ASP.NET Web Service or .NET Remoting

Understanding the basic structure of Microsoft .NET Remoting remote administration and Microsoft ASP.NET Web services can help you get cross-process communication. How these two technologies work and select them accordingly

Summary : Understanding the basic structure of the Microsoft .NET Remoting remote administration method and Microsoft ASP.NET Web services services can help you get cross-process communication activity. How these two technologies work and how to choose them for your application is the main content of today's lesson.

overview

Over time, building the application as a set of distributed components into the general network and working together as part of the overall program is becoming increasingly popular. This form is often referred to as "object-component" technique, such as Distributed Component Object Model (DCOM) of Microsoft, Common Object Request Broker Architecture (CORBA) of Sun's Object Management Group, Remote Method Invocation (RMI). They provide a flexible, reliable structure to timely capture the necessary development of applications.

Although this component-based technology works well in the Intranet environment, there are two significant problems with the Internet. The first is that they do not follow the partial method. All operations are based on objects but do not provide specific details such as managing the operation cycle, supporting the building and supporting inheritance levels. Secondly, it is more important to focus on the typical form of RPC communication which leads to the construction of systems that incorporate tightly around open invocations of the object method.

In contrast, browser-based Web applications are loosely coupled and follow each component clearly. They communicate using the HTTP protocol to convert MIME type data into a variety of different formats. Web services adapt to the traditional Web programming model to be used for all types of applications, not just the browser base type. They exchange SOAP messages using HTTP and many other Internet protocols. Because Web services are based on technology standards such as HTTP, XML, SOAP and WSDL, when they want to bring application functionality to the Internet, they also have to depend on programming languages, platforms and devices.

The ASP.NET Web service basic structure provides a simple API for Web services based on how to map SOAP information to method invocations. The finishing process by providing a very simple programming model based on the form of mapping SOAP information exchanges into each invokes its own method. ASP.NET Web services clients don't need to know the platform, object model or programming language used to build them. The services themselves do not need to know anything about the client sending information to them. The only requirement is that both parties must agree on the type of SOAP information format to be created and used, such as the definition of a Web service shortened expression of type WSDL and XML Schema (XSD).

The .NET Framework supports two separate distributed programming models: Web services and distributed

objects, which often cause confusion for developers. When should I use the ASP.NET Web service and when should I use .NET Remoting? To get an answer to this question, you first need to understand how the mechanisms of both technologies work.

Serial and Metadata

All communication distribution activities ultimately work only on two tasks: arranging the session of the programming data type into the information that can be sent over the network and providing a description of that information. . Previously, this process was done using some serial or sequential forms, which later used metadata forms. The basic difference between ASP.NET Web services and .NET Remoting is how they order data into the information section and how the format they choose for metadata.

ASP.NET Web Services, XmlSerializer and XSD

ASP.NET Web Services is based on the **System.Xml.Serialization.XmlSerializer** class to sort data into SOAP information or retrieve data from it at runtime. With metadata, they create WSDL and XSD definitions that describe what the information contains. Based on the pure WSDL and XSD, the metadata of ASP.NET Web Services is mobile. They describe the data structure in a way that makes the toolkit Web services on many other platforms and many other programming models still understandable. In some cases it is also tied to many types taken from a Web service. **XmlSerializer** only sorts the things described in XSD. **XmlSerializer** does not arrange object graphics and limited support for storage types.

Although these limits seem insignificant to a distributed object by layer, they help ensure consistency between operations with other Web service frameworks - the primary purpose of a loose-fitting Web service model. The support for interoperability is enhanced with a rich set of optional attributes, allowing you to annotate the data type to control how **XmlSerializer** sorts. As a result, you have very useful control over the form of XML created when an object is ordered. In addition, an ASP.NET-based Web service can be modified to describe its information under the term XSD literal or SOAP coding principles. XSD literal is set by default and will be standardized later. The SOAP encryption support function is arranged for all parts with the toolkit available. This is very useful, especially when you need to contact an existing Web service or client, using predefined news.

