http://blog.163.com/waft_xu/blog/static/438152942008124183624 8/

MIDI 文件格式分析

VC 2008-02-02 16:18:36 阅读 769 评论 2 字号：大中小

MIDI 文件属于二进制文件，这种文件一般都有如下基本结构：文件头+数据描述

文件头一般包括文件的类型，因为 Midi 文件仅以.mid 为扩展名的就有 0 类和 1 类两种，而大家熟悉的位图文件的格式就更多了，所以才会出现文件头这种东西。

而数据描述部份是主体，我们现在来一起分析它的结构：

在每个 Midi 文件的开头都有如下内容，它们的十六进制代码为：“4d 54 68 64 00 00 00 06 ff ff nn nn dd dd”。

前四个是 ASCII 字符“MThd”是用来鉴别是否 Midi 文件，而随后的四个字节是指明文件头描述部分的字节数，它总是 6，所以一定是“00 00 00 06”，以下是剩余部分的含义：

ff ff	指定 Midi 的格式	00 00	单音轨
		00 01	多音轨，且同步。这是最常见的
		00 02	多音轨，但不同步
nn nn	指定轨道数	实际音轨数加上一个全局的音轨	
dd dd	指定基本时间	一般为 120(00 78)，即一个四分音符的 tick 数，tick 是 MIDI 中的最小时间单位	

以上就是 MIDI 文件头了，后面的所有内容都是真正做事的，我们先来看看它的构成。

MIDI 的数据是由若干个格式相同的子数据构成的，这些子数据在多音轨的格式中记录了一个轨道的所有信息。多加一个音轨，就简单地把数据追加在前一音轨的后面就可以了，不过**不要忘记更改文件头中的 nn nn(轨道数)**。

先看全局音轨。全局音轨包括歌曲的附加信息(比如标题和版权)、歌曲速度和系统码(Sysx)等内容。

不管是全局音轨还是含有音符的音轨，都以“4D 54 72 6B”开头，它其实是 ASCII 字符“MTrk”，其后跟着一个 4 个字节的整数，它标志了该轨道的字节数，这不包括前面的 4 个字节和本身的 4 个字节。这一点，我们可以在后面的例子中去理解。

接着就是记录数据的地方了，每一个数据有着相同的结构:时间差+事件。

所谓时间差，指的是前一个事件到该事件的时间数，它的单位是 tick(MIDI 的最小时间单位)。它的构成比较特殊，这里要用二进制来说明。

一个字节有 8 位，如果仅使用 7 位，它可以表示 0~127 这 128 个数，而剩下的一位，则用来作为标志。如果要表示的数在以上范围，则这个标志为 0，这时，一个 7 位的字节可以表示 0~127tick。如果要表示的数超出了这个范围(比如 240)，则把标志设置成 1，然后记录下高 7 位，剩下的留给下一个字节，在该例中 240 可以分解成 128*1+112，这里的 1 就是第一个字节要记录的，加上标志位，应该为 10000001，即十六

进制的 81；而 112 是下一个字节记录的，它的十六进制为 70:所以要表示 240 这个时间，要写成 81 70。同理，如果要表示 65535tick，则可以先计算出 65535=128^2*3+128^1*127+128^0*127，然后得出结果:83 FF 7F。由此，我们反过来也可以知道如何确定时间差:只要标志位为 0，则表示结束读取时间差。比如 82 C0 03 表示 128^2*2+128^1*64+128^0*3=40963，如果基本时间为 120，则有 341:043 个四分音符。

以这种方式记录整数的字节称为**动态字节**，它根据记录的整数改变自身的长度，这在后面还要用到，所以必须熟练计算。

看完了这么麻烦的东西，我们再来看个更麻烦的东西:**事件**。在这些标准的解释后面，我们会通过一些例子来进一步掌握这些内容。

事件大体上可以分为音符、控制器和系统信息这几个种类。对于这些事件，都有统一的表达结构:种类+参数。

对于一个音符，由于它的有效范围是 0~127，所以直接用 00~7F 作为“种类”，可以认为是个音符，比如 3C 表示中央 C。而一个音符的最重要的参数是力度(也叫速度:velocity)。比如，3C 64 表示一个力度为十进制 100 的中央 C 音符。

