文档修订说明

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订日期** | **修订说明** | **修订人** |
| V1.0.0 | 2018/12/12 | 创建文档 | 王海 |
| V2.0.0 | 2018/12/19 | 添加了QR码的种类 章节 | 王海 |
| V2.1.0 | 2018/12/23 | 添加了添加格式信息 章节 | 王海 |
| V2.2.0 | 2018/12/26 | 更新了 数据分析 章节 | 王海 |
|  |  |  |  |
|  |  |  |  |

目录

[1. 什么是二维码 4](#_Toc533612947)

[1.1. 二维码的结构 4](#_Toc533612948)

[1.1.1. 堆叠式/行排式 4](#_Toc533612949)

[1.1.2. 矩阵式二维码 4](#_Toc533612950)

[2. 什么是QR Code？ 5](#_Toc533612951)

[2.1. 特点 5](#_Toc533612952)

[2.1.1. 信息容量大 5](#_Toc533612953)

[2.1.2. 占用空间小 5](#_Toc533612954)

[2.1.3. 存储类型多 6](#_Toc533612955)

[2.1.4. 容错能力强 6](#_Toc533612956)

[2.1.5. 全方位识别 7](#_Toc533612957)

[2.1.6. 支持数据合并功能 7](#_Toc533612958)

[2.1.7. 结构连接(可选) 8](#_Toc533612959)

[2.1.8. 扩充解释(可选) 8](#_Toc533612960)

[2.2. QR Code版本 9](#_Toc533612961)

[2.3. 可编码字符集 11](#_Toc533612962)

[2.4. QR Code标准 11](#_Toc533612963)

[2.4.1. QR码的标准化历程 11](#_Toc533612964)

[2.4.2. 如何获取QR码的标准文件 11](#_Toc533612965)

[3. QR Code种类 12](#_Toc533612966)

[3.1. QR码 12](#_Toc533612967)

[3.1.1. 模型1 12](#_Toc533612968)

[3.1.2. 模型2 12](#_Toc533612969)

[3.2. Micro QR码 12](#_Toc533612970)

[3.2.1. 只有一个定位图案 12](#_Toc533612971)

[3.2.2. 占用空间小 12](#_Toc533612972)

[3.2.3. 数据容量小 12](#_Toc533612973)

[3.2.4. 已经标准化 12](#_Toc533612974)

[3.3. iQR码 13](#_Toc533612975)

[3.3.1. 容量大尺寸小 13](#_Toc533612976)

[3.3.2. 长方形化 13](#_Toc533612977)

[3.3.3. 恢复率高 14](#_Toc533612978)

[4. QR Code结构 15](#_Toc533612979)

[4.1. 位置探测图形(finder pattern) 15](#_Toc533612980)

[4.2. 分隔符(separator) 16](#_Toc533612981)

[4.3. 定位图形(timing pattern) 16](#_Toc533612982)

[4.4. 校正标识(alignment pattern) 16](#_Toc533612983)

[4.5. 格式信息(format information) 17](#_Toc533612984)

[4.6. 版本信息(version information) 17](#_Toc533612985)

[4.7. 数据及容错码(data and error correction codewords) 17](#_Toc533612986)

[4.8. Version 1 QR码结构 17](#_Toc533612987)

[5. 编码原理 19](#_Toc533612988)

[5.1. 数据分析 19](#_Toc533612989)

[5.1.1. 数据容量计算 19](#_Toc533612990)

[5.2. 编码数据 26](#_Toc533612991)

[5.2.1. 确定模式指示符 26](#_Toc533612992)

[5.2.2. 确定字符长度指示符 28](#_Toc533612993)

[5.2.3. 对数据内容编码 29](#_Toc533612994)

[5.2.4. 编成8位码字 33](#_Toc533612995)

[5.2.5. 结尾不足8位补0 33](#_Toc533612996)

[5.2.6. 添加终止符 33](#_Toc533612997)

[5.3. 计算纠错码 34](#_Toc533612998)

[5.3.1. 纠错算法 34](#_Toc533612999)

[5.4. 构造最终数据信息 34](#_Toc533613000)

[5.5. 填充位流到矩阵 35](#_Toc533613001)

[5.6. 应用掩码 37](#_Toc533613002)

[5.6.1. 掩码种类 37](#_Toc533613003)

[5.6.2. 使用方法 39](#_Toc533613004)

[5.7. 填充格式信息 41](#_Toc533613005)

[5.8. 填充版本信息 43](#_Toc533613006)

[5.9. 实例分析 44](#_Toc533613007)

[5.9.1. 对字母数字ABCDE123编码(版本1-H) 44](#_Toc533613008)

[5.9.2. 对数字01234567编码(版本1-H) 44](#_Toc533613009)

[6. 中文编码原理 44](#_Toc533613010)

[7. 其他 46](#_Toc533613011)

[7.1. 添加logo图片 46](#_Toc533613012)

[8. 解码流程 47](#_Toc533613013)

[8.1. QR码定位校正流程 47](#_Toc533613014)

[8.2. QR码译流程 47](#_Toc533613015)

[8.2.1. 总结 47](#_Toc533613016)

[9. 参考文档 48](#_Toc533613017)

# 什么是二维码

二维码 （2-dimensional bar code），是用某种特定的几何图形按一定规律在平面（二维方向上）分布的黑白相间的图形记录数据符号信息的。

在二维条码中，常用的码制有：QR Code、Data Matrix、PDF417、Maxi Code, Aztec, , Vericode, , Ultracode, Code 49, Code 16K等。

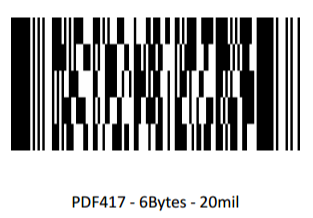
## 二维码的结构

常见的结构有两种：堆叠式和矩阵式

### 堆叠式/行排式

其编码原理是建立在一维条码基础之上，形态上是由多行短截的一维条码堆叠而成。

常见的有：PDF417、Code 16K、Code 49等，如下图：

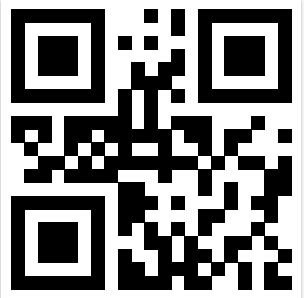
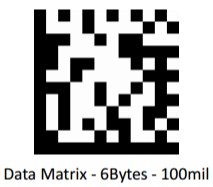


### 矩阵式二维码

矩阵式二维条码（又称棋盘式二维条码）它是在一个矩形空间通过黑、白像素在矩阵中的不同分布进行编码。

在矩阵相应元素位置上，用黑点（方点或其他形状）的出现表示二进制“1”，点的不出现表示二进制的“0”，点的排列组合确定了矩阵式二维条码所代表的意义。

具有代表性的矩阵式二维条码有：QR Code、Data Matrix、Code One、MaxiCode、Han Xin Code、Grid Matrix 等，如下图：

二维码在现实生活中的应用越来越普遍，由于QR CODE的流行，所以二维码又称QR code。

# 什么是QR Code？

QRCode（Quick Response Code：快速响应码）属于矩阵式，是二维条形码中最常用的一种格式。由DENSO(日本电装)公司开发，由JIS和ISO将其标准化。

## 特点

### 信息容量大

传统的条形码只能处理20位左右的信息量，与此相比，QR码可处理条形码的几十倍到几百倍的信息量。

数据类型与容量（参照**最大规格**符号版本**40-L**级）：

数字数据：7,089个字符

字母数据：4,296个字符

8位字节数据：2,953个字符

汉字数据：1,817个字符

各个版本的容量，查看如下链接：

<https://www.qrcode.com/zh/about/version.html>

### 占用空间小

QR码使用纵向和横向两个方向处理数据，如果是相同的信息量，QR码所占空间为条形码的十分之一左右。(还支持Micro QR码，可以在更小空间内处理数据。)



### 存储类型多

QR码是日本发明的，因此非常适合处理日文字母和**汉字**。

QR码字集规格定义是按照日本标准“JIS第一级和第二级的汉字”制定的，因此在日语处理方面，**每一个**全角字母和**汉字**都用**13比特**的数据处理，效率较高，与其他二维码相比，可以多存储20%以上的信息。



QR Code码可高效地表示汉字，相同内容，其尺寸小于相同密度的PDF417条码。其专有的汉字模式**更加适合我国**应用。因此，QR code在我国具有良好的应用前景。

