二维码

二维码生成可以又前端和后端生成。

前端生成，比较容易。只需引入QRCode.js，然后指定一个元素dom和携带信息info，然后将两个元素作为参数new QRCode（dom，info）即可

详细请看菜鸟教程

<http://www.runoob.com/w3cnote/javascript-qrcodejs-library.html>

然而此种前端的方式将二维码的携带信息暴露在了前端，只需要用浏览器调试工具一看就可知道二维码的携带信息，殊不安全。所以我们介绍第二种后端的方式。

首先需要引入Qrcode.jar(需要用到这个类com.swetake.util.Qrcode;)

然后将生成二维码进行封装

主要过程

1. 创建一个图片缓冲区（bufferedImage），
2. 然后将图片画到图片缓冲区中，在这一步中，又分为
   1. 获取图片缓冲区的画笔，
   2. 设置画笔属性
   3. 画一个矩形
   4. 创建Qrcode对象，调用Qrcode.calQrcode()将二维码携带信息info转成

Byte的二维数组

* 1. 将二维数组依次绘制到图片缓冲区

1. 刷新图片缓冲区

上代码

package com.util;

import com.swetake.util.Qrcode;

import javax.imageio.ImageIO;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

public class CreateQRCode {

/\*\*

\* 每个元素的大小

\*/

private int scale;

/\*\*

\* 二维码与边缘的间隔

\*/

private int gap;

/\*\*

\* 版本号 取值1-40

\*/

private int version;

/\*\*

\* 整个二维码宽 可根据scale和gap来计算

\*/

private int width;

/\*\*

\* 整个二维码长 可根据scale和gap来计算

\*/

private int height;

/\*\*

\* 该值参与长和宽的自动计算。QRCode会将信息字符串转成byte类型的二维数组，数组长度为41

\*/

private final static int CONST41 = 41 ;

private final static int DEFAULT\_VERSION= 6 ;

private final static int DEFAULT\_SCALE= 3 ;

private static final int DEFAULT\_GAP = 3 ;

public CreateQRCode() {

this(DEFAULT\_SCALE,DEFAULT\_GAP);

}

public CreateQRCode(int scale, int gap) {

this(scale,gap,DEFAULT\_VERSION);

}

public CreateQRCode(int scale, int gap, int version) {

this(scale,gap,version,scale\*CONST41+2\*gap,scale\*CONST41+2\*gap);

}

public CreateQRCode(int scale, int gap, int version, int width, int height) {

this.scale = scale;

this.gap = gap;

this.version = version;

this.width = width;

this.height = height;

}

/\*\*

\* 生成二维码

\* @param info 二维码携带的信息

\* @return BufferedImage

\*/

public BufferedImage newInstance(String info) {

//缓冲区

BufferedImage bufferedImage = new BufferedImage(this.width, this.height, BufferedImage.TYPE\_INT\_BGR);

//绘图

draw(info,bufferedImage);

//刷新

bufferedImage.flush();

return bufferedImage;

}

/\*\*

\* @param error 纠错等级分

\* level L : 最大 7% 的错误能够被纠正；

\* level M : 最大 15% 的错误能够被纠正；

\* level Q : 最大 25% 的错误能够被纠正；

\* level H : 最大 30% 的错误能够被纠正；

\* @param encodeMode 注意版本信息 N代表数字 、A代表 a-z,A-Z、B代表 其他)

\* @param version 版本号 1-40

\* @return

\*/

public Qrcode createQrcCode(char error,char encodeMode,int version){

Qrcode x = new Qrcode();

x.setQrcodeErrorCorrect(error);

x.setQrcodeEncodeMode(encodeMode);//注意版本信息 N代表数字 、A代表 a-z,A-Z、B代表 其他)

x.setQrcodeVersion(version);//版本号 1-40

return x;

}

/\*\*

\* 根据图片缓冲区生成其Graphics2D对象，设置相关属性

\* @param bufferedImage

\* @return

\*/

public Graphics2D createGraphics2D(BufferedImage bufferedImage){

Graphics2D gs = bufferedImage.createGraphics();

gs.setBackground(Color.WHITE);

gs.setColor(Color.BLACK);

return gs;

}

/\*\*

\*

\* @param info 二维码携带的信息

\* @param bufferedImage 二维码的缓冲区

\* @throws UnsupportedEncodingException

\*/

public void draw(String info,BufferedImage bufferedImage) {

//获取Qrcode对象，主要功能将字符串转为byte的二进制数组，便于写入图片生成二维码

Qrcode qrcCode = createQrcCode('L','B',this.version);

//获取画笔

Graphics2D gs =createGraphics2D(bufferedImage);

//绘制矩形

gs.clearRect(0, 0,this.width,this.height);