因为一个字节有 8 位，所以剩余的一位如果置 1，再联合其他的 7 位，则可以表示各种信息。我们暂且无视一个音轨到底是全局的还是用于记录音符的。它们归根结底都是用来记录各种事件的，只不过有些应出现在全局音轨比较合乎逻辑而已。既然这样，我们就可以从下面的表来看事件:

下表中，x 表示音轨 0~F，比如 81 表示松开第二轨的音符。

种类		参数(十六进制)
字节	含义	
8x	松开音符	音符(00~7F):松开的音符
		力度:00~7F
9x	按下音符	音符(00~7F):按下的音符
		力度:00~7F
Ax	触后音符 (Key After Touch)	音符:00~7F
		力度:00~7F
Bx	控制器	控制器号码:00~7F
		控制器参数:00~7F
Cx	改变乐器	乐器号码:00~7F
Dx	触后通道	值:00~7F
Ex	滑音	音高(Pitch)低位:Pitch mod 128

		音高高位:Pitch div 128
F0	系统码	系统码字节数:动态字节
		系统码:不含开头的F0,但包括结尾的F7
FF	其他格式	程式种类:00~FF
		数据占用的字节数:动态字节
		数据:个数由上一参数确定
00~7F	上次激活格式的参数(8x、9x、Ax、Bx、Cx、Dx、Ex)	

下表详细地列出了FF的详细情况，对于字节数由数据决定的情况，表中以“--”表示。

种类		字节数	数据
字节	含义		
00	设置轨道音序	02	音序号 00 00~ FF FF
01	歌曲备注	--	文本信息
	音轨文本		文本信息
02	歌曲版权	--	版权信息
03	歌曲标题	--	歌曲标题:用于全局音轨，第一次使用表示主标题，第二次表示副标题
	音轨名称	--	音轨名
04	乐器名称	--	音轨文本(同01/2)
05	歌词	--	歌词
06	标记	--	用文本标记(Marker)
07	开始点	--	用文本记录开始点(同01/2)
2F	**音轨结束标志**	**00**	**无**
51	**速度**	**03**	**3字节整数，1个四分音符的微秒数**
58	**节拍**	**04**	**分子**
			分母:00(1),01(2),02(4),03(8)等

			节拍器时钟
			一个四分音符包含的三十二分音符的个数
59	**调号**	**02**	**升降号数:-7~-1(降号),0(C),1~7(升号)**
			大小调:0(大调), 1(小调)
7F	音序特定信息	--	音序特定信息

这些就是MIDI结构的全部内容

MIDI文件格式分析——实践篇

要书写二进制(十六进制)文件，应该准备好一些工具，比如我自己用的是VC++，因为学习MIDI格式无非是想写它的软件，既然VC++可以编辑二进制文件，就将就着用吧。其次，应该找个可以编辑和播放MIDI文件的软件，比如Cakewalk，这样就可以开始了。

首先书写文件头“4d 54 68 64 00 00 00 06”，我们直接写同步多音轨的格式，先写1个音轨，并以120为一个音符的基本时间。这样，随后的字节是:“00 01 00 01 00 78”。

现在，如果用Cakewalk打开会失败，因为我们指定的音轨数为1，但是并没有书写任何音轨，如果改成“00 01 00 00 00 78”再打开，就不会出问题了。所以，今后如果更改了音轨数，千万不要忘记向“上头”汇报。

把轨道数改回01，继续我们的实验。先写音轨的头信息:“4D 54 72 6B”（MTrk)，因为我们还不能确定后面有多少字节，所以先把它假设成“00 00 00 00”，以后再回来改。

我 们先尝试设置歌曲的速度和节拍等基本信息。假设一个四分音符的时间是半秒，即0.5*106微秒。它的十六进制数是07A120,再看事件表，设置速度是 51,但是在其前面必须是FF，然后它须要3个字节作为参数，因此字节数为03，参数为“07 A1 20”，也就是“FF 51 03 07 A1 20”。这是事件部分，不要忘记在其之前有个参数——时间差。这是一开始就应该设置的参数，因此时间差为00。所以，完整的事件应该是“00 FF 51 03 07 A1 20”，我们把这一段追加在Midi文件末尾。

