png隐写数据进TEXTchunk

原创

舞动的痞老板 于 2020-06-06 22:32:47 发布 569 收藏

–> go to 总目录

文章目录

一、PNG描述
1.1 格式
1.2 rgba
二、隐写数据
2.1 写入数据&解析 text chunk
2.2 写水印

想要向 `png` 隐写数据，需求来源于想将图片上传于 `csdn` 并获得 `url`，实现将印象笔记中的 `.enex` 文件格式中的图片url替换成csdn提供的链接。
PNG文件格式解析
* 利用文件类型判断文件头

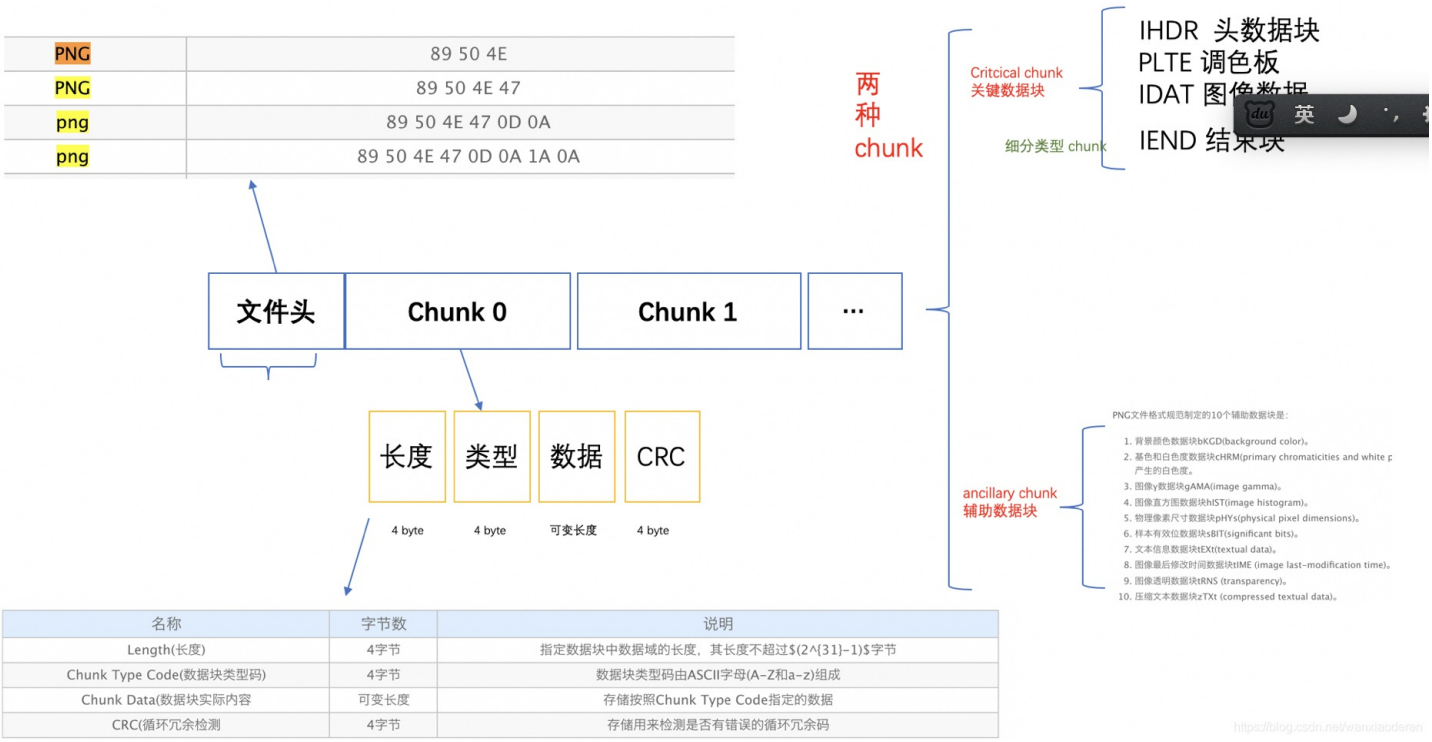
一、PNG描述

> 百度百科

- `体积小` 网络通讯中因受带宽制约，在保证图片清晰、逼真的前提下，网页中不可能大范围的使用文件较大的bmp格式文件。
- `无损压缩` PNG文件采用LZ77算法的派生算法进行压缩，其结果是获得高的压缩比，不损失数据。它利用特殊的编码方法标记重复出现的数据，因而对图像的颜色没有影响，也不可能产生颜色的损失，这样就可以重复保存而不降低图像质量。
- `索引彩色模式` PNG-8格式与GIF图像类似，同样采用8位调色板将RGB彩色图像转换为索引彩色图像。图像中保存的不再是各个像素的彩色信息，而是从图像中挑选出来的具有代表性的颜色编号，每一编号对应一种颜色，图像的数据量也因此减少，这对彩色图像的传播非常有利。
- `更优化的网络传输显示` PNG图像在浏览器上采用流式浏览，即使经过交错处理的图像会在完全下载之前提供浏览者一个基本的图像内容，然后再逐渐清晰起来。它允许连续读出和写入图像数据，这个特性很适合于在通信过程中显示和生成图像。
- `支持透明效果` PNG可以为原图像定义256个透明层次，使得彩色图像的边缘能与任何背景平滑地融合，从而彻底地消除锯齿边缘。这种功能是GIF和JPEG没有的。