.NET Remoting, IFormatter and Common Language Runtime (**Common language** implementation set)

.NET Remoting is based on the implementation of the closed type of the **IFormatter** interface used under the **System.Runtime.Serialization** mechanism to sort the data to and from the exchange information sections. There are two standard formats: **System.Runtime.Serialization.Formatters.Binary.BinaryFormatter** and **System.Runtime.Serialization.Formatters.Soap.SoapFormatter** . **BinaryFormatter** and **SoapFormatter** , like the name, order the types in turn in binary and SOAP format.

With metadata, .NET Remoting is based on Common Language Runtime components, including all information related to the executable data type and how to export it through remapping. Based on the metadata components of the metadata, it is easier to maintain the reliability of the entire system-style runtime. As a result, when .NET Remoting finishes sorting data, it will have all members, including public (public members) and private (private members) of a class; controls the graphical objects correctly and supports all types of storage (like **System.Collections.Hashtable**). However, based on run-time metadata also limits the ability of .NET Remoting system. A client must understand the .NET structure to communicate with the .NET Remoting endpoint. In addition, with closed formats, the .NET Remoting layer supports closed channels that will detail the description of how the information is sent. There are two standard channels, one for raw TCP and one for HTTP.

Information can be sent via one of two channels, depending on the format.

Remoting and Web Service

The presence of a SOAP format set and an HTTP channel raises the question: Can .NET Remoting be used to build Web services? The answer is yes and no. Stack of standard Web service technology is based not only on SOAP baseline information but also on the description of the WSDL and XSD background of that information. The process of completing Remoting can actually generate WSDL definitions that describe production information and endpoint usage. However, many problems arise (will be discussed in more detail below).

First, created WSDL files always describe information in terms of SOAP coding instead of XSD literal. Today, this is not a problem. But people are expected to have more and more tools focused entirely on this program.

Second, the generated WSDL files including the extension are .NET Remoting-specific. For example, here is a simple class that uses .NET Remoting to explain its operation.

```
public class Methods: MarshalByRefObject
{
// The method hi?n th?i ?ã th? th?i gian hi?n th?i và th?i gian
public string Now ()
{
return System.DateTime.Now.ToString ();
}
}
```

If you create WSDL from this class, the join information includes specific .NET Remoting details, as shown below:

```
transport = 'http:///schemas.xmlsoap.org/soap/http'/>



'http://schemas.microsoft.com/clr/nsassem/RemSoap.Methods/methods#Now'/>

.

.
```

These extension elements are valid because WSDL specifies extended support. Any good Web service toolkit that doesn't understand them will simply ignore them. However, there are some toolkit Web services that are not ignored. For example, a Remoting endpoint returns Microsoft® ADO.NET **System.Data.DataSet** collection type data.

```
public class Methods: MarshalByRefObject
{
```

```
public System.Data.DataSet GetEmptyDataSet ()
{
return new System.Data.DataSet ();
}
}
```

Here is the definition of the WSDL generated for the output of this method:

Typically, a WSDL information indicates the types defined in a specific namespace using XML Schema. However, in this case the prefix ns3 namespace applies to the **DataSet** type not defined in XSD. Instead it is defined to be hidden via runtime. The ns3 prefix in this example represents an XML namespace defined by the URL:

http://schemas.microsoft.com/clr/nsassem/System.Data/System.Data%2C%20Version%3D1.0.3300.0%2C%20Culture%

Users of the above type of WSDL definition are often implied to understand the significance of this "famous" URI, a long name consisting of four parts of a specific runtime in the .NET Framework. This WSDL type is suitable for .NET Remoting clients because they can create multiplexed proxies with the correct information for the orderly process. However, other Web service toolkits (including ASP.NET) do not understand this URI address and wait for the frame definition for the **DataSet** type. This WSDL becomes useless.

So again, there are more questions, can you use .NET Remoting to build Web services? Yes, speak seriously, you can. But who can not use .NET Remoting can use them? The answer is possible, if you carefully reduce the endpoint to only the most basic data types and semantics. Specifically, if you want to manipulate other Web service toolkits, you need to limit the parameters to simple built-in types and your own data types (do not use .NET Framework types like **DataSet**), and avoid objects, client events are enabled. In short, if you are interested in this method, you need to limit yourself to the same set of features that ASP.NET Web services support.

