RTMP 协议研究

1协议研究概述

协议设计和分析一直都是在工作遇到，正好在这里总结一下，说到协议，在这个网络的时代，没有人可以离开它了。他存在我们生活中的任何角落，只不过我们平时，并没有注意到它的存在，可以这么说如果没有协议，我们生活和日常的工作生产都不能进行。如果仔细想想你生活中用到的所有东西，协议已经包含其中。那到底什么是协议呢？说的简单一点就是双方达成的共识，以便更好的交流，理论上协议是什么呢？如果学过《信号与系统》的人都知道有个简单的道理，就是信息在经过一个管道的符号集，到另一个符号集时信息不会丢失。

任何复杂的事物都有个最简单的本质，网络上的协议也是这样，有个最基本的本质。除去上下层的概念，协议就只剩下通信双方实体的规则。

一般的协议都包含最基本的协议头，不管是物理层、链路层、还是网络层，这个头就构成了协议的本质东西。通常协议头要包含以下最基本的三项信息：

1 双方实体的唯一标示，用来标示通信双方的实体。

2 类型描述或者是净核描述，标志净核的内容。

3 协议净核的长度，用来在萃取净核的内容应用。

其中，前两项是必须要有的，没有他们，通信双方的交互根本得不到保证，第三项在不太灵活的通信中可以去掉，而有第二项的类型推出。

协议的丰富性，有净核的多样性体现。

协议头除了以上的三项，还可以增加更多的信息（比如控制信息、时间信息等），取决于具体的应用。找到这些基本的东西，再去看协议的时候，能够更好的抓住协议的主体进行分析和设计了。

如图 协议物理结构

2 RTMP 协议概述

RTMP 协议是被 Flash 用于对象、视频、音频的传输。该协议建立在 TCP 协议或者轮询 HTTP 协议之上，RTMP 协议就像一个用来装数据包的容器，这些数据可以是 AMF 格式的数据，也可以是 FLV 中的视/音频数据。一个单一的连接可以通过不同的通道传输多路网络流，这些通道中的包都是按照固定大小的包传输的.

3 RTMP 协议部分

3.1协议头

```
struct RTMP_HEAD
{
        char cChannelid：6;//第一个字节的后6位
        char cCheadsize ; //第一个字节的头两位
        char cTimer[3];   //三个字节表示的时间信息
        char cLength[3]; //三个字节表示的长度
        char cDatatype; //数据类型
        char sStreamid[4]; //流标识
};
```

这里有三个最基本的元素（**唯一标示**）、（**类型**）和（**净核的长度**）分别是：cChannelid、cDatatype 和 cLength。

3.2 数据类型

数据类型决定了协议上层可以做的具体的事情，和使用协议的人必须遵循的规则。
同时**数据类型**说明了**净核**的基本内容。

RTMP 数据类型：

0×01	Chunk Size	changes the chunk size for packets
0×02	Unknown	anyone know this one?
0×03	Bytes Read	send every x bytes read by both sides
0×04	Ping	ping is a stream control message, has subtypes
0×05	Server BW	the servers downstream bw
0×06	Client BW	the clients upstream bw
0×07	Unknown	anyone know this one?
0×08	Audio Data	packet containing audio
0×09	Video Data	packet containing video data
0x0A - 0×11	Unknown	anyone know?
0×12	Notify	an invoke which does not expect a reply
0×13	Shared Object	has subtypes
0×14	Invoke	like remoting call, used for stream actions too.

3.3协议的净核

RTMP 的协议净核是用 AMF 格式来描述，AMF 格式本身的产生就是为了 RTMP 协议服务的，最初的 RTMP 采用 XML 的形式传输数据，但 XML 只是字符形式的值对的格式传输

数据，而随着应用的普及这完全不能满足要求了，比如对象、结构、数组，甚至可以是数据集，配合 DataGrid 组件可以很方便地显示数据。
为了处理复杂数据类型，采用一种独有的方式使 Flash 与应用服务器间可以来回传送数据势在必行。于是 AMF 应运而生。
AMF 是 Adobe 独家开发出来的通信协议，它采用二进制压缩，序列化、反序列化、传输数据，从而为 Flash 播放器与 Flash Remoting 网关通信提供了一种轻量级的、高效能的通信方式。如下图所示。

