Google Protocol Buffers 之.Net应用

吴剑 2011-2-22

原创文章，转载必需注明出处：[http://www.cnblogs.com/wu-jian/](http://www.cnblogs.com/wu-jian)

前言

最近接到一个跨平台的测试项目，服务端Linux，是Java开发的一系列Socket接口，客户端Windows，所以准备用.Net。本想这种 跨主流平台的Socket通信应该不成问题，但随着代码进程，随着一次次反复调试，我发现我错了。花了一周时间至今两者仍呈现北方网通和南方电信的姿态。

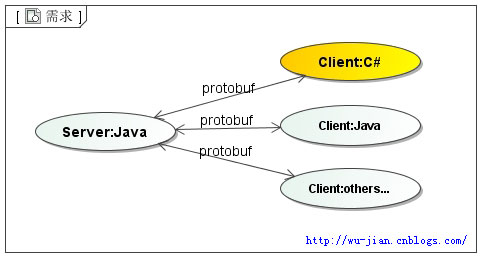
不过总有意外惊喜，过程中认识了Protocol Buffer，比XML、比JSON更为强悍，语言无关、平台无关、更小的存储、更少的歧义、更高的性能，其实Google一直在贡献，不论是Copy Left的还是Copy Right的，回头看看我们的百度，抄IM抄商城抄游戏抄视频抄房地产，还有搜索永远排第一却打不开的百度文库，印象中JQuery盛行N久之后百度开源 了一个JS库，记忆里这也是百度为中国互联网技术做的唯一贡献，大公司的责任呐，好了，再说就偏离主题了。

Protocal Buffer官方站点：<http://code.google.com/p/protobuf/>，遗憾的是不支持.Net，但社区的力量不容忽视，MySQL最近还推出社区版呢，从这个链接可以看到Protobuf的社区阵营：<http://code.google.com/p/protobuf/wiki/ThirdPartyAddOns>

OK，本文主要描述自己在.Net中基于应用层面使用Protobuf的一些体会，作为学习笔记与大家分享，个人能力有限，不足之处还请及时指正。

需求

Java为服务端，.Net为客户端，Socket通信，使用Protobuf进行数据封装和传输，如下图：



DEMO中构造了3个简单的.proto文件供各客户端使用：

message MyRequest {  
//版本号  
required int32 version =1;  
//姓名  
required string name =2;  
//个人网站  
optional string website =3[default="http://www.paotiao.com/"];  
//附加数据  
optional bytes data =4;  
}

message MyResponse {  
//版本号  
required int32 version =1;  
//响应结果  
required int32 result =2;  
}

message MyData {  
//个人简介  
optional string resume =1[default="I'm goodman"];  
}

其中MyRequest为客户端的请求，MyResponse为服务端的响应，MyData作为一个属性附加在MyRequest的data字段中，提醒注意这个byte类型的data字段，为此花费了最多时间并最终导至放弃Protobuf-net来做跨平台的应用。

Protobuf-net

官方站点：<http://code.google.com/p/protobuf-net/>

Protobuf-net是第三方中最强大应用最广泛的一个，支持.Net、C#、WCF、VB，并且DEMO丰富，网上可查到的资料也最多。

**生成.CS类文件**

安装后通过 protogen.exe 就可将.proto文件生成.cs文件(Demo中我将命令封装在/tools/getCS.bat中)：

 echo on  
protogen -i:ProtoMyRequest.proto -o:ProtoMyRequest.cs  
protogen -i:ProtoMyResponse.proto -o:ProtoMyResponse.cs  
protogen -i:ProtoMyData.proto -o:ProtoMyData.cs

接着将生成的3个.cs文件包含在项目中，同时在项目中引用protobuf-net.dll

**代码示例(服务端与客户端)**

using System;

using System.IO;

using System.Text;

using System.Threading;

using System.Net;

using System.Net.Sockets;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using ProtoBuf;

//自定义

using ProtoMyData;

using ProtoMyRequest;

using ProtoMyResponse;

namespace protobuf\_net

{

class Program

{

private static ManualResetEvent allDone = new ManualResetEvent(false);

static void Main(string[] args)

{

beginDemo();

}

private static void beginDemo()

{

//启动服务端

TcpListener server = new TcpListener(IPAddress.Parse("127.0.0.1"), 9527);

server.Start();

server.BeginAcceptTcpClient(clientConnected, server);

Console.WriteLine("SERVER : 等待数据 ---");

//启动客户端

ThreadPool.QueueUserWorkItem(runClient);

allDone.WaitOne();

Console.WriteLine("SERVER : 退出 ---");

server.Stop();

}

//服务端处理

private static void clientConnected(IAsyncResult result)

{

try

{

TcpListener server = (TcpListener)result.AsyncState;

using (TcpClient client = server.EndAcceptTcpClient(result))

using (NetworkStream stream = client.GetStream())

{

//获取

Console.WriteLine("SERVER : 客户端已连接，读取数据 ---");

//proto-buf 使用 Base128 Varints 编码

MyRequest myRequest = Serializer.DeserializeWithLengthPrefix<MyRequest>(stream, PrefixStyle.Base128);

