AVI 是音频视频交错(Audio Video Interleaved)的英文缩写，它是 Microsoft 公司开发的一种符合 RIFF 文件规范的数字音频与视频文件格式，原先用于 Microsoft Video for Windows (简称 VFW)环境，现在已被 Windows 95/98、OS/2 等多数操作系统直接支持。AVI 格式允许视频和音频交错在一起同步播放，支持 256 色和 RLE 压缩，但 AVI 文件并未限定压缩标准，因此，AVI 文件格式只是作为控制界面上的标准，不具有兼容性，用不同压缩算法生成的 AVI 文件，必须使用相应的解压缩算法才能播放出来。常用的 AVI 播放驱动程序，主要是 Microsoft Video for Windows 或 Windows 95/98 中的 Video 1，以及 Intel 公司的 Indeo Video。

在介绍 AVI 文件前，我们要先来看看 RIFF 文件结构。AVI 文件采用的是 RIFF 文件结构方式，RIFF（Resource Interchange File Format，资源互换文件格式）是微软公司定义的一种用于管理 windows 环境中多媒体数据的文件格式，波形音频 wave，MIDI 和数字视频 AVI 都采用这种格式存储。构造 RIFF 文件的基本单元叫做数据块（Chunk），每个数据块包含 3 个部分，

1、4 字节的数据块标记（或者叫做数据块的 ID）

2、数据块的大小

3、数据

整个 RIFF 文件可以看成一个数据块，其数据块 ID 为 RIFF，称为 RIFF 块。一个 RIFF 文件中只允许存在一个 RIFF 块。RIFF 块中包含一系列的子块，其中有一种字块的 ID 为"LIST"，称为 LIST，LIST 块中可以再包含一系列的子块，但除了 LIST 块外的其他所有的子块都不能再包含子块。

RIFF 和 LIST 块分别比普通的数据块多一个被称为形式类型（Form Type）和列表类型（List Type）的数据域，其组成如下：

1、4 字节的数据块标记（Chunk ID）

2、数据块的大小

3、4 字节的形式类型或者列表类型

4、数据

下面我们看看 AVI 文件的结构。AVI 文件是目前使用的最复杂的 RIFF 文件，它能同时存储同步表现的音频视频数据。AVI 的 RIFF 块的形式类型是 AVI，它包含 3 个子块，如下所述：

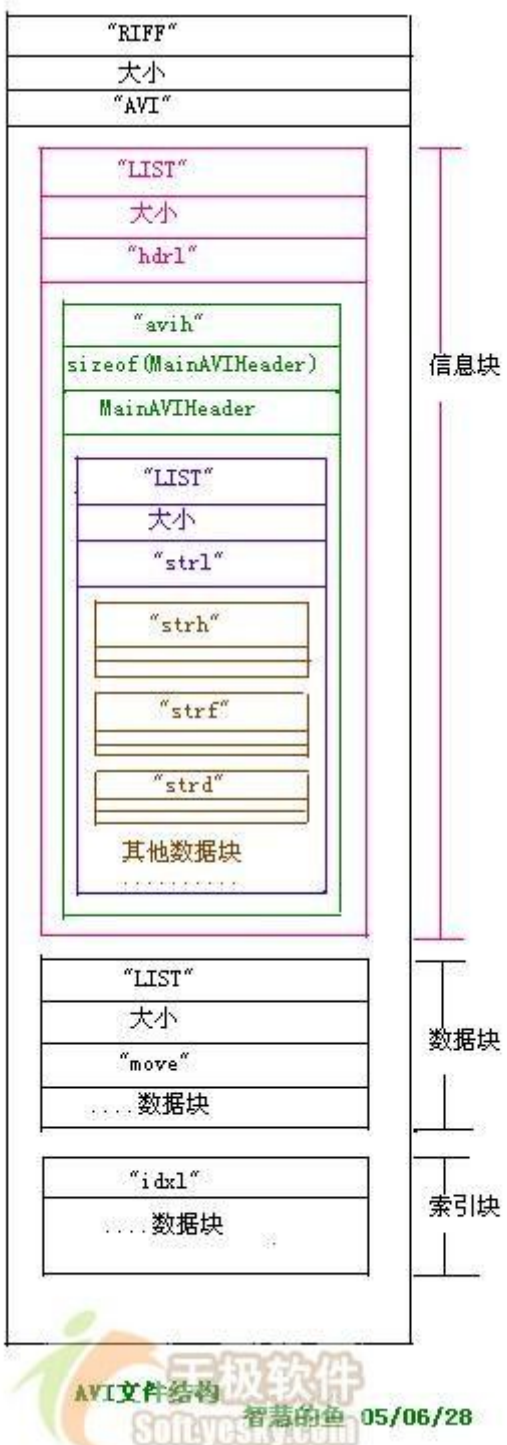
1、信息块，一个 ID 为"hdrl"的 LIST 块，定义 AVI 文件的数据格式。

