The RFB Protocol

Tristan Richardson

RealVNC Ltd
(formerly of Olivetti Research Ltd / AT&T Labs Cambridge)

Version 3.8
Last updated 5 October 2006

RFB 协议

翻译:晓风

E-mail:xfsuper@gmail.com

Blog: http://cuplinux.cublog.cn

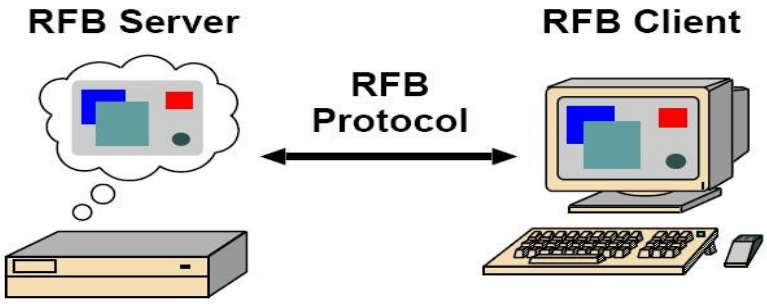
2007.4.12

目录

1 简介

RFB ("remote 帧缓存") 是一个远程图形用户的简单协议，因为它工作在帧缓存级别上，所以它可以应用于所有的窗口系统，例如：X11,Windows 和 Mac 系统。其中 VNC (Virtual Network Computing) 就采用 RFB.

远程终端用户使用机器（比如显示器、键盘/ 鼠标）的叫做 RFB 客户端，提供帧缓存变化的被称为 RFB 服务器。



RFB 是真正意义上的“瘦客机”协议。RFB 协议设计的重点在于减少对客户端的硬件需求。这样客户端就可以运行在许多不同的硬件上，客户机的任务实现上就会尽量的简单。

RFB 协议对于客户端是无状态的。也就是说：如果客户端从服务器端断开，那么如果它重新连接相同的服务器，客户端的状态会被保存。甚至，一个不同的客户端可以用来连接相同的 RFB 服务器。而在新的客户端已经能够获得与前一个客户端相同的用户状态。因此，用户的应用接口变的非常便捷。

只要合适的网络连接存在，那么用户就可以使用自己的应用程序，并且这些应用会一直保存，即使在不同的接入点也不会变化。这样无论在哪，系统都会给用户提供一个熟悉、独特的计算环境。

2 显示协议

显示协议是建立在“把像素数据放在一个由 x,y 定位的方框内”这单一图形基础之上的。乍一看上去，把这么多的用户接口组件绘制出来是非常低效的方法。但是，允许不同的像素数据编码方式，使得我们在处理不同的参数（如：网络带宽，客户端的绘制速度，服务器处理速度）有了很大程度的灵活性。

通过矩形的序列来完成帧缓存的更新。一次更新代表着从一个可用帧缓存状态转换到另一个可用，因此有点和视频的桢类似。尽管矩形的更新一般是分开的，但是并不是必须的。

显示协议的更新部分是由客户端通过命令驱动的。也就是说，更新只是在服务器端响应客户端的请求时发生的。这样就让协议更新质量是可变的。客户端/ 网络越慢，更新速度也就越慢。对于一些应用来说，相同区域的更新是连续不断的。如果用一个慢的客户端，那么帧缓存的缓存状态是可以被忽略的。这样也可以减少对客户端网络速度和绘制速度的要求。

3 输入协议

输入协议是基于标准工作站的键盘和鼠标等设备的连接协议。输入事件就是通过把客户端的输入发送到服务器端。这些输入事件也可以通过非标准的 I/O 设备来综合。例如，手写笔引擎可能产生一个键盘事件。

4 像素数据的重现

初始的交互涉及到RFB客户端和服务器之间传输像素数据格式和编码方式的协调。这种协调被设计的让客户端的工作尽量简单。而设计的底线是：服务器必须按照客户端的要求格式来提供像素数据。如果客户端可以同样的处理多种数据格式或编码格式，那么一般会选择服务器端易于生成的格式。