这时先不要急着用Cakewalk验证，因为我们还没有向“上 级”报告，没错，把前面表示字节数的“00 00 00 00”改成“00 00 00 07”，如果用VC++作为二进制文件的编辑器，选择了事件后，可以在状态栏看到选择的字节长。保存后，再用Cakewalk打开，就可以看见速度是 120。

我们再来设置节拍和调号，因为一般用 Cakewalk 新建一个 Midi 会默认地设置成 4/4，C 大调，我们就改设成 6/8，A 大 调。查阅事件表知道，58 和 59 是分别用来设置节拍和调号的。虽然设置节拍的参数很多，但在现在的系统中，后两个参数是被忽略的，而且 Cakewalk 还 会对其进行修正。因此，我们只要设置好实际有用的就可以了。分子是 6，分母是 8，所以第一个参数是 06，第二个参数是 03(23=8)。 最后，补上前面的时间差和后面的两个被忽略的参数，它应该是“00 FF 58 04 06 03 00 00”；再看调号，A 调有 3 个升号，因此可以这样的事件可以表示为“00 FF 59 02 03 00”。事实上，大小调是个被忽略的参数。我们统计一下至今为止事件的字节数，然后更改前面的参数，即把“00 00 00 07”改成“00 00 00 15”。保存后用 Cakewalk 打开，再进入五线谱窗口，就可以马上验证了。细心的你可能已经发现，进入五线谱窗口前和平常有些延迟，这是因为我们并没 有设置好那些可以忽略的字节，而 Cakewalk 就是在对其进行重新验证，这一点，我们以后再讨论。

用同样的方法，您可以很容易地设置歌曲的标题和版权，这作为一个练习，在这里就不多写了。我们现在学习写一个含有音符的轨道。首先您应该知道要做哪些事：1、写新音轨的信息头；2、向上级汇报多了一个音轨。接下来，我们开始写入一个简单的音符。

假 设向第一拍写一个中音 A，这里可能要先说明一下，音符是从 C0 开始一起向上数的，数到中央 C(C5)是十六进制的 3C，则中音 A 应该为 45，在附件中有详细的计算方法。我们知道在音乐中一个音符通常有三个属性：音高、力度和时值。可是我们在事件表中并没有看见有什么可以直接设置音符时值的标志。不错，事实 上，音符的时值是由按下的时间和松开的时间决定的。我们假设要写入一个八分音符。它的 Tick 数是四分音符的一半，即 60，十六进制表示成 3C。我们先来 看看与音符有关的标志。

在事件表中，9x 是用来打开一个音符，我们这里假设使用第 7 个通道(注意到 MIDI 有 16 个通道(Channel)，而第 10 个被默认地用作打击乐，所以，我们在这个阶段(没有学习 Sysx 之前)，先不要使用第 10 个通道)，则 9x 中的 x 是 6；再看它的参数，一个是音符，这里我们写入 45，第二个是力度，我们用 70，因为是一开始就触发的，所以前面的时间差还是 00。这样我们就在第 5 个通道以力度 112 按下了一个中音 A。对应的字节描述是“00 96 45 70”。它的时值不用想都知道一定是 0，这取决于什么时候把它松开。

特 别地，如果一个音符的力度为 0，则 MIDI 认为用户想松开这个键，因为 9x 已经打开了通道，所以我们直接写入一个带 00 力度的同一音符就可以决定这个音符 的时值了。根据前面的分析，这个时间差应该是 3C，所以我们在写入 3C 后写上音符 45 和它的力度 00，即“3C 45 00”。统计好字节数并向这一轨的头信息中更新，然后保存到磁盘，用 Cakewalk 打开并进入事件列表窗口便可以验证了。

在这个基础 上，我们再尝试在 A 的后面增加一个四分音符中音#G。因为 96 已经打开了通道，我们没有必要每次都使用 9x，只要输入事件信息即可。对于中音#

G，它的十 六进制是44，相对刚才输入00力度的A来说时间差为00，因此可以表示成“00 44 64”，这里我们已经假设力度为100；然后是松开它，因为是四分音符，所以时间差是78H，别忘记力度是00，它的字节应表示成“78 44 00”，做好后面的工作，然后验证看对不对。