2、数据块，一个 ID 为 "movi"的 LIST 块，包含 AVI 的音视频序列数据。

3、索引块，ID 为 "idxl"的子块，定义 "movi"LIST 块的索引数据，是可选块。

AVI 文件的结构如下图所示，下面将具体介绍 AVI 文件的各子块构造。

1、信息块，信息块包含两个子块，即一个 ID 为 avih 的子块和一个 ID 为 strl 的 LIST 块。



"avih"子块的内容可由如下的结构定义：

```
typedef struct
{
  DWORD dwMicroSecPerFrame ; //显示每桢所需的时间 ns，定义 avi 的显示速率
  DWORD dwMaxBytesPerSec; // 最大的数据传输率
  DWORD dwPaddingGranularity; //记录块的长度需为此值的倍数，通常是 2048
  DWORD dwFlages; //AVI 文件的特殊属性，如是否包含索引块，音视频数据是否交叉存储
  DWORD dwTotalFrame; //文件中的总桢数
  DWORD dwInitialFrames; //说明在开始播放前需要多少桢
  DWORD dwStreams; //文件中包含的数据流种类
  DWORD dwSuggestedBufferSize; //建议使用的缓冲区的大小，
  //通常为存储一桢图像以及同步声音所需要的数据之和
  DWORD dwWidth; //图像宽
  DWORD dwHeight; //图像高
  DWORD dwReserved[4]; //保留值
}MainAVIHeader;
```

"strl" LIST 块用于记录 AVI 数据流，每一种数据流都在该 LIST 块中占有 3 个子块，他们的 ID 分别是"strh","strf", "strd"；
"strh"子块由如下结构定义。

```
typedef struct
{
  FOURCC fccType; //4 字节，表示数据流的种类 vids 表示视频数据流
  //auds 音频数据流
  FOURCC fccHandler;//4 字节 ，表示数据流解压缩的驱动程序代号
  DWORD dwFlags; //数据流属性
  WORD wPriority; //此数据流的播放优先级
  WORD wLanguage; //音频的语言代号
  DWORD dwInitalFrames;//说明在开始播放前需要多少桢
  DWORD dwScale; //数据量，视频每桢的大小或者音频的采样大小
  DWORD dwRate; //dwScale /dwRate = 每秒的采样数
  DWORD dwStart; //数据流开始播放的位置，以 dwScale 为单位
  DWORD dwLength; //数据流的数据量，以 dwScale 为单位
  DWORD dwSuggestedBufferSize; //建议缓冲区的大小
  DWORD dwQuality; //解压缩质量参数，值越大，质量越好
  DWORD dwSampleSize; //音频的采样大小
  RECT rcFrame; //视频图像所占的矩形
}AVIStreamHeader;
```

"strf"子块紧跟在"strh"子块之后，其结构视"strh"子块的类型而定，如下所述；如果 strh

子块是视频数据流，则 strf 子块的内容是一个与 windows 设备无关位图的 BIMAPINFO 结构，如下：

```
typedef struct tagBITMAPINFO
{
  BITMAPINFOHEADER bmiHeader;
  RGBQUAD bmiColors[1]; //颜色表
}BITMAPINFO;

typedef struct tagBITMAPINFOHEADER
{
  DWORD biSize;
  LONG biWidth;
  LONG biHeight;
  WORD biPlanes;
  WORD biBitCount;
  DWORD biCompression;
  DWORD biSizeImage;
  LONG biXPelsPerMeter;
  LONG biYPelsPerMeter;
  DWORD biClrUsed;
  DWORD biClrImportant;
}BITMAPINFOHEADER;
```

