离线

随笔 - 77, 文章 - 0, 评论 - 4, 阅读 - 15万

导航

博客园
首页
新随笔
联系
订阅 XML
管理

< 2025年5月 >

日	一	二	三	四	五	六
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

公告

昵称： 离线
园龄： 18年1个月
粉丝： 9
关注： 8
+加关注

搜索

找找看

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

mysql(7)
PLC(5)
MQTT(5)
RabbitMQ(3)
linux(3)
地图(3)
wpf(2)
mdns(1)
json(1)
Golang(1)
更多

随笔分类

C#(4)
Golang(1)
web(1)
其它(6)
数据库(21)
微服务(2)

随笔档案

2024年11月(1)
2023年11月(3)
2023年8月(2)

压缩包Zip格式详析（全网最详细）

原文：https://blog.csdn.net/qq_43278826/article/details/118436116

【前言】

Android的安装包.apk实际上就是个zip格式的压缩包，所以在了解apk签名之前，有必要先来探索一下zip格式压缩包的结构

一、Zip格式结构图总览

二、Zip文件结构详解

2023年6月(1)
2023年5月(1)
2023年4月(3)
2023年1月(1)
2022年2月(3)
2022年1月(2)
2021年11月(3)
2021年10月(2)
2021年9月(2)
2021年8月(1)
2021年7月(1)
2021年6月(4)
更多

阅读排行榜

1. MQTT客户端软件MQTT.fx下载使用使用详解(18554)
2. C#使用 MQTTnet 快速实现 MQTT 通信(18122)
3. mysql 存储及查询亿级数据(12648)
4. Sentinel-2 哨兵二号数据下载及处理教程 (9432)
5. 压缩包Zip格式详析（全网最详细）(9314)

评论排行榜

1. MyEasyClient简单使用(2)
2. .Net 微服务架构技术栈的那些事(1)
3. ZIGBEE抓包工具Ubiqua使用介绍(1)

推荐排行榜

1. 压缩包Zip格式详析（全网最详细）(3)
2. C#使用 MQTTnet 快速实现 MQTT 通信(3)
3. 用C#实现的几种常用数据校验方法整理（CRC校验；LRC校验；BCC校验；累加和校验）(1)
4. .Net 微服务架构技术栈的那些事(1)
5. C# TSC打印二维码和条形码(1)

最新评论

1. Re:.Net 微服务架构技术栈的那些事
总结的很到位，但是仍然不能解答我的疑惑，我最大的疑惑是c#的全栈工程师他在哪，我想找到他聊聊工作机会
--Alice妮妮
2. Re:MyEasyClient简单使用
@InsonLu
SuperSocket.ClientEngine.dll...
--离线
3. Re:ZIGBEE抓包工具Ubiqua使用介绍
断网，无法安装呀！
--SloanYYc
4. Re:MyEasyClient简单使用
未能找到类型或命名空间名“EasyClient<>”(是否缺少 using 指令或程序集引用?)
这个怎么处理，博主请回答。
--InsonLu

zip格式压缩包主要由三大部分组成：`数据区`、`中央目录记录区（也有叫核心目录记录）`、`中央目录记录尾部区`

1、数据区

数据区是由一系列`本地文件记录`组成，本地文件记录主要是记录了`压缩前后文件的元数据`以及`存放压缩后的文件`，组成部分也分为三大部分：`本地文件头`、`文件数据`、`文件描述`

1.1、本地文件头

Offset	Bytes	Description
0	4	文件头标识：0x04034b50
4	2	解压文件所需版本
6	2	通用位标记：Bit 0:加密标志，Bit 3:数据描述标志
8	2	压缩方式
10	2	文件最后修改时间
12	2	文件最后修改日期
14	4	CRC-32校验码
18	4	压缩后的大小
22	4	未压缩的大小
26	2	文件名长度
28	2	扩展区长度
30	n	文件名
30+n	m	扩展区