If it is still not good, use ASP.NET Web Service, because the correct ASP.NET Web Service is designed for this purpose.

Designing distributed applications: ASP.NET Web Service duel .NET Remoting

ASP.NET Web Services focuses more on XML Schema-style systems and provides simple programming models that cross-distribute to each platform. .NET Remoting focuses on the runtime system and provides a more general programming model with more restricted form. The fundamental difference in these two technologies is to identify the main factor in the intended use. There are also a large number of other design elements such as transport protocols, host programs, security, enforcement, status management and transaction support to consider, .

Transport protocols and host programs

Technically, SOAP does not manage HTTP as a transport protocol. But clients can only access Web services that use ASP.NET over HTTP, because it only supports the ASP.NET transport protocol. Services are invoked through IIS and executed in the ASP.NET workgroup program (aspnet_wp.exe).

.NET Remoting gives you a flexible way to manage remote objects of any type of application, such as Windows Form, Windows Service management program, console application or ASP.NET group program. As mentioned above, .NET Remoting provides two transport channels: TCP and HTTP. Both channels provide communication between programs that send and receive separately using sockets.

It is also possible to integrate HTTP channels with IIS and ASP.NET group programs by following several important reasons. First, there is only one way to automatically start a .NET Remoting endpoint when a client requests it. Completion process .Net Remoting does not use DCOM Service Control Manager (SCM) to start remote servers. You can only get remote objects in a separate program when the program is running. At the same time, make sure they are safe, for example, line A cannot activate an object after line B starts the process. If you use ASP.NET remote objects, you can see that the Aspnet_wp.exe group program has both the auto-start function and the secure stream. Secondly, as described in the next section, the ability to integrate with IIS is how you can secure .Net Remoting calls to cross programs.

Both the basic structure of ASP.NET Web Services and .Net Remoting are scalable. You can filter information inside and outside the contour, control aspects of the sort order and general metadata. With .Net Remoting, you also have to deploy separate sets of formats and channels.

Security

Since ASP.NET Web Services uses over HTTP, they are integrated with Internet security infrastructure. ASP.NET leverages security components in conjunction with IIS, providing powerful support for HTTP standard authentication frameworks such as Basic, Digest, digital certificate authentication, and even Microsoft's .NET Passport. (You can use Windows Integrated authentication, but only for clients in a trusted domain.) An advanced improvement when using HTTP authentication frames does not require changing the code in the Web service, IIS performs authentication before ASP.NET Web Services is called. ASP.NET also provides support for .NET Passport-based authentication programs and other custom authentication frameworks. Components that support ASP.NET access control are based on destination URL paths and by integrating them with the basic .NET Code Access Security (CAS) structure. SSL can be used to secure private communication channels over wires.

Although standard transport level techniques that ensure Web service security are quite effective, they only stop there. In many aggregate environments such as multi-service Web services in other trusted domain systems (domains), you must build a variety of customized solutions. Microsoft and many others are still working on a set of security specifications built on the extension of SOAP information to provide information level security capabilities. One of those technologies is XML Web Services Security Language (WS-Security), defined as a framework for reliably conveying information, information integrity and information security.

As noted in the previous section, the .Net Remoting completion process does not guarantee cross-program referrals in the general case. A .Net Remoting endpoint introduced into IIS with ASP.NET can promote all the same security components available to ASP.NET Web services, including secure communication support via SSL fences. If you are using TCP or HTTP for programs instead of aspnet_wp.exe, you must manually authenticate, license and manage security mechanisms.

A traditional security concept is the ability to execute code from a trusted environment somewhat without having to change the default security mechanism. ASP.NET Web Service proxy clients work in such environments. But .Net Remoting proxy does not. To use a .Net Remoting proxy from a somewhat trusted environment, you need to have special ordering rights that do not provide the download code from the default Intranet or Internet. If you

want to use .Net Remoting client from within a somewhat trusted environment, you must edit the default security mechanism for loading code from these areas. In many cases, when you are connecting to systems from a client running in the sandbox (like downloading Windows Forms applications for example), using ASP.NET Web Services will be simpler because you do not need to change the engine. Its default security.