//使用C# BinaryFormatter

IFormatter formatter = new BinaryFormatter();

MyData myData = (MyData)formatter.Deserialize(new MemoryStream(myRequest.data));

//MyData.MyData mydata = Serializer.DeserializeWithLengthPrefix<MyData.MyData>(new MemoryStream(request.data), PrefixStyle.Base128);

Console.WriteLine("SERVER : 获取成功, myRequest.version={0}, myRequest.name={1}, myRequest.website={2}, myData.resume={3}", myRequest.version, myRequest.name, myRequest.website, myData.resume);

//响应(MyResponse)

MyResponse myResponse = new MyResponse();

myResponse.version = myRequest.version;

myResponse.result = 99;

Serializer.SerializeWithLengthPrefix(stream, myResponse, PrefixStyle.Base128);

Console.WriteLine("SERVER : 响应成功 ---");

//DEBUG

//int final = stream.ReadByte();

//if (final == 123)

//{

// Console.WriteLine("SERVER: Got client-happy marker");

//}

//else

//{

// Console.WriteLine("SERVER: OOPS! Something went wrong");

//}

Console.WriteLine("SERVER: 关闭连接 ---");

stream.Close();

client.Close();

}

}

finally

{

allDone.Set();

}

}

//客户端请求

private static void runClient(object state)

{

try

{

//构造MyData

MyData myData = new MyData();

myData.resume = "我的个人简介";

//构造MyRequest

MyRequest myRequest = new MyRequest();

myRequest.version = 1;

myRequest.name = "吴剑";

myRequest.website = "www.paotiao.com";

//使用C# BinaryFormatter

using (MemoryStream ms = new MemoryStream())

{

IFormatter formatter = new BinaryFormatter();

formatter.Serialize(ms, myData);

//Serializer.Serialize(ms, mydata);

myRequest.data = ms.GetBuffer();

ms.Close();

}

Console.WriteLine("CLIENT : 对象构造完毕 ...");

using (TcpClient client = new TcpClient())

{

client.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9527));

Console.WriteLine("CLIENT : socket 连接成功 ...");

using (NetworkStream stream = client.GetStream())

{

//发送

Console.WriteLine("CLIENT : 发送数据 ...");

ProtoBuf.Serializer.SerializeWithLengthPrefix(stream, myRequest, PrefixStyle.Base128);

//接收

Console.WriteLine("CLIENT : 等待响应 ...");

MyResponse myResponse = ProtoBuf.Serializer.DeserializeWithLengthPrefix<MyResponse>(stream, PrefixStyle.Base128);

Console.WriteLine("CLIENT : 成功获取结果, version={0}, result={1}", myResponse.version, myResponse.result);

//DEBUG client-happy marker

//stream.WriteByte(123);

//关闭

stream.Close();

}

client.Close();

Console.WriteLine("CLIENT : 关闭 ...");

}

}

catch (Exception error)

{

Console.WriteLine("CLIENT ERROR : {0}", error.ToString());

}

}

}//end class

}

从代码中可以发现protobuf-net已考虑的非常周到，不论是客户端发送对象还是服务端接收对象，均只需一行代码就可实现：

//客户端发送对象  
ProtoBuf.Serializer.SerializeWithLengthPrefix(stream, myRequest, PrefixStyle.Base128);  
//服务端接收对象  
MyRequest myRequest = Serializer.DeserializeWithLengthPrefix<MyRequest>(stream, PrefixStyle.Base128);

所以如果Server与Client均使用.Net，Protobuf-net会是理想选择。

但我的项目需要跨平台，同时项目中又恰恰使用了byte类型字段，经过反复调试比较，发现一个关键问题：假使proto脚本和对象属性值完全一样， 但只要包含byte类型的字段，那么通过Java序列化的二进制与C#序列化的二进制结果一定不同。而Protobuf中Google原生支持Java， 那么几乎可以确定Protobuf-net对Protobuf的支持存在瑕疵。

后来在使用Protobuf-csharp-sport后验证了这一点，Protobuf-net使用C#的byte[]来实现bytes，而 Java以及Protobuf-csharp-port均使用ByteString，前者是无符号的，后者是有符号的，语言的基本差异导致两者无法兼容， 所以最终我只能放弃Protobuf-net。

Protobuf-csharp-port

官方站点：<http://code.google.com/p/protobuf-csharp-port/>

Protobuf-csharp-port的文档资料、DEMO、应用范围都不如Protobuf-net，但Protobuf-csharp- port更遵循Google的Protobuf，甚至应用和代码都几乎一样，所以跨平台，Protobuf-csharp-port是不二之选。

**生成.CS类文件**

先直接使用Google的 protoc.exe 生成二进制文件。

然后通过 protogen.exe 将二进制文件生成C#类文件(Demo中我将命令封装在/tools/getCS.bat中)：

echo on  
protoc --descriptor\_set\_out=ProtoMyRequest.protobin --include\_imports ProtoMyRequest.proto  
protoc --descriptor\_set\_out=ProtoMyResponse.protobin --include\_imports ProtoMyResponse.proto  
protoc --descriptor\_set\_out=ProtoMyData.protobin --include\_imports ProtoMyData.proto  
  
protogen ProtoMyRequest.protobin  
protogen ProtoMyResponse.protobin  
protogen ProtoMyData.protobin