本地文件头主要是记录了`压缩文件的元数据`：

1）0~3：4个字节，用来存放本地文件头标识：`0x04034b50`，用于解压时候，读取判断文件头的开始

2）4~5：2个字节，记录解压缩文件所需的最低支持的ZIP规范版本，apk压缩版本默认是`20`，即`Deflate`压缩方式

> 当前最低功能版本定义如下：（`压缩包记录的解压版本*10，比如：2.0 * 10 = 20`）
> 1.0 – 默认值
> 1.1 – 文件是卷标
> 2.0 – 文件是一个文件夹（目录）
> 2.0 – 使用 Deflate 压缩来压缩文件
> 2.0 – 使用传统的 PKWARE 加密对文件进行加密
> 2.1 – 使用 Deflate64™ 压缩文件
> 2.5 – 使用 PKWARE DCL Implode 压缩文件
> 2.7 – 文件是补丁数据集
> 4.5 – 文件使用 ZIP64 格式扩展
> 4.6 – 使用 BZIP2 压缩文件压缩
> 5.0 – 文件使用 DES 加密
> 5.0 – 文件使用 3DES 加密
> 5.0 – 使用原始 RC2 加密对文件进行加密
> 5.0 – 使用 RC4 加密对文件进行加密
> 5.1 – 文件使用 AES 加密进行加密
> 5.1 – 使用更正的 RC2 加密对文件进行加密
> 5.2 – 使用更正的 RC2-64 加密对文件进行加密
> 6.1 – 使用非 OAEP 密钥包装对文件进行加密
> 6.2 – 中央目录加密

3）6~7：2个字节，记录通用标志位，`第0位为1`时（即二进制：00000000 00000001），表示`文件被加密`，解压时候需要解密；`第3位为1`时候（即二进制：

00000000 00000100），表示 `有数据描述部分` ，那么本地文件头中的 `CRC-32` 、`压缩大小` 和 `未压缩大小` 字段都被设置为0（虽然zip规范是这么定义，但是发现有些压缩包即使声明有数据描述部分，但是本地文件头的 `CRC-32` 、 `压缩大小` 和 `未压缩大小` 依然还是设置为真实值） ， 正确的值被放在紧跟在压缩数据之后的数据描述部分， `apk的通用标志位默认传0即可，也有传2048、2056` ，目前第15位是PKWARE保留位，没啥意义，更多通用标志位含义可见这里

4）8~9：2个字节，记录压缩包所用到的压缩方式， `apk默认Deflate压缩` ，传 `8` 即可, 要是传 `0` ，则是不压缩，各种压缩方式对应数值如下：

> 0 – The file is stored (no compression)
> 1 – The file is Shrunk
> 2 – The file is Reduced with compression factor 1
> 3 – The file is Reduced with compression factor 2
> 4 – The file is Reduced with compression factor 3
> 5 – The file is Reduced with compression factor 4
> 6 – The file is Imploded
> 7 – Reserved for Tokenizing compression algorithm
> 8 – The file is Deflated
> 9 – Enhanced Deflating using Deflate64™
> 10 – PKWARE Data Compression Library Imploding
> 11 – Reserved by PKWARE
> 12 – File is compressed using BZIP2 algorithm

5）10~11：2个字节，记录 `文件最后修改时间` ，是MS-DOS格式编码的时间

字段	Bits	Description
Date	0–4	Day of the month (1 - 31)
	5-8	Month (1 = January, 2 = February, etc.)
	9-15	Year offset from 1980 (add 1980 to get actual year)
Time	0-4	Second divided by 2
	5-10	Minute (0 - 59)
	11-15	Hour (0 - 23 on a 24-hour clock)

6）12~13：2个字节，记录 `文件最后修改日期` ，是MS-DOS格式编码的日期

7）14~17：4个字节，记录 `文件未压缩时的CRC-32校验码`

8）18~21：4个字节，记录 `文件压缩后的大小`
9）22~25：4个字节，记录 `文件未压缩的大小`
10）26~27：2个字节，记录 `文件名的长度` (假设文件名长度为n)
11）28~29：2个字节，记录 `扩展区的长度` (假设扩展区长度为m)
12）30~30+n: n个字节，记录 `文件名`
13）30+n~30+n+m: m个字节，记录 `扩展数据`