### 容错能力强

QR码具备“纠错功能”，即使部分编码变脏或破损，也可以恢复数据。数据恢复**以码字为单位**（**码字**是组成内部数据的单位，在QR码的情况下，**每8个比特代表1码字**），最多可以纠错约30%（根据变脏和破损程度的不同，也存在无法恢复的情况）。

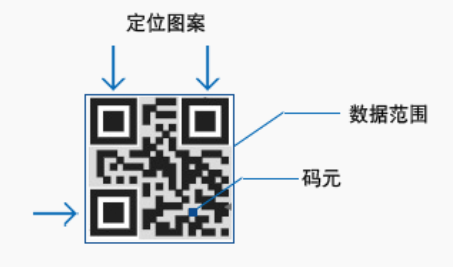
二维码的纠错级别，分为L、M、Q、H四个等级，如下：



在规格一定的条件下，纠错等级越高其真实数据的容量越小。

### 全方位识别

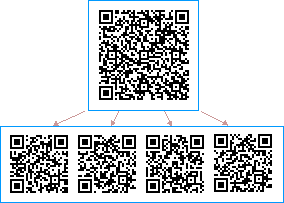
QR码从360°任一方向均可快速读取。其奥秘就在于QR码中的3处定位图案，可以帮助QR码不受背景样式的影响，实现快速稳定的读取。



### 支持数据合并功能

QR码可以将数据分割为多个编码，最多支持16个QR码。使用这一功能，还可以在狭长区域内打印QR码。另外，也可以把多个分割编码合并为单个数据。





### 结构连接(可选)

可用1-16个QR Code码符号表示一组信息

### 扩充解释(可选)

这种方式使符号**可以表示缺省字符集以外的数据**（如阿拉伯字符、古斯拉夫字符、希腊字母等），以及其他解释（如用一定的压缩方式表示的数据）或者对行业特点的需要进行编码。

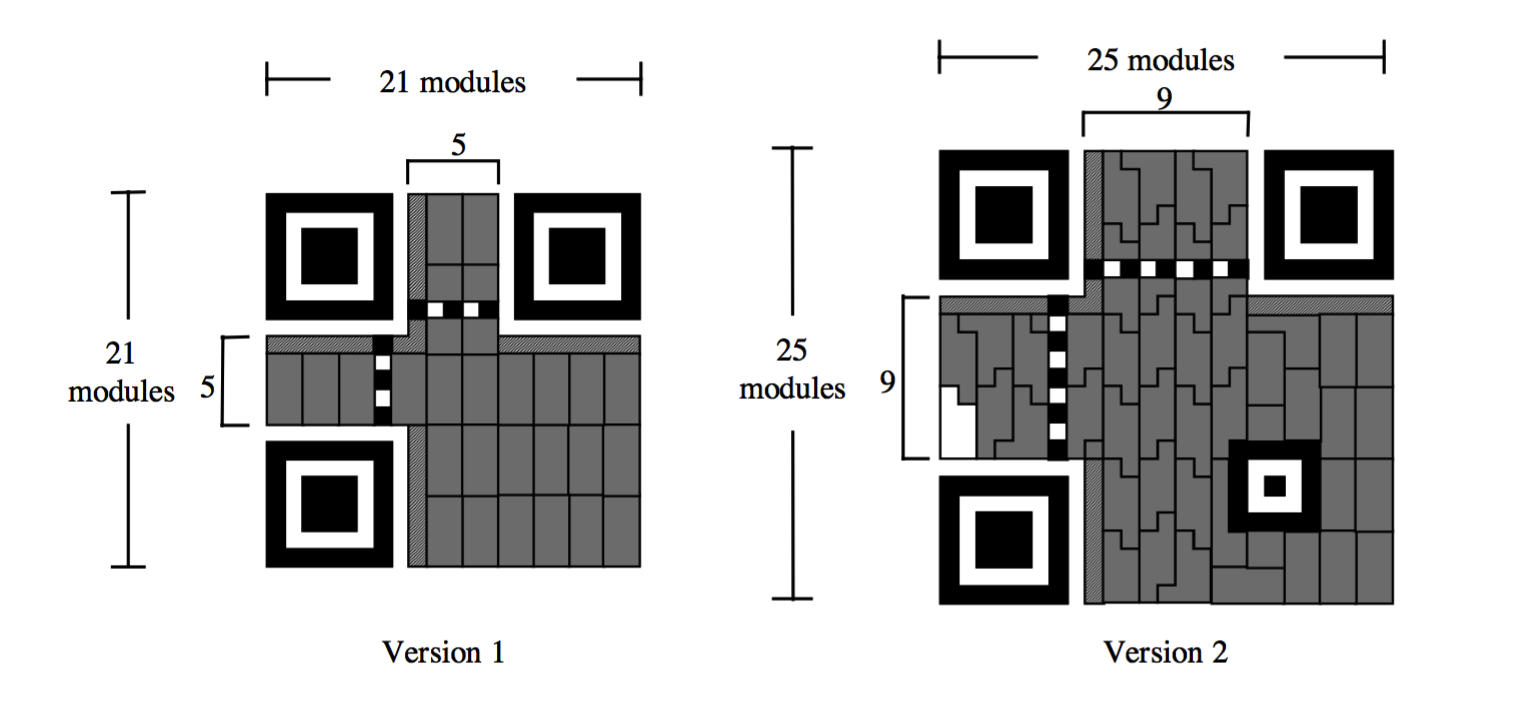
## QR Code版本

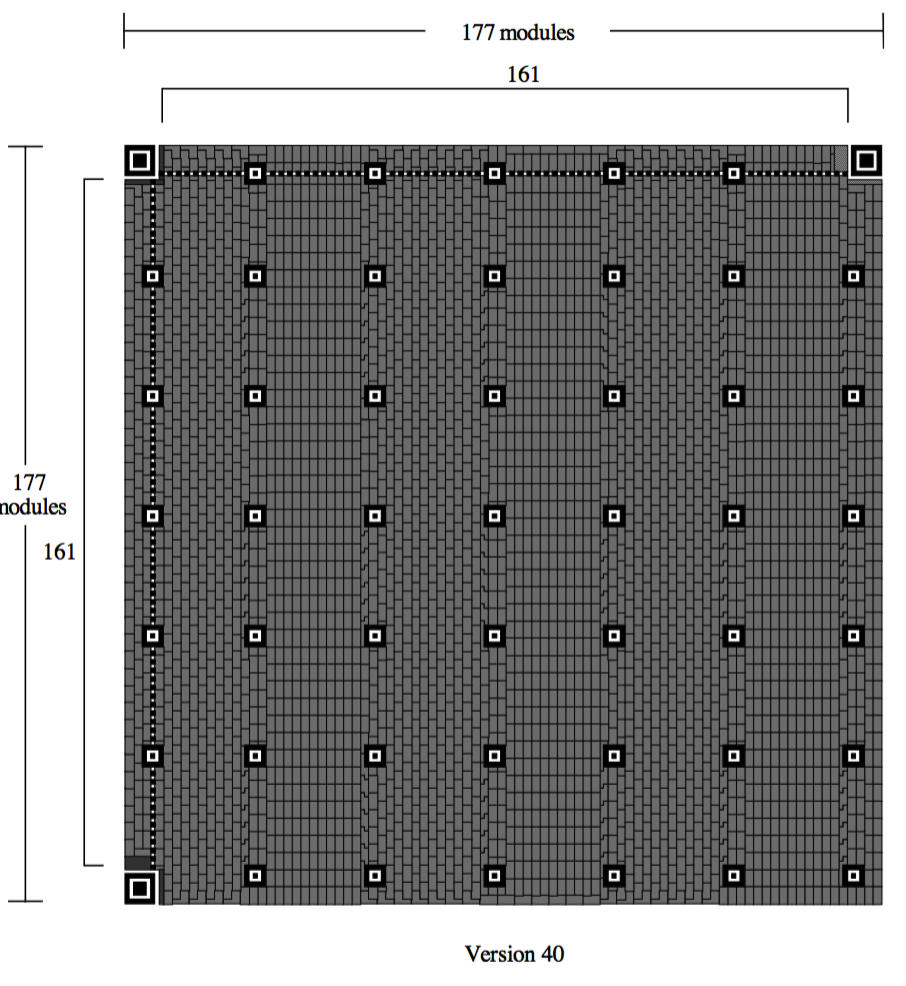
QR码共有1到40个不同的版本（种类)，每个版本都具备固有的**码元**结构(码元数)。码元是指构成QR码的方形黑白点。

“码元结构”是指二维码中的码元数。从版本1(21码元x21码元)开始，一直到版本40(177码元x177码元)，在纵向和横向，每一版本比前一版本每边增加4个模块。