我们再做个稍微复杂一点的实验：在原来的基础上，在同一轨的第一拍加上一个附点四分中音D。这里就不能再使用追加的方法了，因为前面的事件已经过了3个八分音符的时间，无论再加上什么，都只会发生在后面，所以我们要在前面插入一些字节。

9x 已经打开了通道，我们直接在9x按下的音符后加上一个音符事件“00 3E 64”，这里的00显然是个时间差，3E是中音D，64是力度，也就是说，在按下中音A的同时按下了中音D。我们又按下一个键了，要在什么时候，在哪里松开才能保证输入的是个附点八分音符呢？首先，它的时值是3个八分音符，即180，这里还有一点要注意，180是个大于128的数，它的动态字节就应该表示成“81 34”，在哪里输入才好呢？如果你觉得在按下D后输入，或者在任何什么地方输入这个时间差，然后再写上“3E 00”可以表示松开的话就完全误解了时间差的概念。其实，我们只要简单地在松开#G的时候松开D就可以了，所以应该在末尾补上“00 3E 00”。统计好字节数后到Cakewalk中去验证验证吧。

到目前为此，我们应该可以输入任何形式的音符了，不过MIDI除了音符以外，还可以包括各种控制器和系统码，它们的地位不亚于音符，我们现在马上学习如何使用控制器。

控 制器比音符要简单多了，我们尝试在#G前加入相位控制(Pan)，它的十进制代码是10，十六进制是0A，我们将参数设置成111，即十六进制的6F。首 先查得控制器是Bx，这里的x和上面一样，也是6。接下来写入控制器号0A，然后是参数6F，别忘了前面的时间差是00。所以这段字节是“00 B6 0A 6F”，它应放在松开#G的事件之前，与按下#G同时。不过，一旦使用了非音符，而后面还有音符事件时，则必须重新通知打开音符，这说起来复杂，做起来还 是比较容易的，我们只要稍微改写下一个音符事件即可：原本是“时间差+音符+力度”，我们加入一个打开音符的标志，成为“时间差+9x+音符+力度”即 可。校验过头信息后，去Cakewalk中进行更进一步的检验便知它的可行。

其实，时间差为00的控制事件如果出现的时间也是00，则在Cakewalk中会尽可能地把它们放在轨道信息中，而不在事件列表中重复。我们可以利用这一点给音轨设置初始乐器和音量，这就作为一个练习，在此就不再说明了。

至 于其他的诸如触后键等与控制器类似的格式在此就不多说了。在这里有必要提醒的是滑音。滑音的乐理范围是-8192~8191，但是在使用时参数是个正数，比如要设置成0，则应该是0-(-8192)=8192，它才是参数。8192的7位双字节表示成“8192 mod 128=00H；8192 div 128=40H”。如果时间差是00，则应表示成“00 E6 00 40”

最 后我们看看系统码。系统码的构成本来是“F0 厂家 ID 设备号码 格式代码 传送命令 具体参数 F7”，而在文件中，则不以开头的“F0”为系统码，而字节数也仅记录剩余的系统码,比如 XG 的复位码是“F0 43 10 4C 00 00 7E 00 F7”,则在文件中应写成“00 F0 08 43 10 4C 00 00 7E 00 F7”，其中第一个 00 是时间差，F0 是系统码标志，08 是后面的字节数。有一点要注意的是，几个系统码不可以写在一起，比如“00 F0 0D 43 10 4C 00 00 7E 00 F7 F0 AA BB CC F7”或“00 F0 0C 43 10 4C 00 00 7E 00 F7 AA BB CC F7”都是不好的写法。如果存在以上系统码集，可以分成两个事件:“00 F0 08 43 10 4C 00 00 7E 00 F7 00 F0 04 AA BB CC F7”

当然系统码可以写在任何音轨,不过一般我们会考虑把歌曲播放前发送的系统码写在全局音轨中，并把时间差设成 00。

作为一个参考，这里再附上一个 MIDI 样本。

虽 然我们只讨论了同步多音轨的格式，其实对于其他两种，比如较常见的单音轨格式，所有的事件只写在一个音轨中，即只要存在一个“MTrk”就足够了。而相对 地，用于记录音轨数的两个字节也永远为“00 01”，连续事件如果出现的通道不同，也必须重新指定通道(8x~Ex)。在此不详细讨论了。