Status management

The ASP.NET Web Services model uses a stateless service structure as the default. It does not correlate with multiple calls from the same user. In addition, each time a client invokes an ASP.NET Web service, the new object will be created to serve the request. This object is discarded after the method call completes. To maintain the status between requests, you can use the same techniques as ASP.NET pages (eg "Session and Application" properties) or you can deploy your own solutions.

.Net Remoting supports a lot of status management options, which may or may not be equivalent to many calls from a user, depending on the object's life time schema. **SingCall** objects have no state (like the objects used to invoke ASP.NET Web Services), Singleton objects share status for all client and client-enabled objects that maintain state on each Base client (with all possible elasticity and reliability).

Enforcement

In "coarse execution" terms, the complete .Net Remoting provides the fastest communication capability when you use TCP channels and binary formats. In most of the tests offered to compare the relational performance of ASP.NET Web Services and .Net Remoting, ASP.NET Web services outperformed .Net Remoting endpoints using the SOAP format with HTTP or TCP channels. More interesting than ASP.NET and .Net Remoting endpoints use the same binary format and HTTP channel in execution.

Business services

An ASP.NET Web Service or an .Net Remoting object can use local transactions to work in conjunction before a single database. If you need to work with a lot of resources beforehand, it can use .Net Enterprise Services (aka COM +) to declare transactions (a DTC distribution transaction is managed by the COM + program). Also note that only one, or ASP.NET Web services, or .Net Remoting supports declarative transaction transmitting. Therefore, one of the two endpoint types cannot inherit the transaction through the cross-program call.

This is not really necessary. In general, a transaction declaration is usually more expensive than a local transaction. Even if you pass it across the program boundary, it's still more expensive. If you still need this functionality, the easy solution is to deploy a class derived from **System.EnterpriseServices.ServicedComponent** in a .NET Enterprise Services server application. Cross-program calls to objects of this type will be controlled by DCOM to ensure that the transaction context transfer process takes place accordingly. The more difficult solution is to use lower-level APIs to deliver hand-crafted parts.

You should also note that the layer-distributed transaction model is generally not suitable for loose-matched Web services. The model based on offset transactions, is that transactions that recover many of the overlooked operations of other transactions are better because they have fewer strict constraints. There is a common agreement between Web service providers, including Microsoft, which needs to use many flexible transaction models in Web service space, and there are many ongoing activities in this space. . There is currently no standard method for Web service transactions, so you can deploy your own compensation framework, using appropriate

or local declarative transactions.

Choose the type of structure

If you are designing a distributed distribution application on .NET, you must consider all the issues discussed above and outline the diagram of that system structure. It is generally easier than you think. Some cases require special modifications, separate techniques are required. Here we provide some general assumptions, often simpler for you.

The first is to use the default ASP.NET Web Services. They are simpler in deployment and use. They provide the greatest possible way for client platforms. ASP.NET Web Services' proxy client code can be invoked from code running in a sanbox by default security mechanism.

If you need a more traditional distributed object model with complete CLR-type reliability, you don't need to implement the other platform components that control the configuration of both the client and server, like .Net Remoting. If you choose .Net Remoting, it is better to combine HTTP channels with IIS and ASP.NET. If not, you will have to build a program to manage the program operation cycle and its own security facility structure. To be like that, .Net Remoting requires a .NET client. You should use a binary format instead of the SOAP format. Interoperability between parts is not a problem and performance will get better attention.

Finally, use Enterprise Services (COM +) when you need declarative transactions. If you use ServicedComponents, deploy them in the default library applications for better execution. Deploy them in server applications if they need to run on remote machines. (You can also consider COM + server applications if you need to execute code with another program security token, rather than just using aspnet_wp.exe, even if used on the same machine).