**公式**：(V-1)\*4+21







QR码的各个版本结合数据量、字符类型和纠错级别，均设有相对应的最多输入字符数。

## 可编码字符集

QR码支持编码的内容包括 **纯数字**、**数字和大写英文字母和9个符号混合**编码、**8位字节码**、包含汉字在内的**多字节字符**，(**缺省**字符集)如下：

1. **数字**：每三个为一组压缩成**10bit**
2. **混合字符**：每两个为一组，压缩成**11bit**
3. 字节数据：也就是8bit字节数据，使用ISO/IEC 8859-1编码，无压缩直接保存
4. 多字节字符：包含汉字、日文等，每一个字符被压缩成13bit

## QR Code标准

目前，QR码已经在国家标准和国际标准中实现标准化，任何人都可以随意查看该标准。

DENSO WAVE INCORPORATED已宣布，不行使本公司就标准QR码拥有的专利权(专利第2938338号)。

### QR码的标准化历程



### 如何获取QR码的标准文件

QR码在ISO(ISO/IEC18004)中得到标准化。因此，请从以下团体的网站购买QR码标准文件：<https://www.iso.org/home.html>



# QR Code种类



## QR码

### 模型1

是最早制作的QR码，最高版本为14（73×73码元），最多可以处理1167位数字。

### 模型2

QR码的模型2是模型1的改良版，它在二维码变形的情况下也能顺利读取数据。

当二维码被打印在曲面上，或者由于读取角度等原因导致二维码变形时，仍然可以通过设置在二维码内部的**对齐模式**有效读取。

最高版本为40（177×177码元），最多可以处理7089位数字。

现在使用的都是 **模型2**

## Micro QR码

特点如下：

### 只有一个定位图案

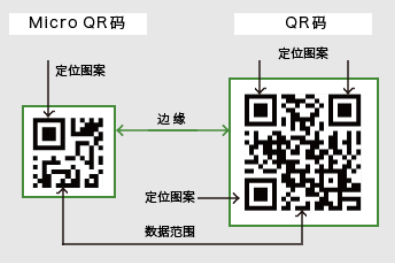
### 占用空间小

### 数据容量小

### 已经标准化

与普通的QR码一样，Micro QR码也已经公开规格，并且DENSO WAVE INCORPORATED已宣布，不行使本公司就Micro QR码拥有的专利权(JP2938338)。

Micro QR码 详细参考此链接：<https://www.qrcode.com/zh/codes/microqr.html>



## iQR码

特点如下：

### 容量大尺寸小

与传统QR码相同尺寸，iQR码可以多存储80%的信息量，如果要存储相同信息量，与传统QR码相比，尺寸可以缩小30%。



### 长方形化

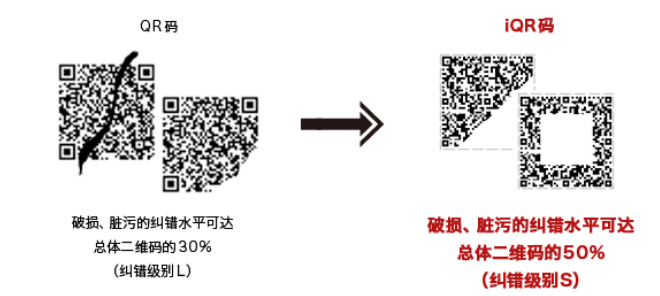
QR码不仅可以生成正方形，还可以生成长方形。这样可以将打印条形码的空间替换为iQR码。



### 恢复率高

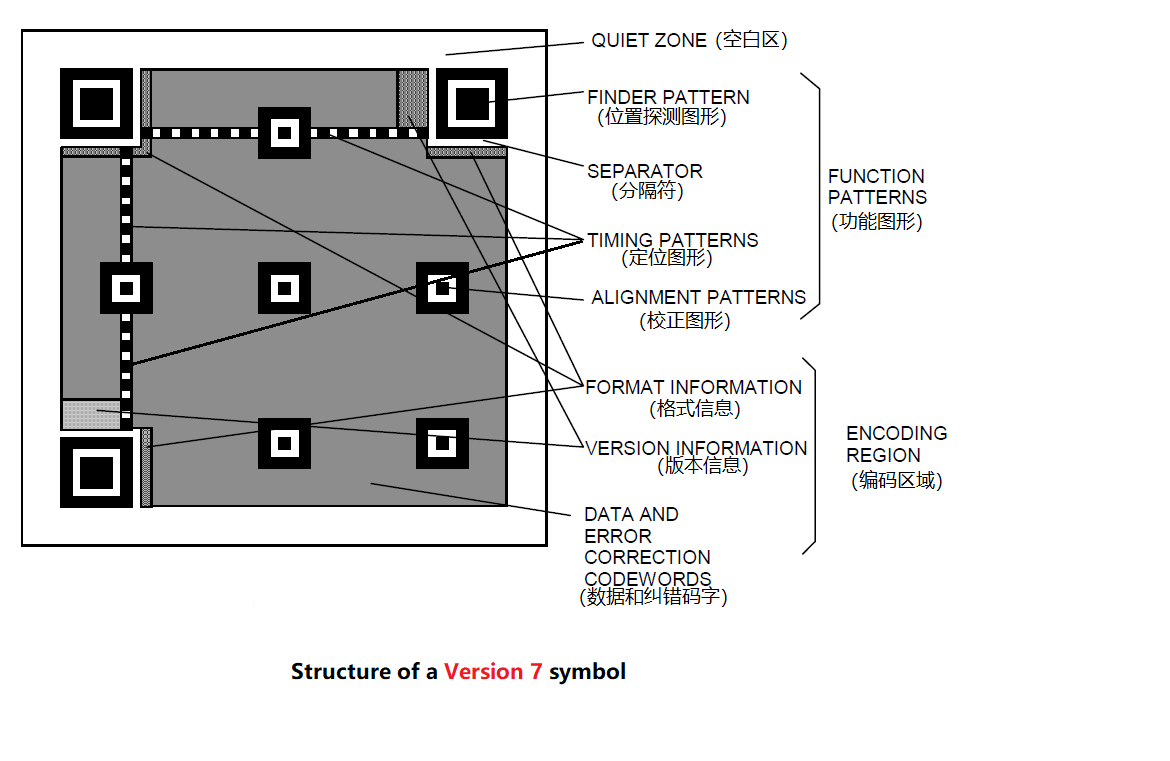
iQR码具备了比传统QR码更高的二维码恢复功能。

在二维码破损、脏污等的数据纠错级别方面，传统QR码最高可纠正整体的30%，而iQR码的纠错功能高达50%。



参考此链接：<https://www.qrcode.com/zh/codes/iqr.html>

# QR Code结构



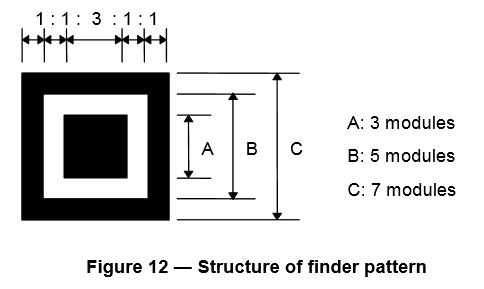
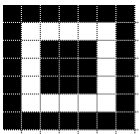
二维码主要由两部分构成：

1. 功能图形：功能图形不进行编码
2. 编码区域：保存编码数据

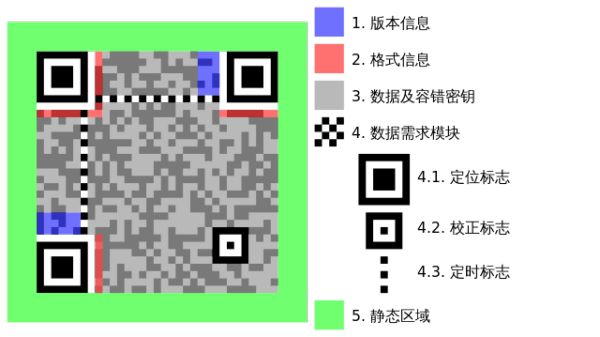
## 位置探测图形(finder pattern)

QR 码符号中有3个位置探测图形，分别位于符号图像4个角中的3个角：左上角、右上角、左下角，每个位置探测图像都是由固定深浅颜色的模块组成，用来确定QR码的位置和方向。

模块深浅颜色顺序为：深色—浅色—深色—浅色—深色，各元素宽度的比例为1:1:3:1:1，如下图所示：

即使图像有旋转，位置探测图像的模块颜色顺序和宽度比例也不变。



## 分隔符(separator)

每个位置探测图形和编码区域之间有一条1单位宽度的分隔符，由白色块组成，以保证 位置探测图形 清晰可识别。

## 定位图形(timing pattern)

黑白色相间交替组成的一行一列两条一单位宽的黑白交替点带，由黑色起始和结束，位于横纵的两两探测图形之间，用于**确定符号的密度和版本**，提供决定模块坐标的**基准位置**，也就是 **确定坐标系**。