1.1 格式

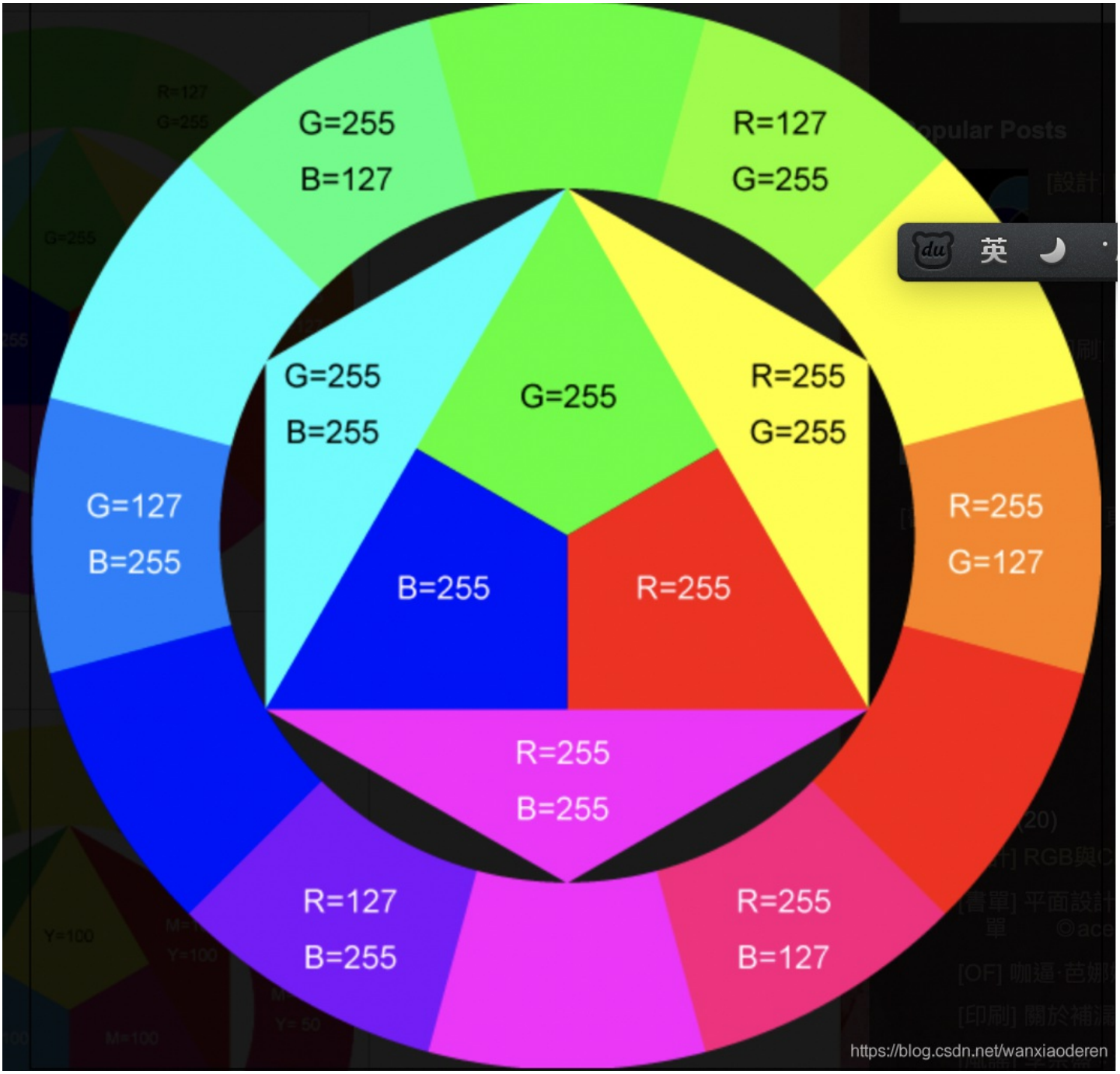
详细的各种辅助块的介绍见PNG文件格式解析



名称	字节数	说明
Length(长度)	4字节	指定数据块中数据域的长度，其长度不超过$(2^{31}-1)$字节
Chunk Type Code(数据块类型码)	4字节	数据块类型码由ASCII字母(A-Z和a-z)组成
Chunk Data(数据块实际内容	可变长度	存储按照Chunk Type Code指定的数据
CRC(循环冗余检测	4字节	存储用来检测是否有错误的循环冗余码

1.2 rgba

色相环（rgb之和加起来是255所形成的一个圆，比如R到G的线就是:B=255,R+G=255）



每一个像素由三原色和明暗表示:`Red`（红色）`Green`（绿色）`Blue`（蓝色）和`Alpha`(亮度)。
所以rgba就是这个`4个byte`组成，每个byte有255个值。
所以用8个16进制表示比如`#7FFF0000`

二、隐写数据

上面介绍的chunk中我们可以使用
`tEXt`加信息
png decoder
png_encoder

2.1 写入数据&解析 text chunk

三个类

- `PNG` 封装png格式信息
- `ImageInfo` 图片的信息:id=UUID,name=图片名称，path=图片路径
- `transformer类` 有两个核心方法

1 . `transorm(srcDir,targetDir)`，接受srcDir下所有png图片，写入targetDir中。写入的过程中，插入 `tExt` chunk — `ImageInfo` 的key和value。
具体为 `" imageInfo uuid_name "`