如果 strh 子块是音频数据流，则 strf 子块的内容是一个 WAVEFORMAT 结构，如下：

```
typedef struct
{
  WORD wFormatTag;
  WORD nChannels; //声道数
  DWORD nSamplesPerSec; //采样率
  DWORD nAvgBytesPerSec; //WAVE 声音中每秒的数据量
  WORD nBlockAlign; //数据块的对齐标志
  WORD biSize; //此结构的大小
}WAVEFORMAT
```

"strd"子块紧跟在 strf 子块后，存储供压缩驱动程序使用的参数，不一定存在，也没有固定的结构。

"strl" LIST 块定义的 AVI 数据流依次将 "hdrl " LIST 块中的数据流头结构与"movi" LIST 块中的数据联系在一起，第一个数据流头结构用于数据流 0，第二个用于数据流 1，依次类推。

数据块中存储视频和音频数据流，数据可直接存于 "movi" LIST 块中。数据块中音视频数

据按不同的字块存放，其结构如下所述，

音频字块
"##wb"
Wave 数据流
视频子块中存储 DIB 数据，又分为压缩或者未压缩 DIB，
"##db"
RGB 数据流
"##dc"
压缩的图像数据流

看到了吧，avi 文件的图像数据可以是压缩的，和非压缩格式的。对于压缩格式来说，也可采用不同的编码，也许你曾经遇到有些 avi 没法识别，就是因为编码方式不一样，如果没有相应的解码，你就没法识别视频数据。AVI 的编码方式有很多种，比较常见的有 mpeg2，mpeg4，divx 等。

索引块，索引快包含数据块在文件中的位置索引，能提高 avi 文件的读写速度，其中存放着一组 AVIINDEXENTRY 结构数据。如下，这个块并不是必需的，也许不存在。

```
typedef struct
{
  DWORD ckid; //记录数据块中子块的标记
  DWORD dwFlags; //表示 chid 所指子块的属性
  DWORD dwChunkOffset; //子块的相对位置
  DWORD dwChunkLength; //子块长度
};
```

小知识：AVI 文件格式----摘自《DirectShow 实务精选》 作者：陆其明

AVI （Audio Video Interleaved 的缩写）是一种 RIFF（Resource Interchange File Format 的缩写）文件格式，多用于音视频捕捉、编辑、回放等应用程序中。通常情况下，一个 AVI 文件可以包含多个不同类型的媒体流（典型的情况下有一 个音频流和一个视频流），不过含有单一音频流或单一视频流的 AVI 文件也是合法的。AVI 可以算是 Windows 操作系统上最基本的、也是最常用的一种媒 体文件格式。

先 来介绍 RIFF 文件格式。RIFF 文件使用四字符码 FOURCC（four-character code）来表征数据类型，比如‘RIFF’、‘AVI ’、‘LIST’等。注意，Windows 操作系统使用的字节顺序是 little-endian，因此一个四字符码‘abcd’实际的 DWORD 值应为 0x64636261。另外，四字符码中像‘AVI ’一样含有空格也是合法的。

RIFF 文件首先含有一个如图 3.31 的文件头结构。

图 3.31 RIFF 文件结构

最 开始的 4 个字节是一个四字符码‘RIFF’，表示这是一个 RIFF 文件；紧跟着后面用 4 个字节表示此 RIFF 文件的大小；然后又是一个四字符码说明文件的 具体类型（比如 AVI、WAVE 等）；最后就是实际的数据。注意文件大小值的计算方法为：实际数据长度 + 4（文件类型域的大小）；也就是说，文件大小的值不包括‘RIFF’域和“文件大小”域本身的大小。

RIFF 文件的实际数据中，通常还使用了列表（List）和块（Chunk）的形式来组织。列表可以嵌套子列表和块。其中，列表的结构为：‘LIST’ list Size listType listData ——‘LIST’是一个四字符码，表示这是一个列表；listSize 占用 4 字节，记录了整个列表的大小；listType 也是一个四字符码，表示本列表 的具体类型；listData 就是实际的列表数据。注意 listSize 值的计算方法为:实际的列表数据长度 + 4(listType 域的大小)；也就是说 listSize 值不包括‘LIST’域和 listSize 域本身的大小。再来看块的结构：ckID ckSize ckData ——ckID 是一个表示块类型的四字符码；ckSize 占用 4 字节，记录了整个块的大小；ckData 为实际的块数据。注意 ckSize 值指的是实际的块 数据长度，而不包括 ckID 域和 ckSize 域本身的大小。（注意：在下面的内容中，将以 LIST （ listType （ listData ））的形式来表示一个列表，以 ckID （ ckData)的形式来表示一个块，如[optional element]中括号中的元素表示为可选项。）