## 校正标识(alignment pattern)

**大于version 1**的都有校正标识。校正标识用于进一步校正坐标系。校正标识的数量取决于版本。

二维码的版本一旦确定，校正图形的数量和位置也就确定了。矫正图形主要用于QR码形状的矫正，尤其是当QR码印刷在不平坦的面上，或者拍照时候发生畸变等。

## 格式信息(format information)

存储 **2位容错级别** 和 **3位掩码代码**，和 **10位**自身的BCH容错码。

## 版本信息(version information)

此区域用来存储版本信息。版本1没有版本信息。

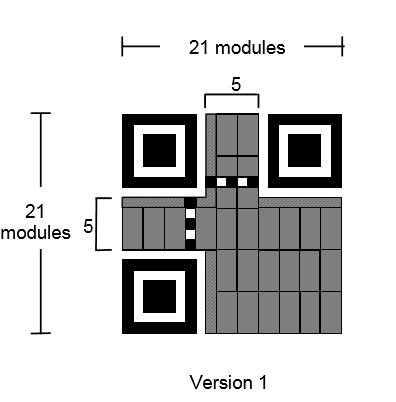
## 数据及容错码(data and error correction codewords)

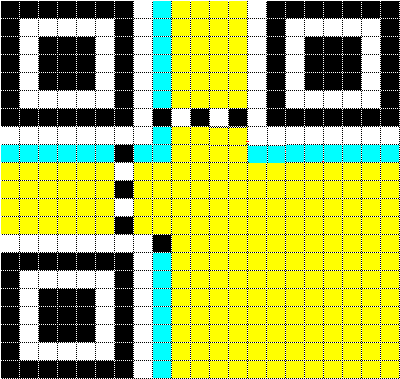
存储实际编码的数据和数据的RS容错码。

8个格子可以编码一个字节

纠错码：用于修正二维码损坏带来的错误

## Version 1 QR码结构

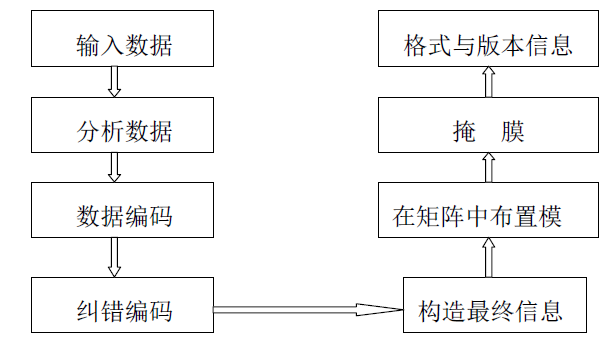




# 编码流程

编码就是把常见的 数字、字符等转换成QR码的方法。

QR码的编码流程如下：



以下举例说明将“ABCDE123”转换成为版本1、Level H的QR码转换方法。

## 数据分析

分析输入数据，根据数据决定要使用的 **编码模式、容错级别、QR码版本**。

1. 低版本的QR码无法编码过长的数据
2. 含有非数字字母字符的数据要使用**扩展字符编码**模式。

如果没有指定version，一般会采用能表示目标信息的最小version。

### 数据容量计算

**一旦版本确认之后**，当前版本的**数据模块数**就确认了，公式：**D=A2-B-C**

参数解释：

D：数据模块总数，也就是码元数目， 1个bit占1个码元

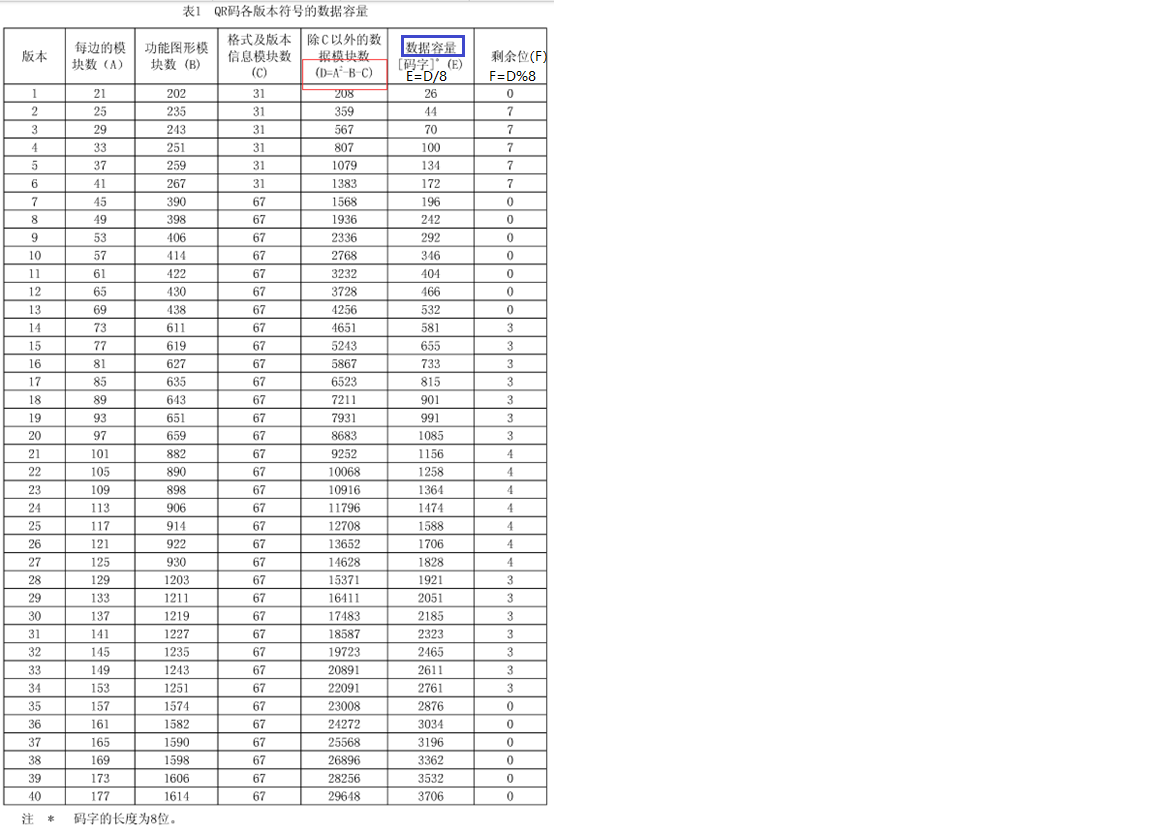
A：每边的模块数

B：功能图形的模块数

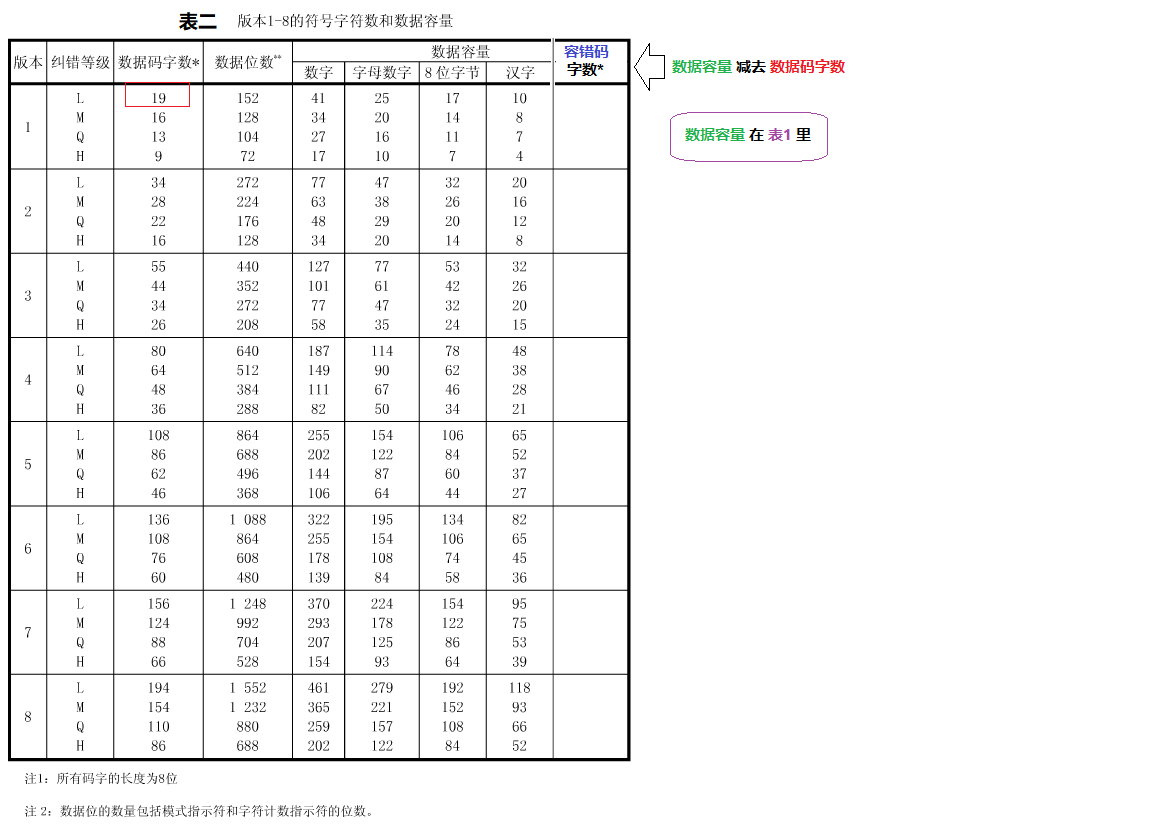
C：格式及版本信息模块数

E：数据容量，也就是码字个数，一个码字占8个bit，也就是8个码元。**数据容量**由**数据码**和**容错码**。由下表1和 表2可知，V1一共有26个码字，19个数字码和7个容错码。

F：剩余位



### 不同版本的数据码数量



详细的参考中文文档：P23

在版本确定的条件下，纠错等级越高，其真实数据的容量越小。

QR码的最大容量取决于选择的版本、纠错级别和编码模式。以版本1、纠错级别为Level Q的QR码为例，可以存储27个纯数字，或16个字母数字混合字符，或11个8bit字节数据。如果要存储同样多的内容同时提高纠错级别，则需要采用更高的版本。

## 编码数据

确定所采用的模式后，按照其定义的规则，将数据字符转换为位流。在当需要进行模式转换时，在新的模式段开始前加入模式指示符进行模式转换。在数据序列后面加入终止符。

将数据字符转换为位流，每8位一个码字，整体构成一个数据的码字序列。必要的时候需加入填充字符以填满按照版本要求的数据码字数。

整个码字序列由模式指示符开始。

### 确定模式指示符

QR码的编码模式(Mode)就是前文提到的 数字、字符、8bit 字节码、多字节码等。对于不同的模式，都有对应的模式标识符（Mode Indicator)来帮助解码程序进行匹配。

模式指示符 是**4bit**的二进制数，如下：



以下是 **常用的**模式：

|  |  |  |
| --- | --- | --- |
| **模式** | **指示符** | **备注** |
| **数字**(numeric mode ) | **0001** |  |
| **字母数字**(alphanumeric mode) | **0010** |  |
| **8位**字节(byte mode) | **0100** |  |
| 日本汉字(KANJI mode） | 1000 |  |
| 中国汉字(GB2312) | 1101 | 对应的汉字和非汉字字符 |
| **终止符** | 0000 |  |

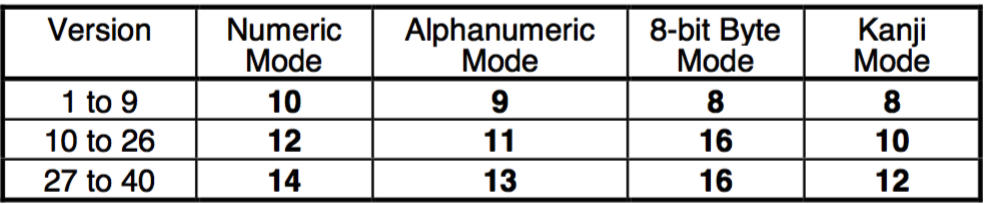
由于示例文本串“ABCDE123”是混合字符，因此将选择alphanumeric mode，其标识码为：0010

### 确定字符长度指示符

文本串计数标识符用来存储源内容字符串的长度，在版本**1-9**的QR码中，文本串长度标识符被定义为：

|  |  |  |
| --- | --- | --- |
| **模式** | **字符数指示符 长度** | **备注** |
| **数字**(numeric mode ) | **10bit** |  |
| **字母数字字符**(alphanumeric mode) | **9bit** |  |
| **8位**字节(byte mode) | **8bit** |  |
| 日本汉字(KANJI mode） | 13bit |  |
| 中国汉字(GB2312) | 13bit |  |

下表 是1~40，**不同version**下，每个模式对应的 [字符数指示符] 的长度：



