[**Socket开发框架之数据加密及完整性检查**](http://www.cnblogs.com/wuhuacong/p/5333846.html)

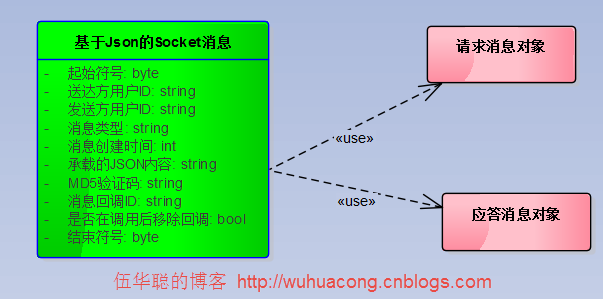
 在前面两篇介绍了Socket框架的设计思路以及数据传输方面的内容，整个框架的设计指导原则就是易于使用及安全性较好，可以用来从客户端到服务端的数据安全传输，那么实现这个目标就需要设计好消息的传输和数据加密的处理。本篇主要介绍如何利用Socket传输协议来实现数据加密和数据完整性校验的处理，数据加密我们可以采用基于RSA非对称加密的方式来实现，数据的完整性，我们可以对传输的内容进行MD5数据的校验对比。

**1、Socket框架传输内容分析**

前面介绍过Socket的协议，除了起止标识符外，整个内容是一个JSON的字符串内容，这种格式如下所示。

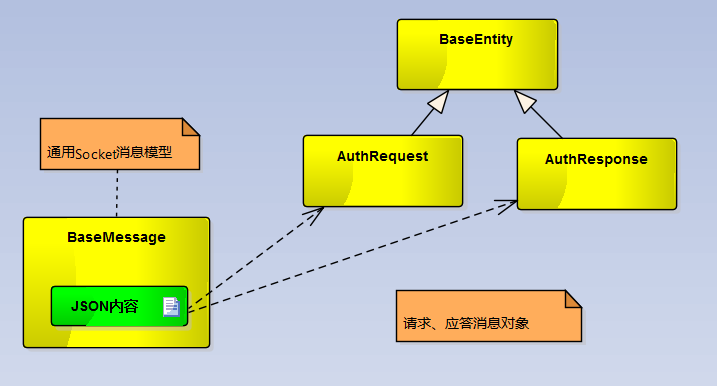
http://images2015.cnblogs.com/blog/8867/201603/8867-20160325225900026-1968480469.png

上述消息内容，我们可以通过开始标识位和结束标识位，抽取出一个完整的Socket消息，这样我们对其中的JSON内容进行序列号就可以得到对应的实体类，我们定义实体类的内容如下所示。

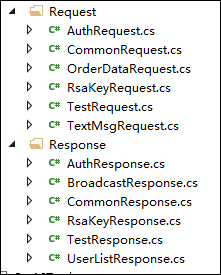


我们把消息对象分为请求消息对象和应答消息对象，他们对应的是Request和Response的消息，也就是一个是发起的消息，一个是应答的消息。其中上图的“承载的JSON内容就是我们另一个传输对象的JSON字符串，这样我们通过这种字符串来传输不同对象的信息，就构造出了一个通用的消息实体对象。

另外这些传输的消息对象，它本身可以继承于一个实体类的基类，这样方便我们对它们的统一处理，如下图所示，就是一个通用的消息对象BaseMessage和其中JSON内容的对象关系图，如AuthRequest是登陆验证请求，AuthorRepsonse是登陆验证的应答。



当然，我们整个Socket应用，可以派生出很多类似的Request和Response的消息对象，如下所示是部分消息的定义。



对于非对称加密的处理，一般来说会有一些性能上的损失，不过我们考虑到如果是安全环境的数据传输处理的话，我们使用非对称加密还是比较好的。

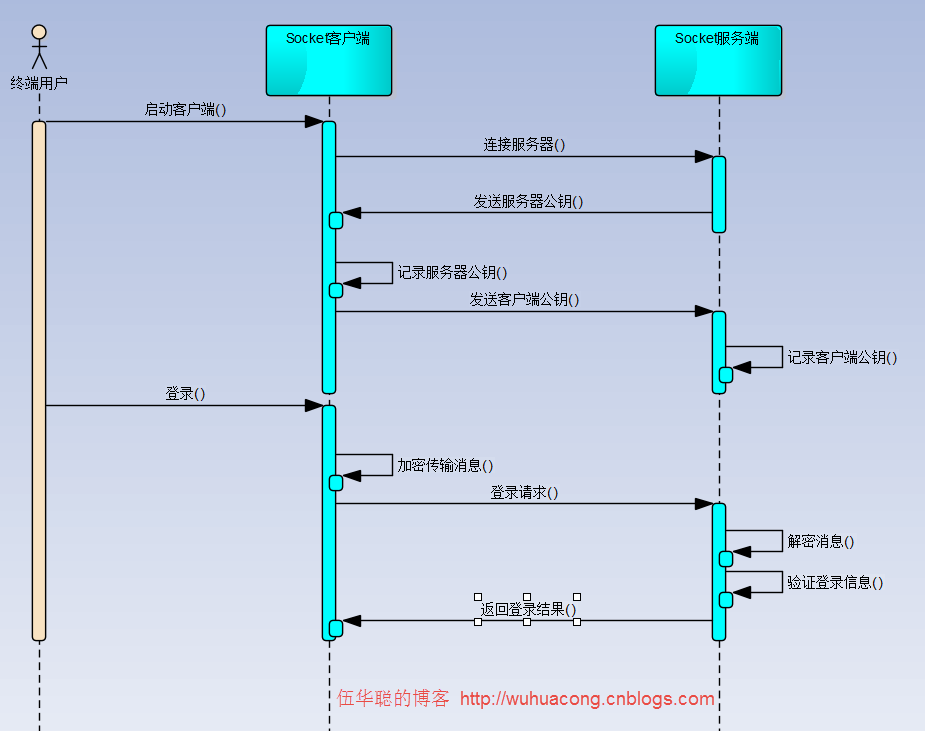
当然也有人建议采用非对称加密部分内容，如双方采用约定的对称加密键，通过非对称加密的方式来传输这个加密键，然后两边采用对称加密算法来处理也是可以的。不过本框架主要介绍采用非对称加密的方式来加密其中的JSON内容，其他部分常规的信息不进行加密。