Here are three common structures according to the above methods:

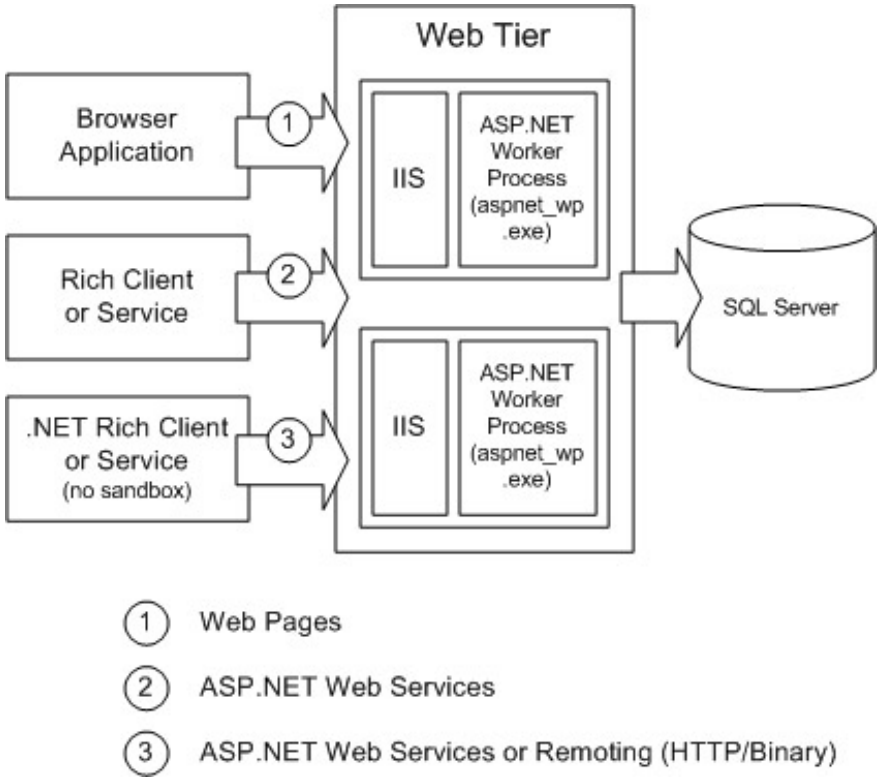

Figure 1. Simple 3-storey architecture

Figure 1 depicts a simple 3-tier structure with a Web server zone supporting multiple different clients. All server code executed in the ASP.NET group program, aspnet_wp.exe. Three different client types access the server using HTTP. Browser-based clients use the ASP.NET website; rich clients (such as Windows Forms applications, 6 Microsoft® Visual Basic® applications) and other Web services using ASP.NET Web Services; Rich .NET clients (such as Windows Forms applications) and Web services using ASP.NET Web Services or .Net Remoting with HTTP channels and binary formats (assuming it is not in the sandbox) as expected.