由于示例文本串“ABCDE123”是混合字符，所以长度为9bit。源文本串一共有8个字符，因此将字符个数8编码为**9bit**二进制表示：000001000

加上混合字符模式标识码，总的编码为0010 000001000

### 对数据内容编码

#### 数字模式

数字模式是对10进制数字**0~9**(ASCII的16进制值为30到39)进行编码。

在数字模式下，每**3个**数据 分为一组，分成若干段。如："123456" 将分成"123" 和 "456"，分别被编码成10bit的二进制数。“123”的**10bit**二进制表示法为：0001111011，实际上就是二进制的123。

当数据的长度不足3个数字时，如果只有**1个**数字则用4bit，如果只有有**2个**数字就用**7bit**来表示。如："9876"被分成"987"和"6"两段，因此被表示为"1111011011 0110"。

数字模式中，二进制位流的长度计算**公式**如下：

**B = 4 + C + 10(D DIV 3) + R**

参数:

B = 总的位流位数

C = 字符长度指示符的位数（和版本有关）

D = 输入的数据字符数

R = 0 if (D MOD 3) = 0，余数为0

R = 4 if (D MOD 3) = 1，余数为1

R = 7 if (D MOD 3) = 2，余数为2

#### 字母数字模式

此模式一共有45个字符，包含的字符如下：

1. **10个**数字0～9
2. **26个大写**字母A～Z、
3. **9个**其他字符：space，$， %，\*，+，-，.，/，：）



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **字符** | **值** |  | **字符** | **值** |  | **字符** | **值** |  | **字符** | **值** |  | **字符** | **值** |
| **0** | **0** |  | **A** | **10** |  | **K** | **20** |  | **U** | **30** |  | **+** | **40** |
| **1** | **1** |  | **B** | **11** |  | **L** | **21** |  | **V** | **31** |  | **-** | **41** |
| **2** | **2** |  | **C** | **12** |  | **M** | **22** |  | **W** | **32** |  | **.** | **42** |
| **3** | **3** |  | **D** | **13** |  | **N** | **23** |  | **X** | **33** |  | **/** | **43** |
| **4** | **4** |  | **E** | **14** |  | **O** | **24** |  | **Y** | **34** |  | **:** | **44** |
| **5** | **5** |  | **F** | **15** |  | **P** | **25** |  | **Z** | **35** |  |  |  |
| **6** | **6** |  | **G** | **16** |  | **Q** | **26** |  | **Space** | **36** |  |  |  |
| **7** | **7** |  | **H** | **17** |  | **R** | **27** |  | **$** | **37** |  |  |  |
| **8** | **8** |  | **I** | **18** |  | **S** | **28** |  | **%** | **38** |  |  |  |
| **9** | **9** |  | **J** | **19** |  | **T** | **29** |  | **\*** | **39** |  |  |  |

通过上表，对于每一个输入的字符，我们都能找到它对应的值，它的值为 0到44。

在混合字符模式下，每**2个**数据 分为一组，每组的 **第一个**字符 **乘上45**，再**加上 第二个**数字。每组被编码成**11bit**的二进制数，如果**字符个数 只有1个**，则用**6bit**表示

示例文本串”ABCDE123”，使用V1-H编码后的结果为：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **模式指示符** | 字符长度 |  |  |  |  |
|  |  |  | AB | CD | E1 | 23 |
|  |  |  | 45\*10+11 | 45\*12+13 | 45\*14+1 | 45\*2+3 |
| 十进制 |  |  | 461 | 553 | 631 | 93 |
| 二进制 | 0010 | 000001000 | 00111001101 | 01000101001 | 01001110111 | 00001011101 |
|  |  |  |  |  |  |  |

字母数字模式中，二进制位流的长度计算**公式**如下：

**B = 4 + C + 11(D DIV 2) + 6(D MOD 2)**

参数:

D = number of input data characters

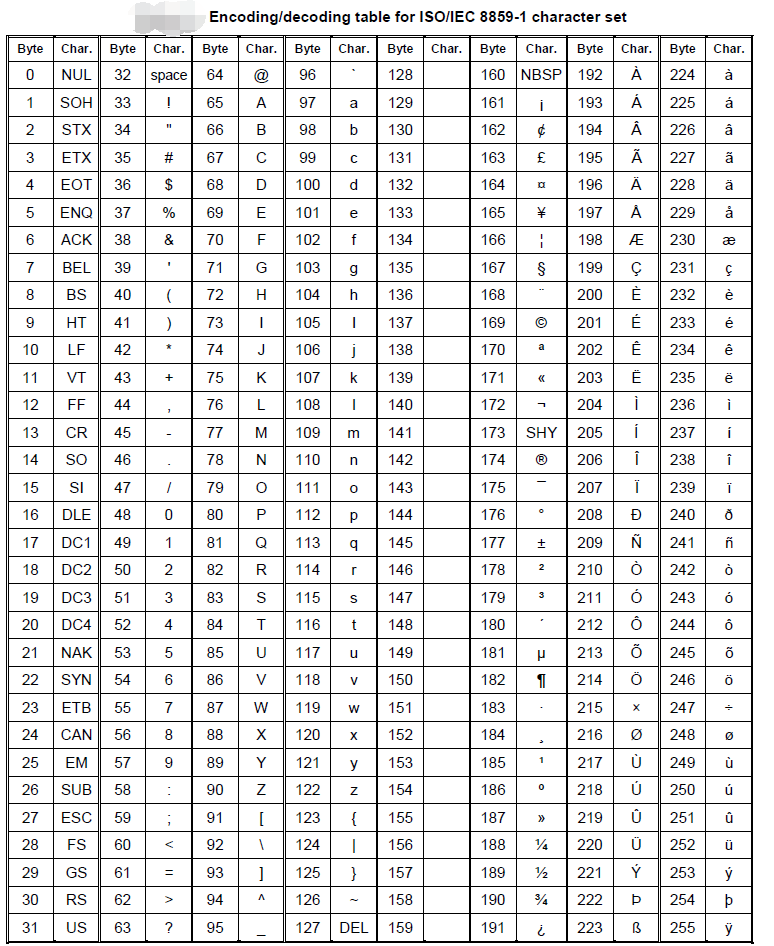
B = 总的位流位数

C = 字符长度指示符的位数（和版本有关）

D = 输入的数据字符数

#### 字节模式

在此模式下，**一个8bit**的码字直接表示 **一个输入的字符**，如下表所示：



上表的0到127和ASCII码一致。

在字节模式下，我们可以 **把每一个输入的字符，按照ASCII码转换为8bit或16bit的二进制数**。

字节模式中，二进制位流的长度计算**公式**如下：

B = 4 + C + 8D

参数:

B = 总的位流位数

C = 字符长度指示符的位数（和版本有关）

D = 输入的数据字符数

#### 汉字模式

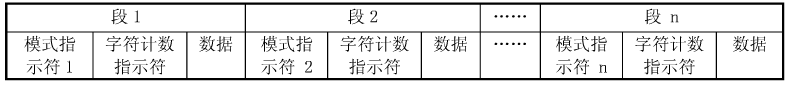
此模式下，每个双字节字符用13bit的二进制表示。

#### 混合模式

从数字模式到汉字模式，每种模式里字符所需的位数依次增加。

对于一串要编码的符号(例如：数字序列后面跟随字母数字序列)，通过**模式的转换**，把这串符号拆分为多种模式混合的字符串，可以使数据的位流长度最短，因此对于一串数据进行混合编码可以比使用单独的一种编码效率更高。

混合模式的**数据格式**，如下：

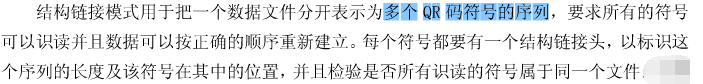


基本结构为 [**模式指示符**/**字符数指示符**/**数据**]，其后紧跟下一段的**模式指示符**，用来开始下一段，上图为有N端数据的结构。

**注意**：

理论上说，用每个数据字符所需的位数最少的模式进行编码是最高效的，但**每次模式转换时都需要有相关的模式指示符 和 字符数量指示符 等形式上的开销**，因此对于较少的字符数，模式的转换并不总是能使位流总量最少。

#### 结构连接模式



### 添加终止符(0000)

在数据的尾部添加 **0000**终止符

### 编成8位码字

将以上文本串”ABCDE123”的编码再按8bit一组，形成码字(code words):

00100000 01000001 11001101 01000101 00101001 11011100 00101110 10000

#### 结尾不足8位补0

如果尾部数据 **不足8bit**，则在**尾部 填充0**，**构成8位数据**：

00100000 01000001 11001101 01000101 00101001 11011100 00101110 10000000

如果数据最后正好是8位，那么就不用补0。

### 添加填充码字

如果编码后的 **数据码字** **不足 当前版本 和 纠错级别** 所存储的**最大**容量，则在尾部补充 "11101100" 和 "00010001"，直到全部填满。具体姿势是 **不断重复11101100和00010001这两个码字**。

最后，版本1、Level H下的"ABCDE123" 的QR码是：

00100000 01000001 11001101 01000101 00101001 11011100 00101110 10000000 **11101100**

十进制表示法为：

32 65 205 69 41 220 46 128 236

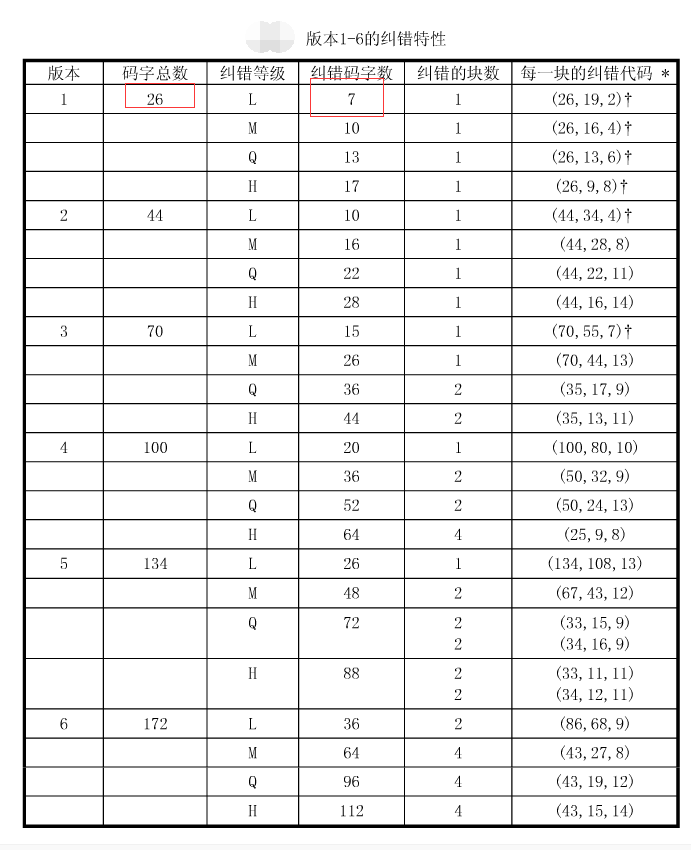
## 计算纠错码

在二维码版本和纠错等级确定的情况下，其所能容纳的码字总数和纠错码字数也就确定了，比如：版本10，纠错等级时H时，总共能容纳346个码字，其中224个纠错码字。

也就是说二维码区域中大约1/3的码字时冗余的。对于这224个纠错码字，它能够纠正112个替代错误（如黑白颠倒）或者224个据读错误（无法读到或者无法译码）。

按版本和纠错等级最终创建所需要的纠错码字，并把**纠错码字 加入到数据码字序列 后面**，成为一个新的序列。

### 不同版本的纠错码数量



详细的参考英文文档：P46，中文文档：P30

### 纠错算法

纠错算法可纠正两种类型的错误：

1、**拒读错误(**错误码字的位置已知)，是没有扫描到或者无法译码的字符。

2、**替代错误**(错误码字的位置未知)，是错误译码的字符。

一个替代错误需要两个纠错码字来纠正。

这里有个公式：e + 2t <= d - p

e:拒读错误数 t:替代错误数 d:纠错码字数 p:错误译码保护码字

主要通过 [**里德-所罗门纠错算法**](https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction) 来实现。