**2、非对称加密的公钥传递**

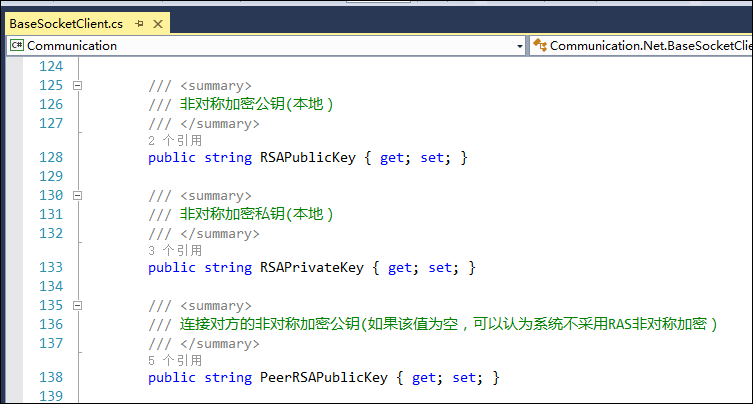
消息加密数据的传输前，我们需要交换算法的公钥，也就是服务器把自己公钥给客户端，客户端收到服务器的公钥请求后，返回客户端的公钥给服务器，实现两者的交换，以后双方的消息都通过对方公钥加密，把加密内容通过标准的Socket消息对象传递，这样对方收到的加密内容，就可以通过自身的私钥进行解密了。

那么要在传递消息前处理这个公钥交换的话，我们可以设计在服务器接入一个新的客户端连接后（在登录处理前），向客户端发送服务器的公钥，客户端受到服务器的公钥后，回应自己的公钥信息，并存储服务器的公钥。这样我们就可以在登陆的时候以及后面的消息传递过程中，使用对方公钥进行加密数据，实现较好的安全性。

公钥传递的过程如下图所示，也就是客户端发起连接服务器请求后，由服务器主动发送一个公钥请求命令，客户端收到后进行响应，发送自身的公钥给服务器，服务器把客户端的公钥信息存储在对应的Socket对象上，以后所有消息都通过客户端公钥加密，然后发送给客户端。



前面我们介绍过，我们所有的自定义Socket对象，都是继承于一个BaseSocketClient这样的基类对象，那么我们只需要在它的对象里面增加几个属性几个，一个是自己的公钥、私钥，一个是对方的公钥信息，如下所示。



在程序的启动后，包括客户端启动，服务器启动，我们都需要构建好自己的公钥私钥信息，如下代码是产生对应的公钥私钥信息，并存储在属性里面。

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

this.RSAPublicKey = rsa.ToXmlString(false);// 公钥

this.RSAPrivateKey = rsa.ToXmlString(true);// 私钥

}

例如在服务器端，在客户端Socket成功接入后，我们就给对应的客户端发送公钥请求消息，如下代码所示。

[复制代码](javascript:void(0);)

/// <summary>

/// 客户端连接后的处理（如发送公钥秘钥）

/// </summary>

/// <param name="client">连接客户端</param>

protected override void OnAfterClientConnected(ClientOfShop client)

{

//先记录服务端的公钥，私钥

client.RSAPrivateKey = Portal.gc.RSAPrivateKey;

client.RSAPublicKey = Portal.gc.RSAPublicKey;

//发送一个公钥交换命令

var request = new RsaKeyRequest(Portal.gc.RSAPublicKey);

var data = request.PackData();

client.SendData(data);

Thread.Sleep(100);

}

[复制代码](javascript:void(0);)

那么在客户端，接收到服务端的消息后，对消息类型判断，如果是公钥请求，那么我们需要进行回应，把自己的公钥发给服务器，否则就进行其他的业务处理了。

[复制代码](javascript:void(0);)