接下来介绍 AVI 文件格式。AVI 文件类型用一个四字符码‘AVI ’来表示。整个 AVI 文件的结构为：一个 RIFF 头 + 两个列表（一个用于描述媒体流格式、一个用于保存媒体流数据） + 一个可选的索引块。AVI 文件的展开结构大致如下：

```
RIFF (‘AVI ’
  LIST (‘hdrl’
        ‘avih’(主 AVI 信息头数据)
       LIST (‘strl’
             ‘strh’ (流的头信息数据)
             ‘strf’ (流的格式信息数据)
            [‘strd’ (可选的额外的头信息数据)]
            [‘strn’ (可选的流的名字)]
            ...
           )
        ...
      )
  LIST (‘movi’
       { SubChunk | LIST (‘rec ’
                          SubChunk1
```

```
                    SubChunk2
                    ...
                   )
          ...
         }
         ...
        )
   [ 'idx1' (可选的 AVI 索引块数据) ]
  )
```

首 先，RIFF ('AVI ' …)表征了 AVI 文件类型。然后就是 AVI 文件必需的第一个列表——'hdrl'列表,用于描述 AVI 文件中各个流的格式信息(AVI 文件中的每一路媒 体数据都称为一个流)。'hdrl'列表嵌套了一系列块和子列表——首先是一个'avih'块，用于记录 AVI 文件的全局信息，比如流的数量、视频图像的 宽和高等，可以使用一个 AVIMAINHEADER 数据结构来操作：

```
typedef struct _avimainheader {
FOURCC fcc;    // 必须为 'avih'
DWORD  cb;     // 本数据结构的大小，不包括最初的 8 个字节（fcc 和 cb 两个域）
DWORD  dwMicroSecPerFrame;    // 视频帧间隔时间（以毫秒为单位）
DWORD  dwMaxBytesPerSec;      // 这个 AVI 文件的最大数据率
DWORD  dwPaddingGranularity; // 数据填充的粒度
DWORD  dwFlags;          // AVI 文件的全局标记，比如是否含有索引块等
DWORD  dwTotalFrames;    // 总帧数
DWORD  dwInitialFrames; // 为交互格式指定初始帧数（非交互格式应该指定为 0）
DWORD  dwStreams;        // 本文件包含的流的个数
DWORD  dwSuggestedBufferSize; // 建议读取本文件的缓存大小（应能容纳最大的块）
DWORD  dwWidth;          // 视频图像的宽（以像素为单位）
DWORD  dwHeight;         // 视频图像的高（以像素为单位）
DWORD  dwReserved[4];    // 保留
} AVIMAINHEADER;
```

然 后，就是一个或多个'strl'子列表。(文件中有多少个流，这里就对应有多少个'strl'子列表。)每个'strl'子列表至少包含一个'strh' 块和一个'strf'块，而'strd'块(保存编解码器需要的一些配置信息)和'strn'块(保存流的名字)是可选的。首先是'strh'块，用于说 明这个流的头信息，可以使用一个 AVISTREAMHEADER 数据结构来操作：

```
typedef struct _avistreamheader {
 FOURCC fcc;  // 必须为 'strh'
 DWORD  cb;   // 本数据结构的大小，不包括最初的 8 个字节（fcc 和 cb 两个域）
FOURCC fccType;    // 流的类型：'auds'（音频流）、'vids'（视频流）、
                   //'mids'（MIDI 流）、'txts'（文字流）
 FOURCC fccHandler; // 指定流的处理者，对于音视频来说就是解码器
 DWORD  dwFlags;    // 标记：是否允许这个流输出？调色板是否变化？
 WORD   wPriority;  // 流的优先级（当有多个相同类型的流时优先级最高的为默认流）
 WORD   wLanguage;
 DWORD  dwInitialFrames; // 为交互格式指定初始帧数
 DWORD  dwScale;   // 这个流使用的时间尺度
 DWORD  dwRate;
```

```
    DWORD  dwStart;   // 流的开始时间
    DWORD  dwLength;  // 流的长度（单位与 dwScale 和 dwRate 的定义有关）
    DWORD  dwSuggestedBufferSize; // 读取这个流数据建议使用的缓存大小
    DWORD  dwQuality;    // 流数据的质量指标（0 ~ 10,000）
    DWORD  dwSampleSize; // Sample 的大小
    struct {
        short int left;
        short int top;
        short int right;
        short int bottom;
}  rcFrame;  // 指定这个流（视频流或文字流）在视频主窗口中的显示位置
             // 视频主窗口由 AVIMAINHEADER 结构中的 dwWidth 和 dwHeight 决定
} AVISTREAMHEADER;
```