AMF 最大的特色在于可直接将 Flash 内置对象，例如 Object, Array, Date, XML，传回服务器端，并且在服务器端自动进行解析成适当的对象，这就减轻了开发人员繁复工作，同时也更省了开发时间。由于 AMF 采用二进制编码，这种方式可以高度压缩数据，因此非常适合用来传递大量的资料。数据量越大，Flash Remoting 的传输效能就越高，远远超过 Web Service。至于 XML, LoadVars 和 loadVariables() ，它们使用纯文本的传输方式，效能就更不能与 Flash Remoting 相提并论了。
注意： Flash Remoting 需要浏览器支持 Binary POST，Flash 播放器在 Netscape 6.x.环境下运行 Flash Remoting 会不起作用（Flash Remoting 调用没有效果也不返回错误）， Netscape 7 已经纠正了这个 bug 。对于早期 Safari 和 Chimera 版的苹果机也有这个问题。
同样是轻量级数据交换协议，同样是通过调用远程服务，同样是基于标准的 HTTP 和 HTTPS 协议， Flash Remoting 为什么选择了使用 AMF 而放弃了 SOAP 与 Flash 播放器通信呢 有如下原因：

SOAP 将数据处理成 XML 格式，相对于二进制的 AFM 太冗长了；
AMF 能更有效序列化数据；因为 AMF 的初衷只是为了支持 Flash ActionScript 的数据类型，而 SOAP 却致力于提供更广泛的用途；
AMF 支持 Flash 播放器 6只需要浏览器增加4 KB 左右（压缩后）的大小，而 SOAP 就大多了；
SOAP 的一些头部文件请求在 Flash 播放器 6不支持。那 Flash 播放器 6为什么能访问基于 SOAP 的 Web 服务呢？原来 Flash Remoting 网关将 SOAP 请求在服务器端与转换成 AFM 格式，然后利用 AFM 与 Flash 播放器通信。另外，AMF 包中包含 onResult 事件（比如说 response 事件）和 onStatus 事件（比如说 error 事件），这些事件对象在 Flash 中可以直接使用。
AMF 从 Flash MX 时代的 AMF0发展到现在的 AMF3。AMF3用作 Flash Playe 9的 ActionScript 3.0的默认序列化格式，而 AMF0则用作旧版的 ActionScript 1.0和2.0的序列化格式。 在网络传输数据方面，AMF3比 AMF0更有效率。AMF3能将 int 和 uint 对象作为整数（integer）传输，并且能序列化 ActionScript 3.0 才支持的数据类型，比如 ByteArray， XML 和 Iexternalizable。

AMF 很好的解决了内容的丰富性。（具体 AMF 格式参考附件格式文档）

3.3.1 AMF 中的数据类型 Data Types

AMF0 supports the following data types (with their type field values):

- NUMBER = 0x00
- BOOLEAN = 0x01
- STRING = 0x02
- OBJECT = 0x03
- MOVIECLIP = 0x04

- NULL_VALUE = 0x05
- UNDEFINED = 0x06
- REFERENCE = 0x07
- ECMA_ARRAY = 0x08
- OBJECT_END = 0x09
- STRICT_ARRAY = 0x0a
- DATE = 0x0b
- LONG_STRING = 0x0c
- UNSUPPORTED = 0x0d
- RECORD_SET = 0x0e
- XML_OBJECT = 0x0f
- TYPED_OBJECT = 0x10

Binary Format

AMF format for a value/object consists of a type byte (see above) followed by zero or more bytes. This section describes the bytes following the type byte for various types.

NUMBER (type byte: 0x00)

Numbers are stored as 8 byte (big endian) float double. On x86 you can just byteswap a double to encode it correctly.

BOOLEAN (type byte: 0x01)

A boolean is encoded in one byte. FIXME: is true sent as 0xff? 0x01?

STRING (type byte: 0x02)

A string is encoded as a 2 byte (big endian) count (number of bytes) followed by that many bytes of text. Note: there is no null terminator.

I think the text is assumed to be UTF-8. Can someone double check me on this?

NULL_VALUE (type byte: 0x05)

A null has zero bytes following the type byte

UNDEFINED (type byte: 0x06)

A undefined has zero bytes following the type byte

OBJECT (type byte: 0x08)

An object is encoded as a series of key/value pairs. The key is encoded as a STRING (above) WITH NO TYPE BYTE, and the value is any AMF value.

The object encoding is terminated by 0x000009 (that is a zero length string key, followed by the OBJECT_END type byte described below.

OBJECT_END (type byte: 0x09)

This is not really a value, but a marker for the end of an OBJECT. See above.

STRICT_ARRAY (type byte: 0x0a)

This is the encoding for arrays such as ["foo", "bar", 1, 2, 3]. For a hash (a set of key/value pairs) you'll need to use OBJECT above.

An array is encoded as 4 byte (big endian) integer which is the number of elements in the array, followed by that many AMF values.

That's it. There's no terminator of any kind.

Use in shared object files

While most AMF objects are just a value, there is a special variation used by shared object

files for properties. Rather than start with the type field, followed by the length, it starts with a byte count, then the name, and then the regular AMF type field, the length, and then the data.