像素格式涉及如何通过像素值来实现不同颜色的重现。最常用的一般像素格式是24位或16位的“真彩色”，它通过位来直接实现像素值到红、绿、蓝亮度的转换。8位“颜色映射”可以任意映射像素值到RGB亮度的转换。

编码解决像素数据如何通过网络传输的问题。每一个矩形像素数据都带有数据的X,Y参数，宽和高是举行，编码类型确定像素数据的编码方式。数据本身遵循特定的编码。

目前的编码方式主要有Raw、CopyRect、RRE、Hextile和ZRLE.在实际应用中我们一般使用ZRLE、Hextile和CopyRect，因为它们提供了典型桌面的最好压缩。(参照6.6 关于每种编码方式的描述)

5 协议扩展

协议可以通过以下方式进行扩展：

新的编码方式

一种新的协议可以通过与现存的客户端和服务端进行相关兼容的添加。因为现存的服务器将会忽略它们所不支持的新编码方式。所以客户端通过新的编码方式进行请求也就不会有结果返回。

伪编码方式

除了真正的编码方式，客户端也可以请求“伪编码”通告服务器，它支持某一协议的扩展。服务器如果不支持这种扩展，那么它将忽略。值得注意的是：客户端必须先假设服务器端不支持这种扩展，直到它获得服务器端支持的确认。(参照6.7 伪编码的描述)

新的安全方式

添加一个新型的安全方式会带来无限的灵活性，它通过修改协议的一些行为，但是并没有牺牲现存客户端和服务器端的兼容性。客户端和服务器端可以通过协议好的安全方式进行交流，当然并不一定与RFB协议类似。

无论如何你都不应使用不同的版本号

RFB协议的版本是由RealVNC公司来制定的。如果你使用一个不同的协议版本可能与RFB/VNC兼容，要保证协议的兼容性，请联系RealVNC公司。这样会减少在编码方式和安全类型上的冲突。请登陆http://www.realvnc.com查看我们的联系方式，加入VNC邮件列表也是一个很好的选择。

6 协议消息

RFB协议可以进行可靠的传输，如字节流或基于消息的。和大多数协议一样，它也是通过TCP/IP协议簇连接。协议由三步完成连接。首先是握手报文，目的是对协议版本和加密方式进行协商。第二步是初始化报文，主要用于客户和服务器的初始化消息。最后就是正常协议的交互，客户端可以按需发送消息，然后可以获得服务器的回复。所有的消息以消息类型开始，接下来是特定的消息数据。

协议消息描述的基本类型有：U8、U16、U32、S8、S16、S32。U 表示无符号整数，S 表示有符号整数。所有字节整数（除了像素值本身）遵从 Endian 顺序。

PIXEL 代表一个像素值 bytesPerPixel 字节，8XbytesPerPixel=bits-per-pixel，这个等式在客户端/ 服务器、ServerInit 消息(参照 6.3.2 节)、SetPixelFormat 消息（参照 6.4.1 节）中是被承认的。

6.1 握手消息

6.1.1 协议版本

握手始于服务器向客户发送协议版本的消息，以告知客户服务器所能支持 RFB 协议的最高版本号。此时客户端会发送相似的消息告诉服务器将要使用的协议版本。客户端不应该请求高于服务器的协议版本。如此一来就给客户和服务器端提供了一种向后兼容机制。

目前发布的协议版本主要有 3.3、3.7、3.8（3.5 版本被报告存在问题），对于新的编码和伪码方式版本号不需要进行修改，因为服务器端可能忽略它不能识别的版本。

协议版本消息由 12 字节的 ASCII 码串组成，它的格式"RFB xxx.yyy\n",其中 xxx 和 yyy 分别是主要和次的版本号，并用 0 进行补充。

6.1.2 安全

一旦协议版本被确定，服务器和客户端必须一致同意连接的安全类型。

V3.7 向上 服务器支持的安全类型：

字节数	类型[值]	描述
1	U8	安全类型号
安全类型号	U8 数组	安全类型

如果客户端能支持服务器的某一安全类型，那么客户端就会发送一个字节来确认连接的安全类型:

字节数	类型[值]	描述
1	U8	安全类型

如果安全类型号是 0，那么连接失败（例如服务器不支持客户请求版本号），这样就会有字符串来描述失败原因：

字节数	类型 [值]	描述
4	U32	原因长度
原因长度	U8 数组	原因字串

服务器在发送原因字串后就会关闭连接。

V3.3 服务器决定安全类型并发送一个字：

字节数	类型 [值]	描述
4	U32	安全类型

安全类型的值一般有 0、1、2。0 表示连接失败，并伴随原因字串。
本文中定义的安全类型有：

号码	名称
0	不可用
1	NONE
2	VNC 认证

其他已注册的安全类型主要包括：

号码	名称
5	RA2
6	RA2ne
16	Tight
17	Ultra
18	TLS
19	VeNCrypt

安全类型确定，数据遵循安全类型的定义（详情参见 6.2）。安全握手报文的末端，一般伴随着安全结果消息。

注意：在安全握手报文之后，很有可能是其他协议数据经过加密或者被修改的通道。

6.1.3 安全结果

服务器发送一个字告诉客户端安全握手成功。

字节号	类型 [值]	描述
4	U32	状态：
	0	成功
	1	失败

如果成功，协议进入初始报文 （第6.3节）

V3.8以上版本 如果不成功，就会有字符串来描述失败原因，并关闭连接：

字节数	类型 [值]	描述
4	U32	原因长度
原因长度	U8数组	原因字串

对于V3.3和 3.7 如果不成功，服务器直接关闭连接。

6.2 安全类型

6.2.1 NONE

不需要认证，协议数据将被使用明文发送。

V3.8以上版本， 还会带有安全结果的消息。

V3.3和 3.7 协议直接进入初始报文(参照6.3).

6.2.2 VNC认证

采用VNC认证，协议数据将采用明文发送，服务器发送一个16字节的随机数验证：

字节数	类型 [值]	描述
16	U8	验证

客户端使用DES对验证进行加密，使用用户密码作为密钥，把16字节的回复返回到服务器：

字节数	类型 [值]	描述
16	U8	验证

随之而来的就是安全结果消息。

6.3 初始化消息

一旦客户和服务器都同意使用同一安全类型进行交流，那么协议进入初始化消息。客户端发送一个客户初始化消息，紧接着就是服务器初始化消息。

6.3.1 客户端初始化

字节数	类型 [值]	描述
1	U8	共享标志

如果服务器同意其他客户继续连接，那么共享标志应该是非零（真）。否则，服务器将断开其他客户的连接。

6.3.2 服务器初始化

服务器收到客户端初始化的消息后，会发送一个服务器初始化消息。主要是告知客户端服务器上帧缓存的高宽，像素格式还有与桌面相关的名称。

字节数	类型 [值]	描述
2	U16	帧缓存宽度
2	U16	帧缓存高度
16	像素格式	服务器像素格式
4	U32	名字长度
名字长度	U8 数组	名字字串

像素格式主要包括以下段：

字节数	类型 [值]	描述
1	U8	位/ 像素
1	U8	深度
1	U8	big-endian 标志
1	U8	真彩标志
2	U16	红色最大值
2	U16	绿色最大值
2	U16	蓝色最大值
1	U8	红色- 替换
1	U8	绿色- 替换
1	U8	蓝色- 替换
3		补充

服务器象素定义服务器本来的象素格式，这种象素格式会被一直使用，除非客户端使

用设置象素格式消息来请求另一种象素格式。（参照 6.4.1）

位每象素表示每一个象素值对应的位数，它必须大于等于每个象素值。目前位每象素必须是8,16 或32——小于8 位象素不被支持。如果多字节象素被看做big endian,那么Big-endian 标志非零。当然了，这对8 位每象素没有任何意义。