2. `praseInfoFrom(String srcDir)`，解析tExtCunk，被还原成 `ImageInfo`

png图片格式

```
package com.pngimage;

import java.util.UUID;

/**
 * <code>ImageInfo</code> description
 *
 * @author sunqiyuan
 * @date 2020-06-05
 */
public class ImageInfo {
    private String id;
    private String name;
    private String path;

    public ImageInfo(String name, String path) {
        this.id = UUID.randomUUID().toString().replace("-", "");
        this.name = name;
        this.path = path;

    }

    public ImageInfo(String id, String name, String path) {
        this.id = id;
        this.name = name;
        this.path = path;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    @Override
    public String toString() {
        return "ImageInfo{" +
                "id='" + id + '\'' +
                ", name='" + name + '\'' +
                ", path='" + path + '\'' +
                '}';
    }
}
```

```
package com.pngimage;

import java.io.IOException;
import java.util.zip.CRC32;

import sun.security.krb5.internal.crypto.crc32;

/**
 * <code>PNG</code> PNG模型信息
 *
 * @author sunqiyuan
 * @date 2020-06-05
 */
public class PNG {
    final byte[] identifier = {-119, 80, 78, 71, 13, 10, 26, 10};
    private static final int bKGDChunk = 0x624B4744;
    private static final int cHRMChunk = 0x6348524D;
    private static final int gAMAChunk = 0x67414D41;
    private static final int hISTChunk = 0x68495354;
    private static final int IDATChunk = 0x49444154;
    public static final int IENDChunk = 0x49454E44;
    private static final int IHDRChunk = 0x49484452;
    private static final int PLTEChunk = 0x504C5445;
    private static final int pHYsChunk = 0x70485973;
    private static final int sBITChunk = 0x73424954;
    public static final int tEXtChunk = 0x74455874;
    private static final int tIMEChunk = 0x74494D45;
    private static final int tRNSChunk = 0x74524E53;
    private static final int zTXtChunk = 0x7A545874;

    public int pos;

    /**
     * png的chunk格式
     */
    public static class Chunk {
```

```
    /**
     * 总长度
     * 4 个字节
     */
    int length;
    /**
     * 4 个字节 标识chunk类型
     */
    int type;
    /**
     * 可变长度，chunk块格式长度，属于每个chunk的长度
     */
    byte[] chunkData;

    /**
     * 4个字节的冗余校验码
     */
    int crc;

    public Chunk(int length, int type, byte[] chunkData, int pos) {
        this.length = length;
        this.type = type;
        this.chunkData = chunkData;
        this.crc = readInt(chunkData, pos + length - 4);
    }

    public Chunk(byte[] chunk, int pos) {
        this.length = readInt(chunk, pos);
        this.type = readInt(chunk, pos + 4);
    }

    public Chunk() {
    }
}

/**
 * IHDR_Chunk
 */
class IHDR_Chunk extends Chunk {
    int width = 0;
    int height = 0;
    int bitdepth = 0;
    int colortype = 0;
    int compressionmethod = 0;
    int filtermethod = 0;
    int interlacemethod = 0;

    public IHDR_Chunk(int length, int type, byte[] chunkData, int pos) {
        super(13, IHDRChunk, chunkData, pos);
        width = readInt(chunkData, pos);
        pos += 4;
        height = readInt(chunkData, pos);
        pos += 4;
        bitdepth = chunkData[pos++];
        colortype = chunkData[pos++];
        compressionmethod = chunkData[pos++];
        filtermethod = chunkData[pos++];
        interlacemethod = chunkData[pos++];
```

```
    }
}

/**
 * IHDR_Chunk
 */
class IDAT_Chunk extends Chunk {

    public IDAT_Chunk(int length, int type, byte[] chunkData, int pos) {
        super(length, IDATChunk, chunkData, pos);
    }
}

/**
 * IHDR_Chunk
 */
static class TEXT_Chunk extends Chunk {
    String key;
    String value;

    public TEXT_Chunk(String key, String value) {
        super();
        this.key = key;
        this.value = value;
    }

    public TEXT_Chunk(int length, int type, byte[] chunkData, int pos) {