3.4客户端和服务器的连接过程

3.4.1客户和服务器的握手

Flash Player 以系统时间作为种子通过某种算法生成的数字签名，大小是1537字节向服务器发起第一次握手，服务器根据客户端的数字签名产生一个3073字节的验证包，给客户端，客户端在接受到服务器的回应以后会发送一个1536字节的回复。

具体的流程：

1 发送第一次握手包 handshark1
2 接收第二次握手包 handshark2
3 发送的三次握手包 handshark3

第一个握手包 handshark1和服务器的回复握手包 handshark2都是以0X03开头。这三次握手不是 RTMP 协议本身的内容，所以在这并没有包含 RTMP 的协议头。是服务器的厂家自己产品做验证用的，严格的说就是你必须用 Adobe 的客户端和服务器才能使用我的协议。

3.4.2客户和服务器通信

具体连接和请求视频的过程

4 发送 rtmp_connect 命令
5 .发送本地带宽消息.默认是125000
6 服务器返回服务器带宽信息
7 服务器返回本地带宽信息
8 服务器返回连接成功消息 "NetConnection.Connect.Success"
9 客户端发送创建流请求 encodeCreateStreamPacket
10 服务器返回创建流成功消息
11 客户端发送播放文件消息 Rtmp_Play
12 服务器返回 TYPE_CHUNK_SIZE 消息
13 服务器返回开始播放消息 "NetStream.Play.Start"
14 服务器返回视频信息(TYPE_STREAM_METADATA)，包括大小，宽高，速率等等信息－－文件长度可以在这里推算出来

RTMP 的净核决定了内容服务，adobe 的服务器采用的 AMF 格式的字串命令来控制视频的传输和播放，具体的字串命令信息如下：（注：字串的定义有厂家（adobe）自己定义，只要满足 AMF 的格式就可以）

1
1
1 NetConnection.Call.Failed
1 NetConnection.Call.BadVersion
1 NetConnection.Connect.AppShutdown
1 NetConnection.Connect.Closed
1 NetConnection.Connect.Rejected
1 NetConnection.Connect.Success

1 NetStream.Clear.Success
1 NetStream.Clear.Failed
1 NetStream.Publish.Start
1 NetStream.Publish.BadName
1 NetStream.Failed
1 NetStream.Unpublish.Success
1 NetStream.Record.Start
1 NetStream.Record.NoAccess
1 NetStream.Record.Stop
1 NetStream.Record.Failed
1 NetStream.Play.InsufficientBW
1 NetStream.Play.Start
1 NetStream.Play.StreamNotFound
1 NetStream.Play.Stop
1 NetStream.Play.Failed
1 NetStream.Play.Reset
1 NetStream.Play.PublishNotify
1 NetStream.Play.UnpublishNotify
1 NetStream.Data.Start
1 Application.Script.Error
1 Application.Script.Warning
1 Application.Resource.LowMemory
1 Application.Shutdown
1 Application.GC
1 Play
1 Pause
1 demoService.getListOfAvailableFLVs
1 getStreamLength
1 connect
1 app
1 flashVer
1 swfUrl
1 tcUrl
1 fpad
1 capabilities
1 audioCodecs
1 audioCodecs
1 videoCodecs
1 videoFunction
1 pageUrl
1 createStream
1 deleteStream
1 duration
1 framerate

1 audiocodecid
1 audiodatarate
1 videocodecid
1 videodatarate
1 height
1 width

3.4.2数据的萃取

在服务器返回开始播放消息 "NetStream.Play.Start"之后，服务器就会开始给客户端传输数据了，一般数据的萃取都是先解析协议的头，然后根据协议头中数据类型和净核长度就可以把数据部分取出，RTMP 协议也是这样。

```
struct RTMP_HEAD
{
        char cChannelid : 6;//第一个字节的后6位
        char cCheadsize ; //第一个字节的头两位
    char cTimer[3];   //三个字节表示的时间信息
    char cLength[3]; //三个字节表示的长度
    char cDatatype; //数据类型
    char sStreamid[4]; //流标识
}
```

首先判断 cDatatype 是那种类型，然后根据不同的类型进行萃取数据部分，进行不同的处理，获取视频的数据的方式先看是否是一下的类型：

0×08	Audio Data	packet containing audio
0×09	Video Data	packet containing video data

根据净核的长度读取出内存中的音视频数据，这里的音视频数据是有一定编码格式的数据，这个取决于应用的具体配置，Flash play 使用的是 FLV 的格式。要对这部分数据进行存取，还有做一部分工作，对 FLV 的视频数据进行去壳，取出数据保存文件就可以了。