1.2、文件数据

文件数据紧跟在本地文件头之后，一般是 `压缩后的文件数据` 或 `压缩方式选择不压缩时候，用来存储未压缩文件数据` 。

1.3、文件描述

文件描述符仅在通用位标志的 `第 3 位被设置为1时` 才存在。 它是字节对齐的，紧跟在文件数据的最后一个字节之后。当且仅当无法在 .ZIP 文件中查找时才使用此描述符，例如：当输出 的.ZIP 文件是标准输出或不可查找设备时使用文件描述，换句话说，正常情况下都不需要使用

Offset	Bytes	Description
0	4	数据描述符标识：0x08074b50
16	4	CRC-32校验码
20	4	压缩后的大小
24	4	未压缩的大小

数据描述符标识不一定有，因为一开始规范是没有的，后面才加上去的

2、中央目录记录区（也称核心目录记录区）

中央目录记录区是有一系列 `中央目录记录` 所组成， `一条中央目录记录` 对应数据区中的 `一个压缩文件记录` ，中央目录记录由以下部分构成：

Offset	Bytes	Description
0	4	中央目录文件头标识：0x02014b50
4	2	压缩所用版本
6	2	解压所需版本
8	2	通用位标记
10	2	压缩方法
12	2	文件最后修改时间
14	2	文件最后修改日期
16	4	CRC-32校验码
20	4	压缩后的大小
24	4	未压缩的大小
28	2	文件名长度
30	2	扩展区长度
32	2	文件注释长度
34	2	文件开始位置的磁盘编号
36	2	内部文件属性
38	4	外部文件属性
42	4	本地文件头的相对位移
46	n	文件名
46+n	m	扩展区
46+n+m	k	文件注释

1）0~3：4个字节，记录核心目录文件头标识： `0x02014b50` ，用于解压时候，查找判断是否是中央目录的开始位置

2）4~5：2个字节， `记录压缩所用的版本` ，同数据区本地文件头的解压所需版本， `apk设置20`

3）6~7：2个字节， `记录解压所需的最小版本` ，同数据区本地文件头的解压所需版本， `apk设置20`

4) 8~9：2个字节， `通用位标记` ，同数据区本地文件头的通用位标记

5） `压缩方法` 、 `文件最后修改时间` 、 `文件最后修改日期` 、 `CRC-32校验码` 、 `压缩后大小` 、 `未压缩大小` 、 `文件名长度` 、 `扩展区长度` ，这几个字段的含义都等同于数据区本地文件头对应字段的含义

6）32~33：2个字节，记录 `文件注释的长度`

7）34~35：2个字节，记录 `文件开始位置的磁盘编号` ，一般传0即可

8） `内部文件属性` 、 `外部文件属性` ，一般也是传0即可

9）42~45：4个字节，记录 `数据区本地文件头相对于压缩包开始位置的偏移量`

3、中央目录记录尾部区

中央目录记录尾部主要作用是用来定位 `中央目录记录区的开始位置` ，同时 `记录压缩包的注释内容`

Offset	Bytes	Description
0	4	中央目录结束标记：0x06054b50
4	2	当前磁盘编号
6	2	中央目录开始位置的磁盘编号
8	2	该磁盘上所记录的中央目录数量
10	2	中央目录结构总数
12	4	中央目录的大小
16	4	中央目录开始位置相对位移
20	2	注释长度
22	n	注释内容

1）0~3：4个字节，中央目录记录尾部开头标记：`0x06054b50`，用于解压时，查找判断中央目录尾部的起始位置
2）4~5：2个字节，记录`中央目录记录尾部区所在磁盘编号`
3）6~7：2个字节，记录`中央目录开始位置所在的磁盘编号`
4）8~9：2个字节，`该磁盘上所记录的核心目录数量`
5）10~11：2个字节，`zip压缩包中的文件总数`
6）12~15：4个字节，`整个中央目录的大小（以字节为单位）`
7）16~19：4个字节，`中央目录开始位置相对位移`
8）20~21：2个字节，`注释内容的长度`（假设长度为n）
9）22~22+n：n个字节，`注释内容`