如果真彩标志非零，那么最后6 项规定如何按照象素值来确定红、绿、蓝的亮度。红的最大值是红色的最大值（=2^n-1, n 表示用在红色上的位数)。注意这个值一般在big endian 的顺序中。红色- 替换表示要得到最低明显bit 所需要的替换个数。绿色最大值、绿色- 替换和蓝色最大值、蓝色- 替换和红色类似。要在0—红色最大值之间找一个红色值，按照以下步骤进行：

- 遵循big-endian 标志进行象素值。(例如：如果big-endian 标志为0，主机的字节顺序是big endian，然后交换)。
- 使用红色—替换将右边替换。
- 和红色最大值进行逻辑与(按照主机字节顺序)。

如果真彩标志是零，那么服务器使用的象素值不是直接由红、绿、蓝的亮度组成，但是服务为索引到颜色图中去。颜色图中的项目是由服务器使用“设置颜色面板条目”消息进行设置的。

6.4 客户到服务器消息

客户到服务器的消息在本文中有如下定义：

号码	名称
0	设置象素格式
2	设置编码
3	帧缓存更新请求
4	按键事件
5	鼠标事件
6	客户剪切文本

其余的注册消息类型有：

号码	名称
255	Anthony Liguori

值得注意的是：如果要发送未在本文中定义的消息，那么必须得到服务器端的消息确认。

6.4.1 设置象素格式

“帧缓存更新”消息中设置什么格式的象素值如何设置。
如果客户端没有发送“设置象素格式”消息，那么服务器发送的象素值将遵循在服务器初始化消息中所包括的象素格式（参照6.3.2)。

如果真彩标志是零，那么意味着使用“颜色面板”，只要客户端发送颜色面板空的消息，或者是面板项被服务器端重设，服务器可以使用设置颜色面板项目进行颜色面板的设置(参照6.5.2)。

字节数	类型 [值]	描述
1	U8 0	消息类型
3		填充
16	象素格式	象素格式

注：其中的象素格式如在6.3.2 中的描述

6.4.2设置编码方式

设置编码方式可以来确定服务器发送象素数据的类型。消息中编码方式的顺序是客户端按照优先级来排列(第一个拥有最高的优先级)。服务器可能选择这种顺序，也可能不选择。象素数据也可以使用“原始编码”如果没有具体说明。

除了基本的编码方式，客户端也可以请求“伪编码”通告服务器它支持某一种扩展协议。如果服务器不支持这种扩展，它就会忽略这种伪编码。注意：这意味着客户端在得到服务器的确认之前都要假设服务器并不支持它的扩展。

参照6.6(每种编码方式的描述) 6.7(伪编码的含义)。

字节数	类型 [值]	描述
1	U8 2	消息类型
1		填充
2	U16	编码编号

接下来就是编码编号的重复

字节数	类型 [值]	描述
4	S32	编码类型

6.4.3 帧缓存更新请求

客户端只在乎帧缓存区域的x、y、宽度和高度。服务器一般通过发送帧缓存更新来响应更新请求。当然，有可能一个帧缓存更新可能是多个请求的响应结果。

通常服务器假设客户端有所有它所感兴趣的帧缓存，因此服务器只需要进行增量更新就 OK 了。

但是如果因为某种原因，客户端丢失某一部分必须的内容，那么它发送帧缓存更新请求的时候就会将增量设置为零。这样就会请求服务器尽快把所需内容进行发送。而这块不会使用 CopyRect 编码方式进行更新。

反之，客户端就会增量设置为非零。如果只是在特定区域发生改变，那么服务器就会发送帧缓存更新。注意：在请求和更新之间可能存在不确定时间段。

在客户端比较快的情况下，客户端有可能会规定增量请求的速率，这样可以避免无端占用网络。

字节数	类型 [值]	描述
1	U8 3	消息类型
1	U8	增量标志
2	U16	x 坐标
2	U16	y 坐标
2	U16	宽度
2	U16	高度

6.4.4 按键事件

某一个键的按下与释放。如果某一个键被按下，那么按下标志非零。释放的时候变为零。在 X Window 系统中键本身被赋值为“keysym”。

字节数	类型 [值]	描述
1	U8 4	消息类型
1	U8	按下标志
2		补充
4	U32	键号

对于大多数键来说，“keysym”与 ASCII 码相对应，具体参考《The Xlib Reference Manual》或者参考<X11/keysysmdef.h>。部分按键对应如下：