## 构造最终的码字序列

详细的参考中文文档：P41

## 填充码字到矩阵

把数据和功能性图样 根据标准填充到矩阵中。

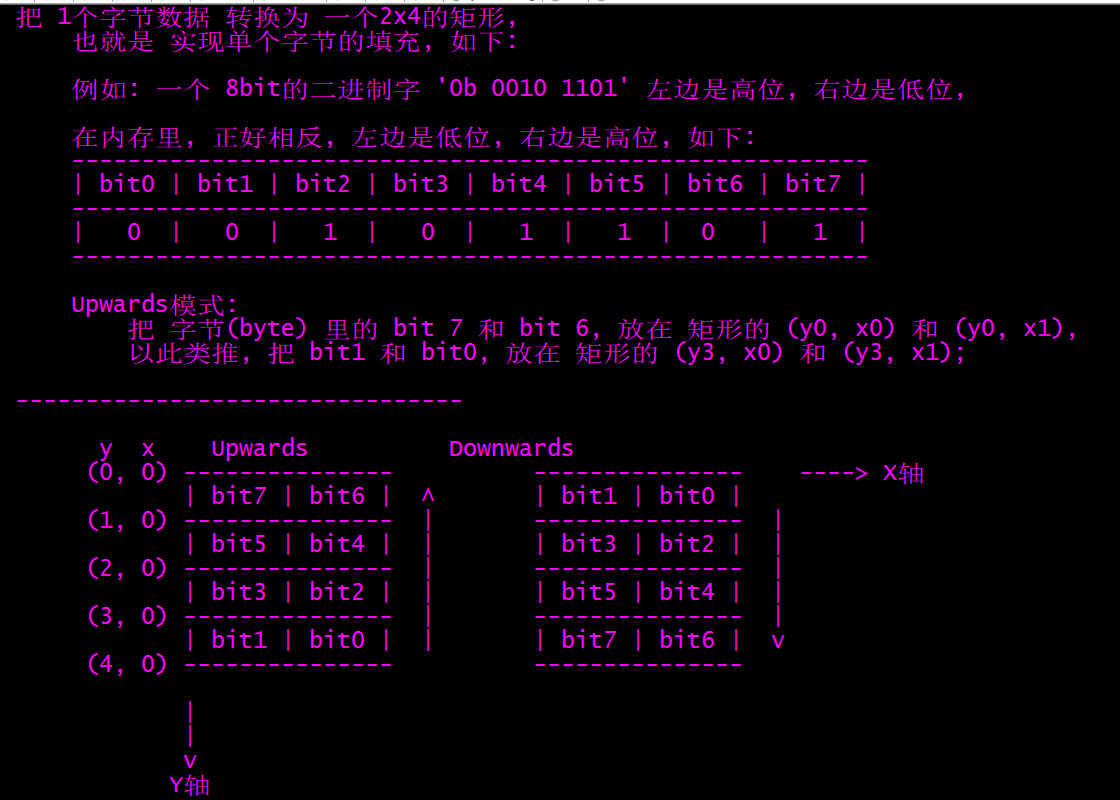
### 功能图形的布置

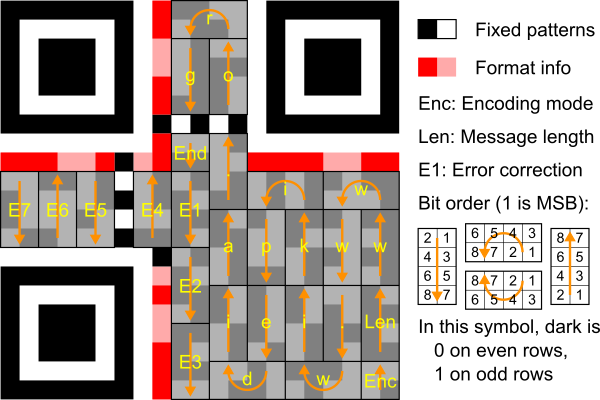
将 探测图形、分隔符、定位图形、校正图形 填入相应的位置，他们对所有的版本都是相同的。

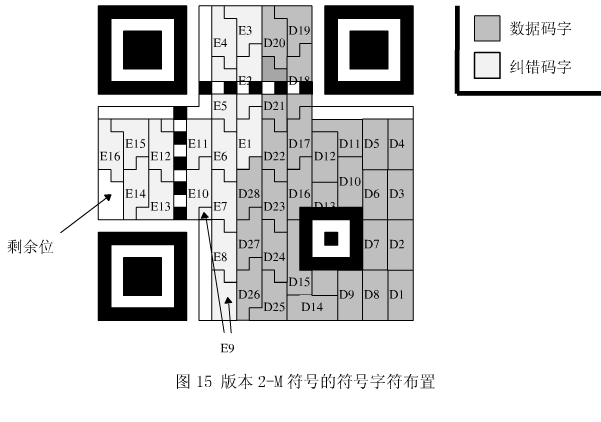
### 符号字符的布置

将**码字模**块(数据码和纠错码) 放入矩阵中，如下 以V1-L为例：

QR码标准将8个二进制位（1字节）规定为一个**数据元组**，先将编码后数据的每一个字节填充到 2×4 的矩阵（高版本QR码中会出现不规则形状的字节元组）中，再将这些小的矩阵填入QR码矩阵。标准也规定了字节填入小矩阵的方式：







详细的参考中文文档：P42

## 应用掩码

将掩码图形**用于**符号的 **编码**区域，使得二维码图形中的深色和浅色（黑色和白色）区域能够比率最优的分布。

也就是为了可靠性，使用掩码图形让矩阵中的黑白色块比例接近1:1并均匀分布，尽可能避免探测图形出现在别的区域，使因相邻模块的排列造成译码困难的可能性降为最小。所以每个二维码咋一看，都是一模一样。

关于掩码有几点需要说明：

1、**不影响 功能区、格式信息和版本信息**(Function Patterns)

2、对编码区域(Encoding Region)进行 **异或**(**XOR)运算**(**黑色为1，白色为0**)

3、对结果图不符合要求的部分计分

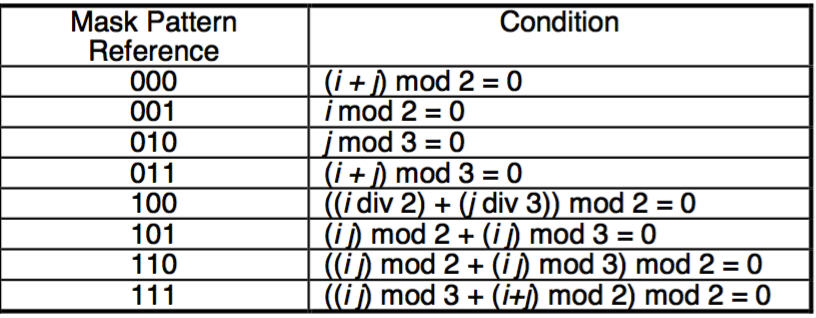
4、评估后选择最佳的掩码(**分数最低的**那个)

5、**不影响 格式**信息

### 掩码种类

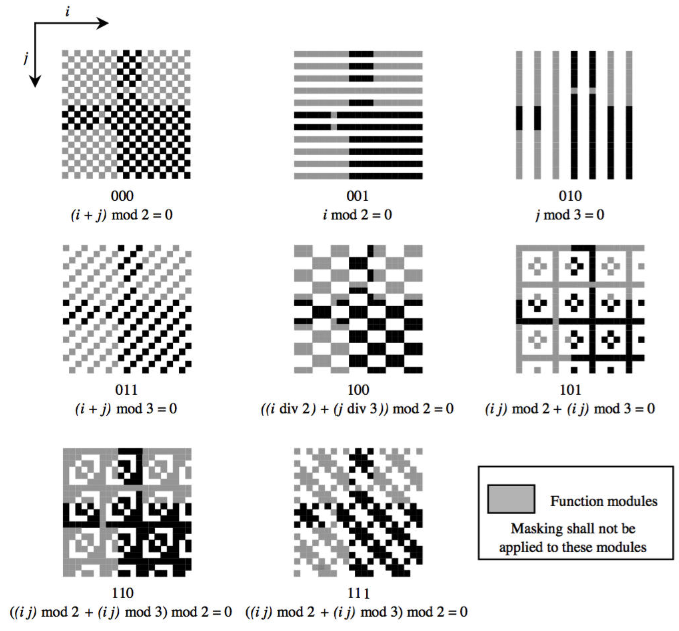
考虑到数据的多样性，一种掩码难以达到预期的效果，所以QR标准定义了八个掩码，要求在应用掩码时先分别应用所有的掩码产生八个结果，然后根据惩罚规则计算出每个结果矩阵的惩罚分，再选出**惩罚分最小**，效果最好的掩码当做最终结果。