三、压缩包解压过程

1、先从`中央目录尾部区`着手，目标是找到中央目录尾部开头标记：`0x06054b50`，从上述对zip压缩包结构分析可知，中央目录尾部区除了注释内容之外，`固定大小占22个字节`，那么假如注释内容为空的时候，将指针从文件尾部往前移动22个字节，然后读取4个字节的数据，就正好是中央目录尾部开头标记：`0x06054b50`，但是注释内容是否为空在实际操作中是不可得知的，所以只能设置一个循环，每次递增一个字节，不断推测注释内容的长度，又因为`注释长度用2个字节`表示，那么`注释长度最大只能是65535个字节`，所以可以在0~65535这个范围内不断推测注释内容的长度.
下面是java代码实现的查找中央目录尾部开始位置的案例：

```
/** * 查找中央目录结尾的开始位置 * @param zipContents * @return */
  private static int findZipEndOfCentralDirectoryRecord(ByteBuffer zi
      //判断是否小端模式排列
      assertByteOrderLittleEndian(zipContents);

      int archiveSize = zipContents.capacity();
      //由于核心目录尾部大小至少是22个字节，小于22就是没意义的
      if (archiveSize < 22) {
          return -1;
      }

      //注释内容长度只可能是：【压缩包大小 - 核心目录尾部固定大小（22字节）】与
      int maxCommentLength = Math.min(archiveSize - 22, 65535);
      //假如没有注释内容，那么核心目录尾部开始位置是：压缩包大小 -22
      int eocdWithEmptyCommentStartPosition = archiveSize - 22;
      // 循环查找，假设没有注释内容到每次递增一个字节的注释内容，查找出：核心目录
      for (int expectedCommentLength = 0; expectedCommentLength < max

          int eocdStartPos = eocdWithEmptyCommentStartPosition - expe
          // 核心目录结尾标志：0x06054b50（十进制为：101010256），标志位长度
          // zipContents.getInt(eocdStartPos)，即从eocdStartPos位置开始
          if (zipContents.getInt(eocdStartPos) == 101010256) {

              //核心目录结尾标志的开始位置偏移20个字节就是注释内容长度，因为注
              int actualCommentLength = getUnsignedInt16(zipContents,
```

```
                // 要是从压缩包中读取到的注释长度跟循环查找计算出的注释长度一致，
                if (actualCommentLength == expectedCommentLength) {
                    return eocdStartPos;
                }
            }
        }

        return -1;
    }
    static void assertByteOrderLittleEndian(ByteBuffer buffer) {
        if (buffer.order() != ByteOrder.LITTLE_ENDIAN) {
            throw new IllegalArgumentException("ByteBuffer byte order n
        }
    }
```

2、中央目录尾部开始位置找到之后，那么可以从中获取到 `中央目录的开始位置` ，`中央目录的大小` ，以及 `中央目录记录条目总数`

3、接着，又可以从中央目录中读取到 `本地文件头相对压缩包开始位置的偏移量` ，那么就能读到本地文件记录，并从中解压出文件数据，大概的解压流程就到此结束啦

【扩展知识】

刚才在java举例中有涉及到一个 `小端排序` 问题，因为在Jvm中默认都是按 `大端模式` 储存, 而 .ZIP格式的数据是按 `小端模式` 编排的，所以需要手动对ByteBuffer中的数据进行 `小端排序` ，那么，什么是小端模式，什么是大端模式呢?

1、 `大端模式` ：Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端，大端模式是跟人读写习惯是一致的，比如：数字0x12345678 与0x11223344,大端模式表示如下：

```
低地址 ------------------------------------------------------> 高地址
0x12  |  0x34  |  0x56  |  0x78 | 0x11  |  0x22  |  0x33  |  0x44
```

2、 `小端模式` ：Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端，比如：数字0x12345678 与 0x11223344,小端模式表示如下：

```
低地址 ------------------------------------------------------> 高地址
0x78  |  0x56  |  0x34  |  0x12 | 0x44  |  0x33  |  0x22  |  0x11
```

那么，为啥会存在大小端不统一的问题呢?