/// <summary>

/// 重写读取消息的处理

/// </summary>

/// <param name="message">获取到的完整Socket消息对象</param>

protected override void OnMessageReceived(BaseMessage message)

{

if (message.MsgType == DataTypeKey.RSARequest)

{

var info = JsonTools.DeserializeObject<RsaKeyRequest>(message.Content);

if (info != null)

{

//记录对方的公钥到Socket对象里面

this.PeerRSAPublicKey = Portal.gc.UseRSAEncrypt ? info.RSAPublicKey : "";

Console.WriteLine("使用RAS加密：{0}，获取到加密公钥为：{1}", Portal.gc.UseRSAEncrypt, info.RSAPublicKey);

//公钥请求应答

var publicKey = Portal.gc.UseRSAEncrypt ? Portal.gc.RSAPublicKey : "";

var data = new RsaKeyResponse(publicKey);//返回客户端的公钥

var msg = data.PackData(message);

SendData(msg);

Thread.Sleep(100);//暂停下

}

}

else

{

//交给业务消息处理过程

this.MessageReceiver.AppendMessage(message);

this.MessageReceiver.Check();

}

}

[复制代码](javascript:void(0);)

如果我们交换成功后，我们后续的消息，就可以通过RSA非对称加密进行处理了，如下代码所示。

data.Content = RSASecurityHelper.RSAEncrypt(this.PeerRSAPublicKey, data.Content);

而解密消息，则是上面代码的逆过程，如下所示。

message.Content = RSASecurityHelper.RSADecrypt(this.RSAPrivateKey, message.Content);

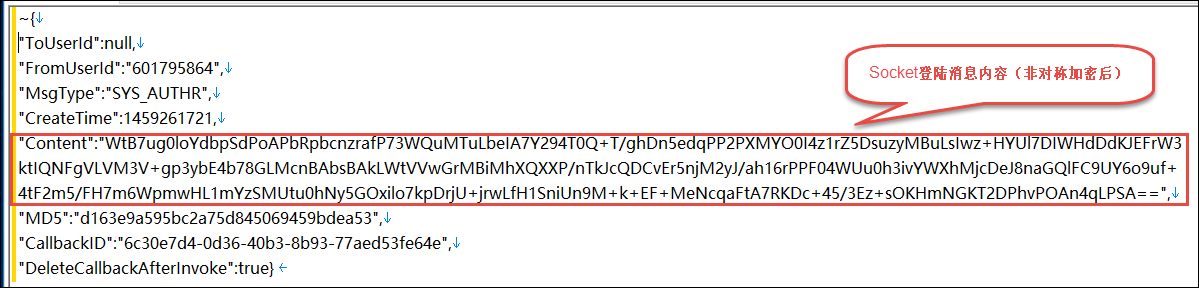
最后我们把加密后的内容组成一个待发送的Socket消息，包含起止标识符，如下所示。

//转为JSON，并组装为发送协议格式

var json = JsonTools.ObjectToJson(data);

toSendData = string.Format("{0}{1}{2}", (char)this.StartByte, json, (char)this.EndByte);

这样就是我们需要发送的消息内容了，我们拦截内容，可以看到大概的内容如下所示。



上面红色框的内容，必须使用原有的私钥才能进行解密，也就是在网络上，被谁拦截了，也无法进行解开，保证了数据的安全性。

**3、数据完整性检查**

 数据的完整性，我们可以通过消息内容的MD5值进行比对，实现检查是否内容被篡改过，不过如果是采用了非对称加密，这种 完整性检查也可以忽略，不过我们可以保留它作为一个检查处理。

因此在封装数据的时候，就把内容部分MD5值计算出来，如下所示。

data.MD5 = MD5Util.GetMD5\_32(data.Content);//获取内容的MD5值

然后在获得消息，并进行解密后（如果有），那么在服务器端计算一下MD5值，并和传递过来的MD5值进行比对，如果一致则说明没有被篡改过，如下代码所示。

[复制代码](javascript:void(0);)

var md5 = MD5Util.GetMD5\_32(message.Content);

if (md5 == message.MD5)

{

OnMessageReceived(message);//给子类重载

}

else

{

Log.WriteInfo(string.Format("收到一个被修改过的消息：\r\n{0}", message.Content));

}

[复制代码](javascript:void(0);)

以上就是我在Socket开发框架里面，实现传输数据的非对称加密，以及数据完整性校验的处理过程。

http://www.cnblogs.com/Images/OutliningIndicators/None.gif主要研究技术：代码生成工具、会员管理系统、客户关系管理软件、病人资料管理软件、Visio二次开发、酒店管理系统、仓库管理系统等共享软件开发  
 专注于[Winform开发框架/混合式开发框架](http://www.iqidi.com/Framework/index.html)、[Web开发框架](http://www.iqidi.com/Framework/WebIntroduce.htm)、[Bootstrap开发框架](http://www.iqidi.com/Framework/BootstrapIndex.html)、[微信门户开发框架的研究及应用](http://www.iqidi.com/Framework/WeixinIndex.htm)。  
  转载请注明出处：  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif撰写人：伍华聪  [http://www.iqidi.com](http://www.iqidi.com/) 