Key name	Keysym value	Key name	Keysym value
BackSpace	0xff08	F1	0xffbe
Tab	0xff09	F2	0xffbf
Return or Enter	0xff0d	F3	0xffc0
Escape	0xff1b	F4	0xffc1
Insert	0xff63	...	...
Delete	0xffff	F12	0xffc9
Home	0xff50	Shift (left)	0xffe1
End	0xff57	Shift (right)	0xffe2
Page Up	0xff55	Control (left)	0xffe3
Page Down	0xff56	Control (right)	0xffe4
Left	0xff51	Meta (left)	0xffe7
Up	0xff52	Meta (right)	0xffe8
Right	0xff53	Alt (left)	0xffe9
Down	0xff54	Alt (right)	0xffea

下扌 遵守以

在美式键
键盘的服
，这样的

对待。比如：
Shift 按键

的是根据

eysyms。
户端来发

”来让

keysyms 来正确的解释。例如，在德国的个人计算机键盘中，ctrl+alt+q 产生@字符，这样的话，我们必须让客户端正确的输入。

- 在 X window 系统中没有统一的“Backward”按键，在某些系统中，shift+tab 产生“ISO_Left_Tab”，而在其他可能提供一个“BackTab”，也有“Tab”和应用来告诉服务器 shift 的状态表示 backward-tab 而不是 forward-tab。在 RFB 协议中更接近后者的实现方式。客户端应该产生一个切换 Tab 而不是“ISO_Left_Tab”. 尽管如此，为了与目前客户端向后兼容，服务器应该把 ISO_Left_Tab 看做为变换的 Tab 键。