> 这是因为计算机系统中内存是以 `字节` 为单位进行编址的，每个地址单元都唯一的对应着 `1个字节` （8 bit）。但是在C语言中除了8bit的char之外，还有16bit的short型，32bit的long型（要看具体的编译器），另外，对于位数大于8位的处理器，例如16位或者32位的处理器，由于寄存器宽度大于一个字节，那么必然存在着一个如果将多个字节安排的问题。因此就导致了大端存储模式和小端存储模式。我们常用的 `X86结构是小端模式` ，而KEIL C51则为大端模式。很多的ARM，DSP都为小端模式。有些ARM处理器还可以由硬件来选择是大端模式还是小端模式。TCP/IP协议隆重出场，RFC1700规定使用 `“大端”字节序为网络字节序`

既然大小端都有存在的必要性，那大小端模式各有啥优势呢?

小端模式优点：

内存的低地址处存放低字节，所以在强制转换数据时不需要调整字节的内容（注解：比如把int的4字节强制转换成short的2字节时，就直接把int数据存储的前两个字节给short就行，因为其前两个字节刚好就是最低的两个字节，符合转换逻辑）；

CPU做数值运算时从内存中依顺序依次从低位到高位取数据进行运算，直到最后刷新最高位的符号位，这样的运算方式会更高效

大端模式优点：

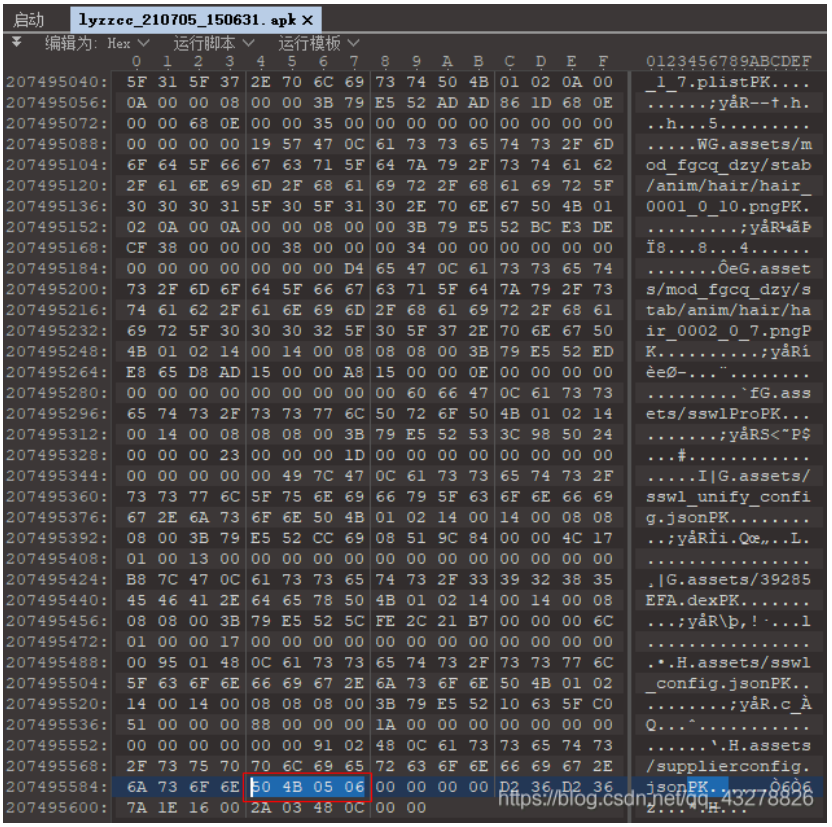
符号位在所表示的数据的内存的第一个字节中，便于快速判断数据的正负和大小

【注意】字符是只有1个字节，故对于字符不存在大小端模式之分，只有大于1个字节的才分大小端模式

【实践案例】

理论说了一大篇幅，想必各位看官已是头昏脑涨，咱们来动手分析一个压缩包看看，是否如咱们理论所言那般，下面是一个安卓安装包（.apk）的案例：

1、首先，先找到中央目录结尾标志：`0x06054b50`，因为zip格式是`小端模式`，那么，咱们看到的应该是：`50 4B 05 06`，用`010 Editor`打开apk，成功查找到`中央目录结尾标志`



从截图可以看到：

1）`当前磁盘编号`为：0x0000（即十进制：0）

2）`中央目录开始位置的磁盘编号`也是：0x0000（即十进制：0），

3）`该磁盘上所记录的中央目录数量`：0xD236(转为大端模式就是0x36D2，十进制：14034)