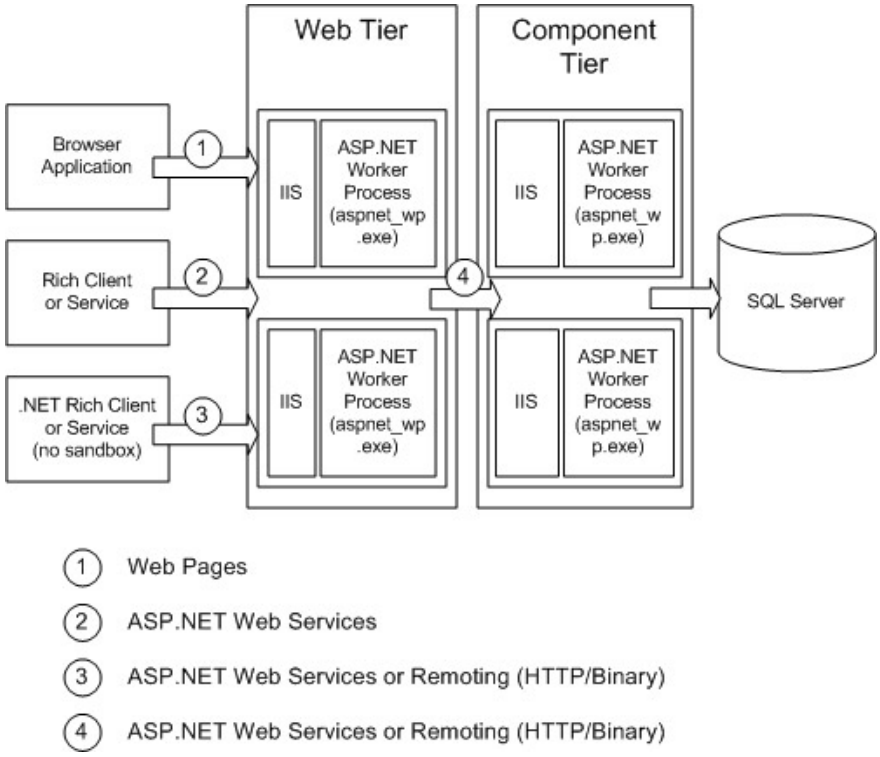

Figure 2. Multi-tier architecture using ASP.NET

Some very large applications use second models to operate offload from the outside of Web servers. This structure is depicted in Figure 2. Note that in this case, the second layer also uses functionality through ASP.NET.

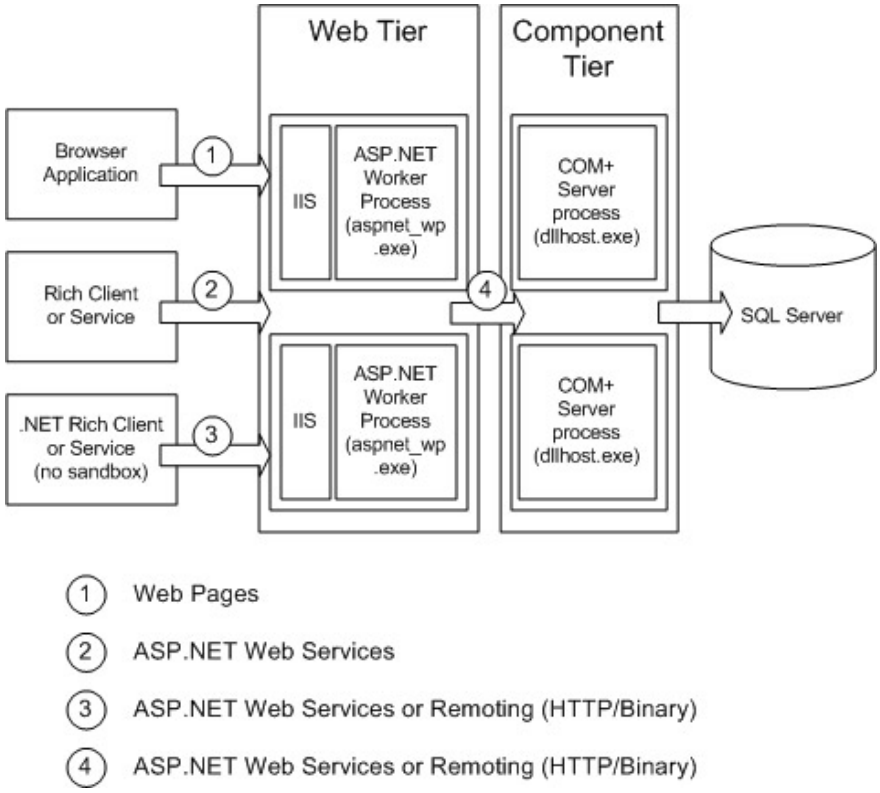

Figure 3. Multi-tier architecture using Enterprise Services (COM +)

Figure 3 depicts another description of a multistory architecture. The second layer exploits the feature ServicedComponents deployed in COM +.

Of course this is not the entire .NET Framework architecture supported. But they provide the foundation that helps you develop and build your own system.

Conclude

Although both the .Net Remoting and ASP.NET Web Services structures allow cross-process communication. But each type is designed differently, depending on the direction they use. ASP.NET Web Services provides a simple, wide-ranging programming model. .Net Remoting provides a more general programming model but the scope is much restricted. It is important that you understand how these two technologies work to choose the most appropriate method for your application. One of two cases can use IIS and ASP.NET to manage the program operation cycle and provide a general level of security in some cases.