortInt;

34 if (!int.TryParse(LoadBalancerPort, out loadBalancerPortInt))

35 {

36 throw new Exception("You need to specified the load balancer port by app setting 'LoadBalancerPort'.");

37 }

38

39 try

40 {

41 \_remoteServiceLoadBalancerAccessClient = new RemoteServiceLoadBalancerAccessClient (LoadBalancerHost, loadBalancerPortInt);

42 \_remoteServiceLoadBalancerAccessClient.Start ();

43 }

44 catch(Exception ex)

45 {

46 FileLogUtility.Error (string.Format("Faild to connect to load balancer '{0}'.", \_remoteServiceLoadBalancerContext));

47 FileLogUtility.Error (ex);

48 throw;

49 }

50 }

51

52 private bool Initialize(string serviceContractName)

53 {

54 if(\_remoteServiceClient != null)

55 {

56 \_remoteServiceClient.Stop();

57 }

58

59 RemoteServiceHost remoteHost = null;

60 try

61 {

62 remoteHost = \_remoteServiceLoadBalancerAccessClient.Balance(serviceContractName);

63 FileLogUtility.Inform(string.Format("Get the remote service host '{0}' by load balancer '{1}'.", remoteHost, \_remoteServiceLoadBalancerContext));

64 }

65 catch(Exception ex)

66 {

67 FileLogUtility.Error (string.Format("Faild to get a remote service host by load balancer '{0}'.", \_remoteServiceLoadBalancerContext));

68 FileLogUtility.Error (ex);

69 return false;

70 }

71 if (remoteHost != null)

72 {

73 \_remoteServiceClient = new RemoteServiceClient (remoteHost.IPAddress, remoteHost.Port);

74 try

75 {

76 \_remoteServiceClient.Start ();

77 return true;

78 }

79 catch(Exception ex)

80 {

81 FileLogUtility.Error (string.Format("Failed to connect to the remote service host '{0}' by using load balancer '{1}'.", remoteHost, \_remoteServiceLoadBalancerContext));

82 }

83 }

84

85 return false;

86 }

87

88 public void Intercept(IInvocation invocation)

89 {

90 var serviceContractName = invocation.Method.DeclaringType.FullName;

91 if (!\_initialized)

92 {

93 \_initialized = Initialize (serviceContractName);

94 if (!\_initialized)

95 {

96 invocation.ReturnValue = null;

97 return;

98 }

99 }

100

101 var jsonParameters = new List<string>();

102 foreach (var param in invocation.Arguments)

103 {

104 jsonParameters.Add(JSON.ToJSON(param));

105 }

106

107 int tryTimes = 1;

108

109 for (int i = 0; i < tryTimes; i ++ )

110 {

111 try

112 {

113 var resultJson = \_remoteServiceClient.Invoke(serviceContractName, invocation.Method.Name, jsonParameters);

114

115 if (!invocation.Method.ReturnType.FullName.Equals("System.Void"))

116 {

117 invocation.ReturnValue = JSON.ToObject(resultJson, invocation.Method.ReturnType);

118 }

119 else

120 {

121 invocation.ReturnValue = null;

122 }

123 return;

124 }

125 catch(Exception ex)

126 {

127 FileLogUtility.Error (string.Format("Failed to invoke the remote service 'Remote Service: {0}, Method: {1}.'",

128 serviceContractName, invocation.Method.Name));

129 FileLogUtility.Error (ex);

130 if (i == tryTimes)

131 {

132 throw;

133 }

134 if (!((\_initialized = Initialize (serviceContractName)) == true)) // 重新Balance

135 {

136 throw;

137 }

138 }

139 }

140 }

141

142 public void Dispose()

143 {

144 if (\_remoteServiceClient != null)

145 {

146 \_remoteServiceClient.Stop();

147 \_remoteServiceClient = null;

148 }

149 }

150 }

[复制代码](javascript:void(0);)