4）`zip压缩包中的文件总数`：0xD236(转为大端模式就是0x36D2，十进制：14034)

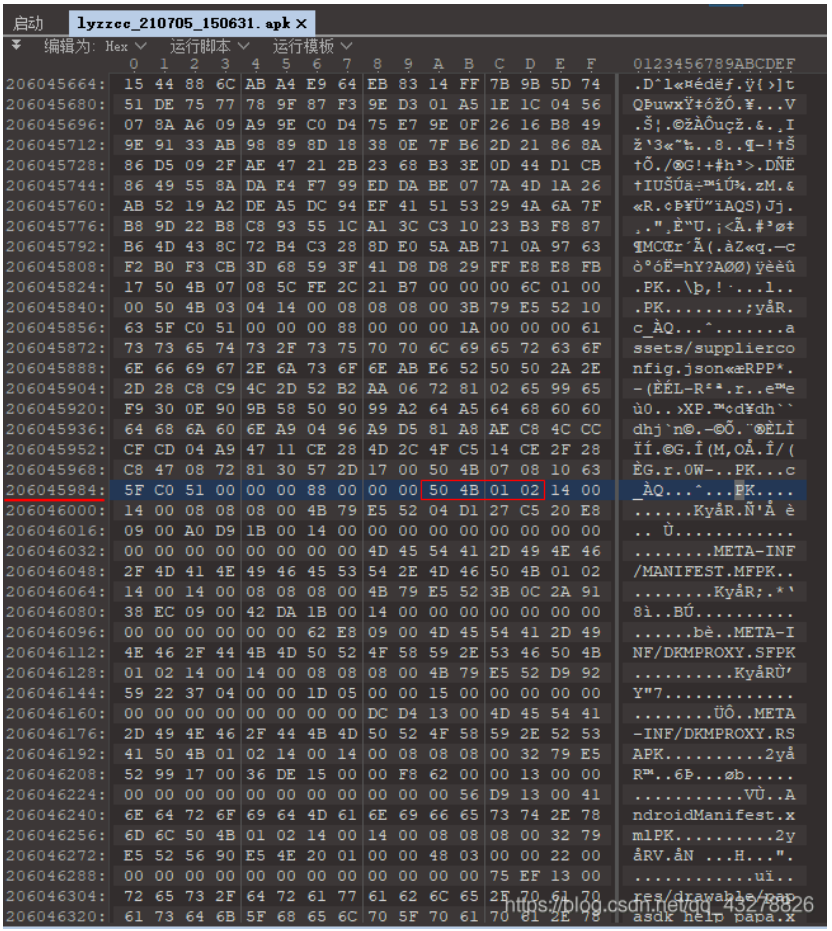
5）`中央目录大小`：0x7A1E1600(转为大端模式就是0x00161E7A，十进制：

1449594),

6) `中央目录开始位置的相对位移` ：0x2A03480C(转为大端模式就是0x0C48032A，十进制：206045994)

7) `注释长度` ：0x0000(即长度为0)

2、从第一步中，咱们可以知道 `中央目录开始位置` 是在地址 `206045994` ，那么查一下这个地址：



从截图可以看到

① 从地址 `206045994` 开始读取4个字节，得到 `0x504B0102` ，按大端模式排序为：`0x02014b50` ，刚好就是前面提到的 `中央目录文件头标识`

② `压缩所用版本` ：0x1400(转为大端模式就是0x0014，十进制：20)

③ `解压所需版本` ：0x1400(转为大端模式就是0x0014，十进制：20)

④ `通用位标记` ：0x0808(十进制：2056, 那么就是第3位设置为1,说明数据区有文件描述符)

⑤ `压缩方法` ：0x0800(转为大端模式就是0x0008，十进制：8)

⑥ `文件最后修改时间` ：0x4B79(转为大端模式就是0x794B，二进制：0111100101001011)

字段	Bits	Description
Date	0–4	Day of the month (1 - 31)
	5-8	Month (1 = January, 2 = February, etc.)
	9-15	Year offset from 1980 (add 1980 to get actual year)
Time	0-4	Second divided by 2
	5-10	Minute (0 - 59)
	11-15	Hour (0 - 23 on a 24-hour clock)