        super(length, tEXtChunk, chunkData, pos);
        int sep = 0;
        for (int i = pos; i < pos + length - 4; i++) {
            if (chunkData[i] == 0) {
                sep = i;
                break;
            }
        }
        key = new String(chunkData, pos, sep - pos);
        value = new String(chunkData, sep+1 ,  length - (sep - pos)-1);
    }

    public byte[] tochunk() {
        int length = key.getBytes().length + 1 + value.getBytes().length;
        byte[] chunk = new byte[length + 8 + 4];
        int pos = 0;
        chunk[pos++] = (byte) (length >> 24);
        chunk[pos++] = (byte) (length >> 16);
        chunk[pos++] = (byte) (length >> 8);
        chunk[pos++] = (byte) (length);
        chunk[pos++] = (byte) (tEXtChunk >> 24);
        chunk[pos++] = (byte) (tEXtChunk >> 16);
        chunk[pos++] = (byte) (tEXtChunk >> 8);
        chunk[pos++] = (byte) (tEXtChunk);
        System.arraycopy(key.getBytes(), 0, chunk, pos, key.length());
        pos += key.length();
        chunk[pos++] = 0;
        System.arraycopy(value.getBytes(), 0, chunk, pos, value.length());
        pos += value.length();
        CRC32 crc32 = new CRC32();
        crc32.update(chunk, 0, pos);
        long c = crc32.getValue();
        chunk[pos++] = (byte) (c >> 24);
```

```
            chunk[pos++] = (byte) (c >> 24);
            chunk[pos++] = (byte) (c >> 16);
            chunk[pos++] = (byte) (c >> 8);
            chunk[pos++] = (byte) (c);
            return chunk;
        }
    }

    /**
     * IHDR_Chunk
     */
    static class IEND_Chunk extends Chunk {

        public IEND_Chunk(int length, int type, byte[] chunkData, int pos) {
            super(length, IENDChunk, chunkData, pos);
        }
    }

    static int readInt(byte[] data, int pos) {
        return (((data[pos++] & 0xff) << 24) +
                        ((data[pos++] & 0xff) << 16) +
                        ((data[pos++] & 0xff) << 8) +
                        ((data[pos++] & 0xff)));
    }

    public boolean isPng(byte[] data, int pos) {
        return compare(data, pos, identifier.length, identifier, 0, identifier.length);
    }

    boolean compare(byte[] b1, int pos1, int length1, byte[] b2, int pos2, int length2) {
        if (length1 != length2) {
            return (false);
        }
        for (int i = 0; i < length1; i++) {
            if (b1[pos1++] != b2[pos2++]) {
                return (false);
            }
        }
        return (true);
    }

    void checkEquality(byte[] b1, int pos1, int length1, byte[] b2, int pos2, int length2) {
        if (!compare(b1, pos1, length1, b2, pos2, length2)) {
            throw (new RuntimeException("Format error"));
        }
    }

    public static void main(String[] args) {
        PNG.TEXT_Chunk text = new PNG.TEXT_Chunk("imageInfo",
                "sunqiyuan" + "__" + "sunqiyuanname");
        byte[] insert = text.tochunk();
        System.out.println("pp");

    }
}
```

```
package com.pngimage;

import java.io.IOException;
```

```java
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Stream;

/**
 * <code>Convter</code> description
 * 给图片加信息
 *
 * @author sunqiyuan
 * @date 2020-06-05
 */

public class Transformer {

    /**
     * attach the imageInfo to BufferedImage
     *
     * @param imageInfo the original png info
     * @return
     */
    public void attach(ImageInfo imageInfo, String target) throws Exception {
        Path imagePath = Paths.get(imageInfo.getPath());
        if (!Files.exists(imagePath)) {
            System.out.println(imageInfo.getPath() + "  not exist!!");
            return;
        }

        Path tp = Paths.get(target, imageInfo.getName());

        if (!Files.exists(Paths.get(target))) {
            Files.createDirectory(Paths.get(target));
        }
        if (!Files.exists(tp)) {