6.4.5 鼠标(指针)事件

检测指针移动或者某一个键的按下或释放。指针目前在(x 坐标、y 坐标），按钮的 1 到 8 状态通过 0 到 7 位来表示，0 表示松开，1 表示按下。

拿普通鼠标来说，1，2，3 分别响应左、中、右键。对于滑轮鼠标来说，滚轮向上表示 4 键的按下和释放，而向下表示 5 键的按下和释放。

字节数	类型 [值]	描述
1	U8 5	消息类型
1	U8	按键屏蔽
2	U16	x 坐标
2	U16	y 坐标

6.4.6 客户端文本剪切

客户端有新的 ISO8859-1(Latin-1) 文本在它的剪切缓存里，行

的末尾通过新行字符(值为 10)来表示。 需要无回车（值为 13)。目前还没有找到传输非 Latin-1 字符集的方法。

字节数	类型 [值]	描述
1	U8 6	消息类型
3		填充
4	U32	长度
长度	U8 数组	文本

6.5 服务器到客户消息

服务器到客户消息在本文中定义如下：

号码	名称
0	帧缓存更新
1	设置颜色面板条目
2	响铃
3	服务器剪切文本

其余注册的消息类型：

号码	名称
255	Anthony Liguorui

注意在服务器发送消息之前必须确认客户端支持相关扩展，通常在请求“伪编码”的时候使用。

6.5.1 帧缓存更新

帧缓存更新是由一系列像素数据矩形而组成，这些矩形会被客户端送入它的帧缓存中。它是对客户端帧缓存更新请求的响应。而在请求和响应之间有可能存在不确定时期。

字节数	类型 [值]	描述
1	U8 0	消息类型
1		填充
2	U16	矩形编号

字节数	类型 [值]	描述
2	U16	x 坐标
2	U16	y 坐标
2	U16	宽度
2	U16	高度
4	S32	编码类型

随着像素数据矩形的个数，每个矩形包括以下内容：

字节数	类型 [值]	描述
2	U16	x 坐标
2	U16	y 坐标
2	U16	宽度
2	U16	高度
4	S32	编码类型

对应像素数据也是由特定的编码方式，参照 6.6（编码的数据格式)参照 6.7（伪编码的含义）

6.5.2 设置颜色面板条目

当像素格式使用“颜色面板”时，消息告诉客户端对应像素值如何映射为 RGB 亮度。

字节数	类型 [值]	描述
1	U8 1	消息类型
1		填充
2	U16	第 1 种颜色
2	U16	颜色数

对应颜色数字：

字节数	类型 [值]	描述
2	U16	红
2	U16	绿
2	U16	蓝

6.5.3 响铃

如果有响铃事件，就在客户端上响铃。

字节数	类型 [值]	描述
1	U8 2	消息类型

6.5.4 服务器剪切文本

同客户端剪切文本（参照 6.4.6)

6.6 编码

本文中定义的编码类型如下：

号码	名称
0	Raw
1	CopyRect
2	RRE
5	Hextile
16	ZRLE
-239	Cursor 伪编码
-223	DesktopSize 伪编码

其余注册的编码方式：

号码	名称
4	CoRRE
6,7,8	zlib, tight, zlibhex
-272 to -257	Anthony Liguori
-256 to -240	
-238 to -224	
-222 to -1	tight 选项

6.6.1 Raw 编码

最简单的编码类型就是 Raw 像素数据，这种情况下，数据是由宽 X 高组成的（宽、高分别表示矩形的宽高）。像素值就是简单的通过从左到右的扫描顺序来反映。所有的 RFB 客户端必须能够处理使用 Raw 编码的像素数据， 并且 RFB 服务器只能发送 Raw 编码的数据，除非客户端特别声明要求其他的编码方式。

字节数	类型 [值]	描述
宽 x 高 x 字节/ 象素	PIXEL 数组	象素

6.6.2 CopyRect 编码

CopyRect 编码方式对于客户端在某些已经有了相同的象素数据的时候是非常简单和有效的。这种编码方式在网络中表现为 x,y 坐标。让客户端知道去拷贝那一个矩形的象素数据。它可以应用于很多种情况。最明显的就是当用户在屏幕上移动某一个窗口的时候，还有在窗口内容滚动的时候。在优化画的时候不是很明显，一个比较智能的服务器可能会发送一次，因为它知道在客户端的帧缓存里已经存在了。接下来使用 CopyRect 编码方式发送相同的式样。

字节数	类型 [值]	描述
2	U16	x 源位置
2	U16	y 源位置

6.6.3 RRE 编码

RRE 表示提升和运行长度，正如它名字暗示的那样，它实质上表示二维向量的运行长度编码。RRE 把矩形编码成可以被客户机的图形引擎翻译的格式。RRE 不适合复杂的桌面，但在一些情况下比较有用。

RRE 的思想就是把像素矩形的数据分成一些子区域，和一些压缩原始区域的单元。最近最佳的分区方式一般是比较容易计算的。

编码是由像素值组成的，Vb(基本上是在矩形中最常用的像素值）和一个计数 N，紧接着是 N 的子矩形列表，这些里面由数组<v,x,y,w,h>组成，(x,y)是对应子矩形的坐标，表示子矩形上- 左的坐标值，(w,h) 则表示子矩形的宽高。客户端可以通过绘制使用背景像素数据值，然后再根据子矩形来绘制原始矩形。

在传输中，数据以下列描述开始；

字节数	类型 [值]	描述
4	U32	子矩形数目
字节每象素	PIXEL	背景象素值

对应子矩形的结构如下：

字节数	类型 [值]	描述
字节每象素	PIXEL	子矩形象素值
2	U16	x-position
2	U16	y-position
2	U16	宽
2	U16	高