按照上面的MS-DOS时间编码规则，对二进制 `01111 001010 01011` 进行 拆分计算，时：01111（十进制：15），分：001010（十进制：10），秒：01011（十进制：11，这是秒除以2的值，故实际秒为11 * 2 = 22），那么，文件的最后修改时间为： `15:10:22`

⑦ *文件最后修改日期：0xE552(转为大端模式就是0x52E5，二进制：0101001 0111 00101)，年：0101001（十进制：41，1980 + 41 = 2021），月：0111（十进制：7），日：00101（十进制：5），那么文件的最后修改日期为：2021-7-5，比对一下跟压缩软件的结果是一致的*

⑧ CRC-32校验码：0x04D127C5(转为大端模式就是0xC527D104),跟上述压缩软件结果也是一致的

⑨ *压缩后的大小：0x20E80900(转为大端模式就是0x0009E820, 十进制：649248，约为634.03KB)*

⑩ *未压缩的大小：0xA0D91B00(转为大端模式就是0x001BD9A0, 十进制：1825184，约为1.74MB),跟上述压缩软件结果也是一致的*

⑪ *文件名长度：0x1400(转为大端模式就是0x0014，十进制：20)*

⑫ *扩展区长度*、*文件注释长度*、*文件开始位置的磁盘编号*、*内部文件属性*都是：*0x0000*

⑬ *外部文件属性*、*本地文件头的相对位移*都是：*0x00000000*

⑭ *文件名：0x4D 45 54 41 2D 49 4E 46 2F 4D 41 4E 49 46 45 53 54 2E 4D 46, 这些是字符ASCII码，转为字符是：META-INF/MANIFEST.MF*

【注意】假如文件名有中文的话，那这里存放UTF-8编码数据，中文一般先转换为Unicode编码字符，然后用UTF-8编码方式存储(Unicode只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储，UTF-8是unicode的一种实现方式，unicode实现方式还有UTF-16和UTF-32)

【UTF-8小知识】

UTF-8最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8的编码规则很简单，只有2条：

1 对于单字节的符号，字节的第1位（字节的最高位）设为0，后面7位为这个符号的unicode码。因此对于英语字母，UTF-8编码和ASCII码是相同的。

2 对于n字节的符号（n>1），第1个字节的前n位都设为1，第n+1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部为这个符号的unicode码

Unicode符号范围（十六进制）	UTF-8编码方式（二进制）
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

比如：已知“严”的unicode是4E25（100111000100101），根据上表，可以发现4E25处在第3行的范围内（0000 0800-0000 FFFF），因此“严”的UTF-8编码需要3个字节，即格式是“1110xxxx 10xxxxxx 10xxxxxx”。然后，从“严”的最后1个二进制位开始，依次从后向前填入格式中的x，多出的位补0。这样就得到了“严”的UTF-8编码是“11100100 10111000 10100101”，转换成十六进制就是E4B8A5

⑮ 因为扩展区长度为0，所以文件名后面紧跟压缩之后的文件数据，由上面分析的压缩长度为649248，所以后面649248个字节的数据都是文件数据

⑯ 因为是本地文件头的通用标志位第3位设置为1，所以存在数据描述区，数据描述区标识：0x504B0708(转换为大端模式：0x08074b50)

⑰ 数据描述符中的CRC-32校验码、压缩大小、未压缩大小跟本地文件头的值一致

离线
粉丝 - 9 关注 - 8
+加关注

« 上一篇： MQTT介绍
» 下一篇： C#设计模式 ---- 总结汇总（转载）

posted on 2023-11-10 14:43 离线 阅读(9323) 评论(0) 收藏 举报

刷新页面 返回顶部

相关博文：
· 转（zip文件格式说明）
· Arduino语法详解_含示例详解
· 压缩包信息
· 安卓逆向基础知识之apk文件结构 获取静态文件
· 常见文件格式----2023.2.16

阅读排行：
· C#开发的Panel滚动分页控件 - 开源研究系列文章
· ShadowSql之开源不易
· 如何反向绘制出 .NET程序 异步方法调用栈
· C#多线程编程精要：从用户线程到线程池的效能进化论
· 上周热点回顾（5.5-5.11）