到目前为此，我们应该可以构造出任何 MIDI 音乐了，作为一些辅助性的参考，可以查阅下一篇连载《MIDI 文件格式分析——附件篇》。另外，对于扩展名为.rmi 的格式，可以参阅下下篇连载《MIDI 文件格式分析——RMI 篇》。

MIDI 文件格式分析——附件篇

该文档是前两编的补充，主要讲述以下内容:

音符十六进制的计算

关于乐器选择

RPN 和 NRPN

在 MIDI 中，中央 C 是 C5，最低音是 C0，最高音是 G5，要计算任何一个音符对应的十六进制，可以使用这个公式:

假设音符是 NO，表示第 O 八度的音名 N，比如 G2 中，N 为 G，O 为 2，则它的十进制为 O*12+N，N 的值为了简便起见，用下表给出:

音名	C	#C	D	#D	E	F	#F	G	#G	A	#A	B
十进制值	0	1	2	3	4	5	6	7	8	9	10	11

这样 G2 的十进制值为 2*12+7=31，十六进制为 1F。

若知道音符的十六进制，也可以很容易求出音符，比如 64(16)=100(10)，

而 100 div 12=8，100 mod 12=4，对应音符为 E8。写成公式就是：N=B mod 12；O=B div 12；（设 B 为表示音符的字节的十进制数）

乐 器是 MIDI 中比较重要的因素，要选择所有的乐器不仅仅只是使用 Cx 号标志就能完成的，还必须结合 BankSelect(乐队选择)，而 BankSelect 其实是由 0 号控制器和 32 号控制器完成的，它们的十六进制代码分别是 00 和 20.比如要选择出 XG 标准中的 Slow Violin，它在第 08H 个乐队中的第 28H 号乐器中，所以它的完整代码应为“00 B0 00 00 00 20 08 00 C0 28”。我们来分析它的构成：这里我们假设时间差为 00，所有信息都发给通道 00，所以第一个 00 是时间差，B0 是打开控制器的标志，并指定发送到通道 00，接下来的 00 00，是由控制器号 00 和控制器参数 00 构成的，它事实上是表示 0 号控制器的参数为 0，即 BankSelect-MSB 的参数为 0，然后是下一个事件，它 是“00 20 08”，即时间差是 00，使用 20H 号控制器，参数为 08H，即 BankSelect-LSB 的参数是 08H，这样就指定了 Bank(乐队)。再下来就是 “00 C0 28”，就是所谓的 Patch Change 事件了，它的时间差为 00，参数是 28H。这样就完成了标准的乐器选择。

事实上，它是由三个事件共同完成的，如果某次选择乐器和上次的乐器有共同的参数，可以不必重复使用相关的 X 作。

在 本例中，您可能发现了一点，当连续使用同类 X 作时可以不必每次指定 X 作种类，比如这里的连续再次使用控制器，所以第二个控制器并没有使用 B0 作为标志，而 是直接使用控制器号码和它的参数。这一点和音符是一样的。其实，如果连续使用 Patch Change 事件(我是说如果)，则也可不必每次都写 Cx，打开了一次就可以了；不过就 Patch Change 事件而言，连续地更换乐器的结果是仅最后一个有效而已。

在前面的文档中并没有提及 RPN 和 NRPN，其实它们是由四个连续的控制器来实现的。我们假设要使用 RPN 事件的 Coarse Turning，它的参数假设是 4096，则它的字节是“00 B0 65 00 00 64 02 00 06 20 00 26 00”，我们来分析这段字节：首先我们先看看 RPN 是由哪四个控制器组成的——首先设置 RPN-MSB 和 RPN-LSB，分别对应的控制器是 65H 和 64H，Coarse Turning 的 RPN 码是 2，所以 MSB 为 0，LSB 为 2；然后是设置 Data Entry MSB 和 Data Entry LSB，对应的控制器是 06H 和 26H，而 4096 div 128=32，4096 mod 128=0，对应的十六进制数分别是 20H 和 00H。因此就构成了上面的字节。而 NRPN 和 RPN 原理是一样的，只不过不用 RPN-MSB 和 RPN- LSB，而改用 N

RPN-MSB 和 NRPN-LSB 而已，它们对应的十六进制数分别为 63H 和 62H。