            Files.createFile(tp);
        }
        try (InputStream in = Files.newInputStream(imagePath);

             OutputStream out = Files.newOutputStream(tp);
        ) {
            int pos = 0;
            int size = 4096;
            int num = 0;
            boolean chunkchanged = true;
            int chunklength = 0;
            PNG.Chunk chunk = null;
            byte[] inbuf = new byte[size];

            boolean pngStart = true;

            while ((num = in.read(inbuf, pos, size - pos)) > 0) {
                int limit = pos + num;
                pos = 0;
                PNG png = new PNG();
```

```
// 首次读取时判断是不是png
if (pngStart && !png.isPng(inbuf, pos)) {
    break;
}
if (pngStart) {
    pngStart = false;
    pos += 8;
}

// 找IEND trunk
while (true) {
    if (chunkchanged) {
        // 能不能满足基本的读取chunk的长度和类型，不能的话先刷新
        if (limit - pos <= 8) {
            out.write(inbuf, 0, pos);
            System.arraycopy(inbuf, pos, inbuf, 0, limit - pos);
            pos = limit - pos;
            break;
        }
        // 读取chunk数据
        try {
            chunk = new PNG.Chunk(inbuf, pos);
            chunkchanged = false;
            chunklength = 8 + chunk.length + 4;
            // 根据chunk对应操作
            if (chunk.type == PNG.IENDChunk) {
                out.write(inbuf, 0, pos);
                PNG.TEXT_Chunk text = new PNG.TEXT_Chunk("imageInfo",
                        imageInfo.getId() + "__" + imageInfo.getName());
                byte[] insert = text.tochunk();
                out.write(insert);
                out.write(inbuf, pos, limit - pos);
                break;
            }

        } catch (Exception e) {
            throw e;
        }
    }

    if (limit - pos < chunklength) {
        //chunk 很大，甚至一个inbuf放不下，这时pos还是等于0，就跳过copy
        if (pos != 0) {
            out.write(inbuf, 0, pos);
            System.arraycopy(inbuf, pos, inbuf, 0, limit - pos);
            pos = limit - pos;
        } else {
            out.write(inbuf, 0, limit);
            chunklength -= limit;
        }
        break;
    } else {
        // 开始下一个chunk
        chunkchanged = true;
        pos += chunklength;

    }

}
```

```
        }

    }

}

/**
 * prase the imageInfo from image
 *
 * @param src the original png info
 * @return
 */
public ImageInfo prase(String src) throws Exception {
    ImageInfo result = null;

    Path imagePath = Paths.get(src);
    if (!Files.exists(imagePath)) {
        System.out.println(src + "  not exist!!");
        return result;
    }

    if (!Files.exists(Paths.get(src))) {
        return null;
    }
    try (InputStream in = Files.newInputStream(imagePath);
    ) {
        int pos = 0;
        int size = 4096;
        int num = 0;
        boolean chunkchanged = true;
        int chunklength = 0;
        PNG.Chunk chunk = null;
        byte[] inbuf = new byte[size];

        boolean pngStart = true;

        while ((num = in.read(inbuf, pos, size - pos)) > 0) {
            int limit = pos + num;
            pos = 0;
            PNG png = new PNG();
            // 首次读取时判断是不是png
            if (pngStart && !png.isPng(inbuf, pos)) {
                break;
            }
            if (pngStart) {
                pngStart = false;
                pos += 8;
            }

            // 找IEND trunk
            while (true) {
                if (chunkchanged) {
                    // 能不能满足基本的读取chunk的长度和类型，不能的话先刷新
                    if (limit - pos <= 8) {
                        System.arraycopy(inbuf, pos, inbuf, 0, limit - pos);
                        pos = limit - pos;
                        break;
                    }
                    // 读取chunk数据
```

```
                    try {
                        chunk = new PNG.Chunk(inbuf, pos);
                        chunkchanged = false;
                        chunklength = 8 + chunk.length + 4;