接着将生成的3个.cs文件包含在项目中，同时在项目中引用Google.ProtocolBuffers.dll

**代码示例（服务端与客户端）**

using System;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using Google.ProtocolBuffers;

namespace protobuf\_csharp\_sport

{

class Program

{

private static ManualResetEvent allDone = new ManualResetEvent(false);

static void Main(string[] args)

{

beginDemo();

}

private static void beginDemo()

{

//启动服务端

TcpListener server = new TcpListener(IPAddress.Parse("127.0.0.1"), 9528);

server.Start();

server.BeginAcceptTcpClient(clientConnected, server);

Console.WriteLine("SERVER : 等待数据 ---");

//启动客户端

ThreadPool.QueueUserWorkItem(runClient);

allDone.WaitOne();

Console.WriteLine("SERVER : 退出 ---");

server.Stop();

}

//服务端处理

private static void clientConnected(IAsyncResult result)

{

try

{

TcpListener server = (TcpListener)result.AsyncState;

using (TcpClient client = server.EndAcceptTcpClient(result))

{

using (NetworkStream stream = client.GetStream())

{

//获取

Console.WriteLine("SERVER : 客户端已连接，数据读取中 --- ");

byte[] myRequestBuffer = new byte[49];

int myRequestLength = 0;

do

{

myRequestLength = stream.Read(myRequestBuffer, 0, myRequestBuffer.Length);

}

while (stream.DataAvailable);

MyRequest myRequest = MyRequest.ParseFrom(myRequestBuffer);

MyData myData = MyData.ParseFrom(myRequest.Data);

Console.WriteLine("SERVER : 获取成功, myRequest.Version={0}, myRequest.Name={1}, myRequest.Website={2}, myData.Resume={3}", myRequest.Version, myRequest.Name, myRequest.Website, myData.Resume);

//响应(MyResponse)

MyResponse.Builder myResponseBuilder = MyResponse.CreateBuilder();

myResponseBuilder.Version = myRequest.Version;

myResponseBuilder.Result = 99;

MyResponse myResponse = myResponseBuilder.Build();

myResponse.WriteTo(stream);

Console.WriteLine("SERVER : 响应成功 ---");

Console.WriteLine("SERVER: 关闭连接 ---");

stream.Close();

}

client.Close();

}

}

finally

{

allDone.Set();

}

}

//客户端请求

private static void runClient(object state)

{

try

{

//构造MyData

MyData.Builder myDataBuilder = MyData.CreateBuilder();

myDataBuilder.Resume = "我的个人简介";

MyData myData = myDataBuilder.Build();

//构造MyRequest

MyRequest.Builder myRequestBuilder = MyRequest.CreateBuilder();

myRequestBuilder.Version = 1;

myRequestBuilder.Name = "吴剑";

myRequestBuilder.Website = "www.paotiao.com";

//注：直接支持ByteString类型

myRequestBuilder.Data = myData.ToByteString();

MyRequest myRequest = myRequestBuilder.Build();

Console.WriteLine("CLIENT : 对象构造完毕 ...");

using (TcpClient client = new TcpClient())

{

client.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9528));

Console.WriteLine("CLIENT : socket 连接成功 ...");

using (NetworkStream stream = client.GetStream())

{

//发送

Console.WriteLine("CLIENT : 发送数据 ...");

myRequest.WriteTo(stream);

//接收

Console.WriteLine("CLIENT : 等待响应 ...");

byte[] myResponseBuffer = new byte[4];

int myResponseLength = 0;

do

{

myResponseLength = stream.Read(myResponseBuffer, 0, myResponseBuffer.Length);

}

while (stream.DataAvailable);

MyResponse myResponse = MyResponse.ParseFrom(myResponseBuffer);

Console.WriteLine("CLIENT : 成功获取结果, myResponse.Version={0}, myResponse.Result={1}", myResponse.Version, myResponse.Result);

//关闭

stream.Close();

}

client.Close();

Console.WriteLine("CLIENT : 关闭 ...");

}

}

catch (Exception error)

{

Console.WriteLine("CLIENT ERROR : {0}", error.ToString());

}

}

}//end class

}

Protobuf#

官方站点：<http://code.google.com/p/protosharp/>

暂未测试Protobuf#

结束语

基本花了一周时间了解和学习了Google Protobuf在.NET下的应用，也找到了Protobuf跨平台的方案，但好事多魔，C# Socket发送的protobuf数据包在**Java Netty** 中怎么也获取不到，我想也许是平台差异，但对Java知之甚少，如有知情人士还请指点迷津。

DEMO

DEMO下载：<http://files.cnblogs.com/wu-jian/ProtobufDemo.rar>

DEMO运行环境：.Net Framework 4.0， VS2010

来源： <<http://www.cnblogs.com/wu-jian/archive/2011/02/22/1961104.html>>