然后是‘strf’块，用于说明流的具体格式。如果是视频流，则使用一个 BITMAPINFO 数据结构来描述；如果是音频流，则使用一个 WAVEFORMATEX 数据结构来描述。

当 AVI 文件中的所有流都使用一个‘strl’子列表说明了以后（注意：‘strl’子列表出现的顺序与媒体流的编号是对应的，比如第一个‘strl’子列表说明 的是第一个流（Stream 0），第二个‘strl’子列表说明的是第二个流（Stream 1），以此类推），‘hdrl’列表的任务也就完成了，随后跟着的就是 AVI 文件必需的第二个列表——‘movi’列表，用于保存真正的媒体流数据（视频 图像帧数据或音频采样数据等）。那么，怎么来组织这些数据呢？可以将数据块直接嵌在‘movi’列表里面，也可以将几个数据块分组成一个‘rec ’列表后再编排进‘movi’列表。（注意：在读取 AVI 文件内容时，建议将一个‘rec ’列表中的所有数据块一次性读出。）但是，当 AVI 文件中包含有多个流的时候，数据块与数据块之间如何来区别呢？于是数据块使用了一个四字符码来表征它的 类型，这个四字符码由 2 个字节的类型码和 2 个字节的流编号组成。标准的类型码定义如下：‘db’（非压缩视频帧）、‘dc’（压缩视频帧）、‘pc’（改 用新的调色板）、‘wb’（音缩视频）。比如第一个流（Stream 0）是音频，则表征音频数据块的四字符码为‘00wb’；第二个流（Stream 1）是视频，则表征视频数据块的四字符码为‘00db’或‘00dc’。对于视频数据来说，在 AVI 数据序列中间还可以定义一个新的调色板，每个改变的调 色板数据块用‘xxpc’来表征，新的调色板使用一个数据结构 AVIPALCHANGE 来定义。（注意：如果一个流的调色办中途可能改变，则应在这个流格 式的描述中，也就是 AVISTREAMHEADER 结构的 dwFlags 中包含一个 AVISF_VIDEO_PALCHANGES 标记。）另外，文字流数 据块可以使用随意的类型码表征。

最 后，紧跟在‘hdrl’列表和‘movi’列表之后的，就是 AVI 文件可选的索引块。这个索引块为 AVI 文件中每一个媒体数据块进行索引，并且记录它们在 文件中的偏移（可能相对于‘movi’列表，也可能相对于 AVI 文件开头）。索引块使用一个四字符码‘idx1’来表征，索引信息使用一个数据结构来 AVIOLDINDEX 定义。

```
 typedef struct _avioldindex {
  FOURCC  fcc;  // 必须为‘idx1’
  DWORD   cb;   // 本数据结构的大小，不包括最初的 8 个字节（fcc 和 cb 两个域）
  struct _avioldindex_entry {
    DWORD   dwChunkId;   // 表征本数据块的四字符码
    DWORD   dwFlags;     // 说明本数据块是不是关键帧、是不是‘rec ’列表等信息
    DWORD   dwOffset;    // 本数据块在文件中的偏移量
    DWORD   dwSize;      // 本数据块的大小
  } aIndex[]; // 这是一个数组！为每个媒体数据块都定义一个索引信息
} AVIOLDINDEX;
```

注意：如果一个 AVI 文件包含有索引块，则应在主 AVI 信息头的描述中，也就是 AVIMAINHEADER 结构的 dwFlags 中包含一个 AVIF_HASINDEX 标记。

还有一种特殊的数据块，用一个四字符码‘JUNK’来表征，它用于内部数据的队齐（填充），应用程序应该忽略这些数据块的实际意义。