6.6.4 Hextile 编码

Hextile 是 RRE 编码的变种，矩形被分割成 16x16 小片，允许每个小片的维数为 4 位，总共 16 位。矩形被分割的小片从上开始，遵守自左到右，自顶向下的顺序。小片的编码内容按照预定的顺序进行编码。如果整个矩形的宽度不是 16 的整数倍，那么每行最后的小片也相应减少。高度也类似。

每个小片可以使用 raw 编码，也可以是 RRE 编码的变种。每个小片有一个背景像素值。但是，如果小片的背景像素值和前一个小片相同，那么就不需要明确定义。如果小片的子矩形有相同的像素值，那么前景像素值就可以只定义一次。和背景像素值一样，前景像素值也可以通过前一个小片获得。

因此由小片组成的数据是按照顺序进行编码的。每一个小片以子编码类型的字节开始。它是位数的屏蔽组成。

字节数	类型 [值]	描述
1	U8	子编码掩码:
	1	Raw
	2	背景定义
	4	前景定义
	8	任意子矩形
	16	子矩形着色位

如果 Raw 位被设置，那么其余的位就无效；接着是宽 X 高像素值（宽和高是小片的宽高）。否则其他的位就有效。

背景定义- 如果设置，那么像素值就会跟着小片的背景色：

字节数	类型 [值]	描述
字节每象素	PIXEL	背景象素值

在矩形中的第一片非 Raw 小片必须设置这一位，如果不设置，那么它的背景就会和上一片相同。

前景定义- 如果设置，那么像素值就会定义小片中所有子矩形的前景色。

字节数	类型 [值]	描述
字节每象素	PIXEL	前景象素值

如果这一位被设置，那么子矩形着色位必须为 0。

任意子矩形- 如果设置，那么一个字节包含着子矩形的个数。

字节数	类型 [值]	描述
1	U8	子矩形数目

如果这一位不设置，那么就不会有子矩形。（例如，整个小片就是背景颜色）

子矩形着色- 如果设置，那么任意子矩形的像素值的优先级都高于子矩形的颜色定义，因此子矩形是：

字节数	类型 [值]	描述
字节每象素	PIXEL	子矩形象素值
1	U8	x-and-y-position
1	U8	宽和高

如果不设置，所有子矩形都是前景色的颜色，如果前景定义没有设置，那么前景色和前一个片的相同。子矩形就是：

字节数	类型 [值]	描述
1	U8	x-and-y-position
1	U8	宽和高

每一个子矩形的位置和大小都是使用两位进行定义，x-and-y-position 和 width-and-height。最重要的四位 x-and-y-position 定义 X 的位置，不重要的定义 Y 位置。最重要的四位 width-and-height 定义宽度-1，不重要的定义高度-1。

6.6.5 ZRLE 编码

ZRLE(Zlib Run-Length Encoding),它结合了 zlib 压缩，片技术、调色板和运行长度编码。在传输中，矩形以 4 字节长度区域开始，紧接着是 zlib 压缩的数据，一个单一的 zlib“流”对象被用在 RFB 协议的连接上，因此 ZRLE 矩形必须严格的按照顺序进行编码和译码。

字节数	类型 [值]	描述
4	U32	长度
长度	U8 array	zlibData

---

zlibData 在没有压缩之前，代表了由 64x64 像素组成的从左到右，从高到低的顺序的片，和 hextile 编码有点类似。如果整个矩形的宽度不是 64 的整数倍，那么每行最后的小片也相应减少。高度也类似。

ZRLE 编码利用了一种新的压缩像素 CPIXEL(Compressed PIXEL)。这个和 PIXEL 有着相同的像素格式，除了真彩标志是非零，位每像素是 32，色深不大于 24。所有的位组成红，绿和蓝的亮度填充最不重要的或最重要的三字节。如果 CPIXEL 只有 3 字节长，并且包含有合适的最不重要或最重要 3 字节。那么 bytesPerCPixel 就是 CPIXEL 的字节数。

每片都是以子编码类型字节开始，如果片被使用运行长度编码，那么本字节的最高位就会被设置。其余 7 位表示绘图样式- 零表示没有样式，1 表示片为单色，2-127 表示对应