                        // 根据chunk对应操作
                        if (chunk.type == PNG.IENDChunk) {

                            break;
                        }
                        if (chunk.type == PNG.tEXtChunk) {

                            PNG.TEXT_Chunk newc = new PNG.TEXT_Chunk(chunk.length, PNG.tEXtChunk, inbuf, pos
+ 8);

                            if (newc.key.equals("imageInfo")) {
                                String[] id_name = newc.value.split("__");
                                if (id_name.length >= 2) {
                                    result = new ImageInfo(id_name[0], id_name[1], src);

                                }

                            }

                        }

                    } catch (Exception e) {
                        throw e;
                    }
                }

                if (limit - pos < chunklength) {
                    //chunk 很大，甚至一个inbuf放不下，这时pos还是等于0，就跳过copy
                    if (pos != 0) {
                        System.arraycopy(inbuf, pos, inbuf, 0, limit - pos);
                        pos = limit - pos;
                    } else {
                        chunklength -= limit;
                    }
                    break;
                } else {
                    // 开始下一个chunk
                    chunkchanged = true;
                    pos += chunklength;

                }

            }

        }

    }
    return result;
}

/**
 * accept the pngs of src dir and write to target dir
 *
 * @return
 */
public int transform(String srcDir, String targetDir) throws Exception {
    Path src = Paths.get(srcDir);
```

```
        ArrayList<ImageInfo> imageInfos = new ArrayList<>();

        try (Stream<Path> stream = Files.list(src)) {
            Iterator<Path> ite = stream.iterator();
            while (ite.hasNext()) {
                Path pp = ite.next();
                if (!pp.getFileName().toString().endsWith("png")) {
                    continue;
                }
                ImageInfo i = new ImageInfo(pp.getFileName().toString(), pp.toAbsolutePath().toString());
                imageInfos.add(i);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        for (ImageInfo im : imageInfos) {
            System.out.println(im.getName());
            attach(im, targetDir);
        }
        return imageInfos.size();
    }

    /**
     * praseInfoFrom the pngs of src dir and write to target dir
     *
     * @return
     */
    public List<ImageInfo> praseInfoFrom(String srcDir) throws Exception {
        Path src = Paths.get(srcDir);
        ArrayList<ImageInfo> imageInfos = new ArrayList<>();

        try (Stream<Path> stream = Files.list(src)) {
            Iterator<Path> ite = stream.iterator();
            while (ite.hasNext()) {
                Path pp = ite.next();
                if (!pp.getFileName().toString().endsWith("png")) {
                    continue;
                }
                ImageInfo i = new ImageInfo(pp.getFileName().toString(), pp.toAbsolutePath().toString());
                imageInfos.add(i);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        for (ImageInfo im : imageInfos) {
            System.out.println(im.getName());
            ImageInfo temp = prase(im.getPath());
            System.out.println(temp.getId() + " _ " + temp.getName());
        }
        return imageInfos;
    }

    public static void main(String[] args) throws Exception {
        Transformer transformer = new Transformer();
        transformer.transform("/Users/xxxx/Desktop/我的笔记/b bytebuddy 进阶.resources", "/Users/xxxx/Desktop"
                + "/mynotes");
        List<ImageInfo> imageInfos = transformer.praseInfoFrom("/Users/xxxx/Desktop/mynotes");
```

```
    }
}
```

2.2 写水印

插入隐形图像

原理是对指定位置的像素的R通道降低颜色，在单通道时颜色就会显现出来。

平常看不出来。