//汉字转格式需要抛出异常

byte[] d = new byte[0];

try {

d = info.getBytes("utf-8");

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

/\*\*

\* 容易踩坑的地方

\* 1.注意for循环里面的i，j的顺序，

\* s[j][i]二维数组的j，i的顺序要与这个方法中的 gs.fillRect(j\*3+pixoff,i\*3+pixoff, 3, 3);

\* 顺序匹配，否则会出现解析图片是一串数字

\* 2.注意此判断if (d.length > 0 && d.length < 120)

\* 是否会引起字符串长度大于120导致生成代码不执行，二维码空白

\* 根据自己的字符串大小来设置此配置

\*/

if (d.length > 0 && d.length < 120) {

boolean[][] s = qrcCode.calQrcode(d);

for (int i = 0; i < s.length; i++) {

for (int j = 0; j < s.length; j++) {

if (s[j][i]) {

gs.fillRect(j \* this.scale + this.gap, i \* this.scale + this.gap, this.scale, this.scale);

}

}

}

}

gs.dispose();

}

//省略getter and setter

/\*\*

\* 这里是测试

\* @param args

\* @throws IOException

\*/

public static void main(String[] args) throws IOException {

BufferedImage bufferedImage = new CreateQRCode().newImage("http://www.baidu.com?a=1");

try {

ImageIO.write(bufferedImage, "JPG", new File("D:/qrcode.JPG"));

} catch (IOException e) {

e.printStackTrace();

}

}

}

封装到这一步，调用CreateQRCode().newImage("http://www.baidu.com?a=1")

就已经在内存中生成了二维码的图片

ImageIO.write(bufferedImage, "JPG", new File("D:/qrcode.JPG"));这行代码可以将二维码保存到本地硬盘。

那接下来就是将二维码展现在前端页面。

后台部分分为三部分

1. response参数，主要是将浏览器预计接受的数据类型改为图片

response.setContentType("image/jpeg");

1. 然后调用我们封装的对象，生成二维码的缓冲对象。

BufferedImage bi = new CreateQRCode().newImage("http://www.baidu.com?a=1");

1. 获取response的输出流，将bi写入。

ServletOutputStream out = response.getOutputStream();

ImageIO.write(bi, "jpeg", out);

上代码springmvc

@Controller

@RequestMapping("common")

public class CommonController {

@RequestMapping("verity")

private void verity ( HttpServletResponse response)

throws ServletException, IOException

{

response.setHeader("Pragma", "no-cache");

response.setHeader("Cache-Control", "no-cache");

response.setDateHeader("Expires", 0);

**response.setContentType("image/jpeg");**

BufferedImage bi = new CreateQRCode().newImage("http://www.baidu.com?a=1");

// 转成JPEG格式

ServletOutputStream out = response.getOutputStream();

@Controller

@RequestMapping("common")

public class CommonController {

@RequestMapping("verity")

private void verity ( HttpServletResponse response)

throws ServletException, IOException

{

response.setHeader("Pragma", "no-cache");

response.setHeader("Cache-Control", "no-cache");

response.setDateHeader("Expires", 0);

response.setContentType("image/jpeg");

BufferedImage bi = new CreateQRCode().newImage("http://www.baidu.com?a=1");

// 转成JPEG格式

ServletOutputStream out = response.getOutputStream();

ImageIO.write(bi, "jpeg", out);

out.close();

}

}

前端部分就比较简单了，写个img标签，src填上刚才那个请求即可。

**上代码**

<div>

二维码 <img src="common/verity" >

</div>

启动项目 你就可以看到二维码了。

另外

另外，Qrcode的依赖有好几个，但是都不是我们需要的，我们需要的对象全路径名是这个com.swetake.util.Qrcode。

[https://www.cnblogs.com/bigroc/p/7496995.html 这篇博客上提供了该jar](https://www.cnblogs.com/bigroc/p/7496995.html%20这篇博客上提供了该jar)包的下载地址

另外下载下来是jar包，，如果是maven项目该怎么办呢

附上方法

mvn install:install-file -Dfile=D:\\Qrcode.jar -DgroupId=QRCode -DartifactId=QRCode -Dversion=3.0 -Dpackaging=jar -DgeneratePom=true

说明

1将 jar包放到D盘，2maven的环境变量必须得配置

3引入

<dependency>

<groupId>QRCode</groupId>

<artifactId>QRCode</artifactId>

<version>3.0</version>

</dependency>

后来又看到使用google的zxing生成二维码的，这个依赖对二维码的封装度要高一些，不用自己封装一些东西，使用方便一些。

<https://blog.csdn.net/liuxuli232/article/details/78780385>

推荐阅读 二维码的生成细节和原理

https://coolshell.cn/articles/10590.html