的样式。可能的子编码值如下：

0-Raw 像素数据 宽 X 高像素值（宽和高为对应片的宽和高，对应像素值如下：

字节数	类型 [值]	描述
宽 x 高 x 字节/CPIXEL	CPIXEL 数组	象素

1-单色的片，对应像素值如下：

字节数	类型 [值]	描述
字节/CPIXEL	CPIXEL 数组	象素

2-16-打包的样式类型。对应像素值是由 paletteSize(=子编码)像素值，打包像素值组成，每个打包像素值表示为一位区域服从样式索引（0 表示第一个条目），对应 paletteSize 2，1 位被使用，paletteSize 3，4 有两位被使用，从 5-16 均有 4 位区域被使用。位的区域被打包成字节，最重要的位表示最左边像素。因为片并不是 8,4,2 像素宽的乘积，所以填充位被用来按照字节数排列每一个行。

字节数	类型 [值]	描述
调色板 x 字节/CPIXEL	CPIXEL 数组	调色板
m	U8 数组	打包象素

m 表示打包像素的字节数。对于 paletteSize 2 就是 floor((width + 7)/8) x height，相应 3,4 就是 floor((width + 3)/4) x height，而 5-16 就是 floor((width + 1)/2)x height。

17-127 未使用(对于 palette RLE 并没有什么优势）。

128-简单 RLE 它由一些不断重复的执行组成，一直到片结束。执行可能从一行的结束到另一行的开始。每一次运行是通过一个像素值和像素值长度来表示的。长度一般为 1 个或多个字节。经过计算多于所有字节总和+1 作为长度。除了 255 任何字节值都隐含最后的字节。例如长度 1 表示为[0]，255 表示为[254]，256 表示为[255，0]，257 表示为[255，1]，510 表示为[255，254]，511 表示为[255，255，0]等等。

字节数	类型 [值]	描述
字节/CPIXEL	CPIXEL 数组	象素值
floor((runLength-1)/255)	U8 数组 255	
1	U8	(runLength - 1)%255

129-未使用

130-255 调色 RLE。调色紧跟其后，由 paletteSize = (subencoding - 128) 像素值组成：

字节数	类型 [值]	描述
调色板大小 x 字节/CPIXEL	CPIXEL 数组	调色板

接下来就合简单 RLE 相似，一些不断重复的执行组成，一直到片结束。执行长度通过调色板索引来表示。

字节数	类型 [值]	描述
1	U8	调色板索引

如果执行长度使用多于一位来表示调色板索引，并且最高位被设置。那么就会带有执行长度。

字节数	类型 [值]	描述
1	U8	调色板索引+128
floor((runLength - 1)=255)	U8 数组 255	
1	U8	(runLength – 1)%255

6.7 伪编码

6.7.1 指针/ 鼠标伪编码

如果客户端请求指针/ 鼠标伪编码，那么就是说它有能力进行本地绘制鼠标。这样就可以明显改善传输性能。服务器通过发送带有伪鼠标编码的伪矩形来设置鼠标的形状作为更新的一部分。伪矩形的 x 和 y 表示鼠标的热点，宽和高表示用像素来表示鼠标的宽和高。包含宽 X 高像素值的数据带有位掩码。位掩码是由从左到右，从上到下的扫描线组成，而每一扫描线被填充为 floor((width+7)/8)。对应每一字节最重要的位表示最左边像素，对应 1 位表示相应指针的像素是正确的。

字节数	类型 [值]	描述
宽 x 高 x 字节/PIXEL	PIXEL 数组	指针象素
floor((宽 + 7)=8) x 高	U8 数组	位掩码

6.7.2 桌面大小伪编码

如果客户端请求桌面大小伪编码，那么就是说它能处理帧缓存宽/ 高的改变。服务器通过发送带有桌面大小伪编码的伪矩形作为上一个矩形来完成一次更新。伪矩形的 x 和 y 被忽略，而宽和高表示帧缓存新的宽和高。没有其他的数据与伪矩形有关。