QR码一共定义了8种掩码，如下，i代表行(row)，j代表列(column)：

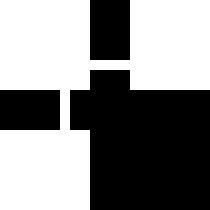


|  |  |  |
| --- | --- | --- |
| **模块颜色** | **掩码索引** | **条件** |
| **黑色**  (dark) | 000 | if (row + column) mod 2 == 0 |
| 001 | if (row) mod 2 == 0 |
| 010 | if (column) mod 3 == 0 |
| 011 | if (row + column) mod 3 == 0 |
| 100 | if ( floor(row / 2) + floor(column / 3) ) mod 2 == 0 |
| 101 | if ((row \* column) mod 2) + ((row \* column) mod 3) == 0 |
| 110 | if ( ((row \* column) mod 2) + ((row \* column) mod 3) ) mod 2 == 0 |
| 111 | if ( ((row + column) mod 2) + ((row \* column) mod 3) ) mod 2 == 0 |
|  |  |

下图给出了 **版本1** 符号 所对应的所有的掩码图像：

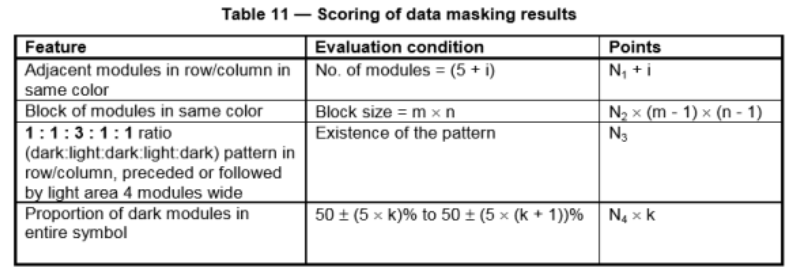


下图的黑色区域是掩码有效区：



### 使用方法

QR标准把惩罚分 分成了**四项**，分别对应行/列中的 **连续色条、大面积的色块、行/列中类似定位标识的部分、整个矩阵中颜色的不平衡** 做出加权惩罚。



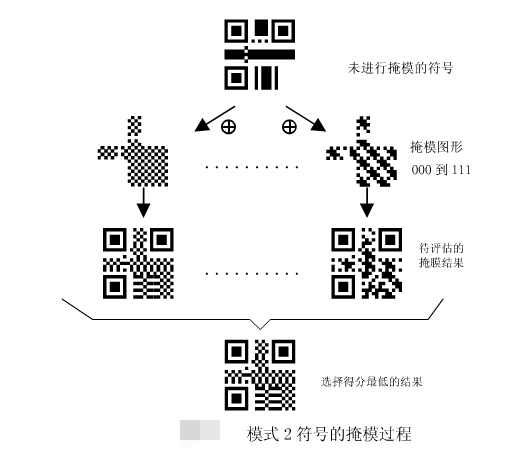
**罚点记分：**

在依次用每一个掩模图形进行掩模操作之后，要通过对每一次 上面情况的出现进行罚点记分，以便对每一个结果进行评估，**分数越高，其结果越不可用**。在上表中，N1到N4为对不好的特征所罚分数的权重（N1=3，N2=3，N3=40，N4=10），i为紧邻的颜色相同模块数大于5的次数，k为符号深色模块所占比率离50%的差距，步长为5%。

虽然掩模操作仅对编码区域进行，不包括格式信息，但评价是对整个符号进行的。

最终，应选择掩模结果中**罚分最低**的掩模图形 用于符号掩模。

----------------------------------------------------------------------------------------

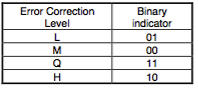


最终把当前QR码要使用的掩码ID添加到**格式信息**里，也就是告诉扫描软件应该用哪个掩码来还原数据。

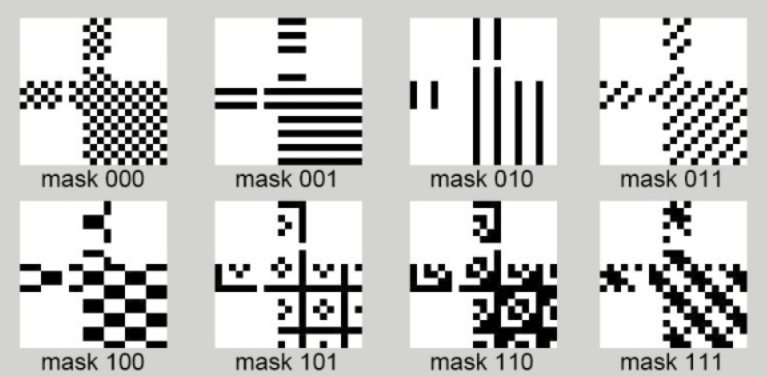
## 填充格式信息

格式信息一共15位，其中**5个数据位 和 10个纠错位（BCH）**。

1、数据位：**前2位** 是 **纠错等级**，如下：



2、数据位：**第3到第5位** 是 **掩码标识**，如下：

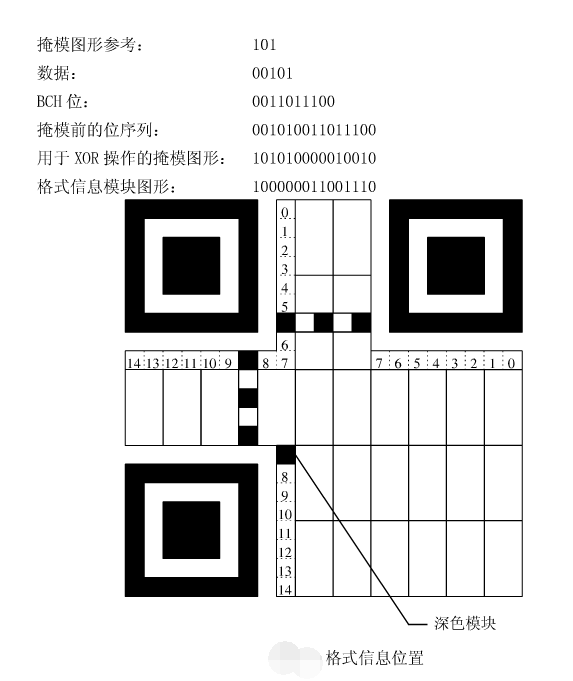


1. 纠错位：根据纠错算法 得到**10位** 的**纠错码**，添加到5位数据位**后面**。

这个纠错码是 **根据前5位数据计算出来的**。

4、最后将15位格式信息与掩码**10101 00000 10010**进行**异或**，以确保纠错等级和掩码图像合在一起的结果不全是0。

二维码上一共保存了**两份** 格式信息，它们是冗余的，如下水平和垂直方向：



其中**14**代表最**高位**（most significant bit），0代表最低位（least significant bit）。

## 填充版本信息

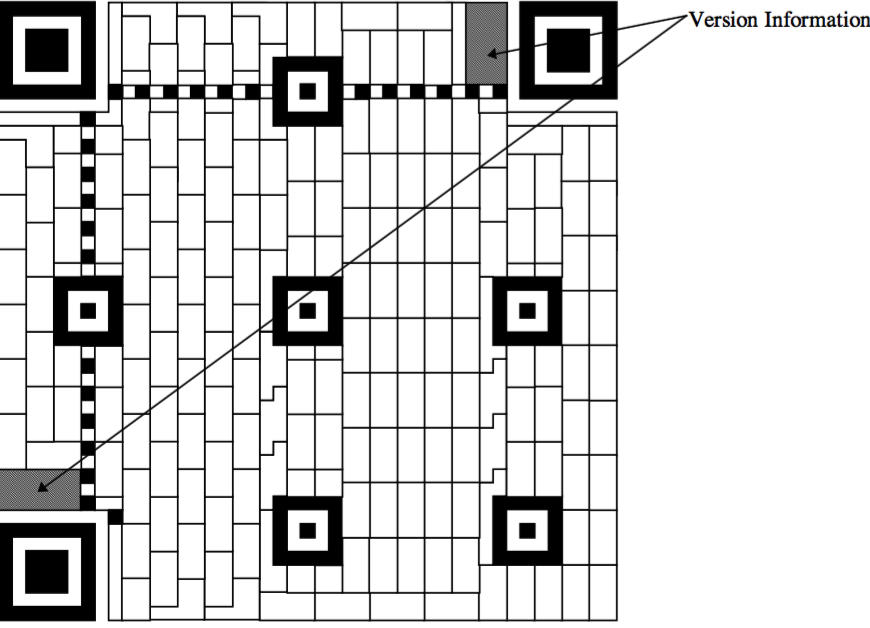
版本信息一共**18**位，**6个数据位 和 12个纠错位（BCH）**。

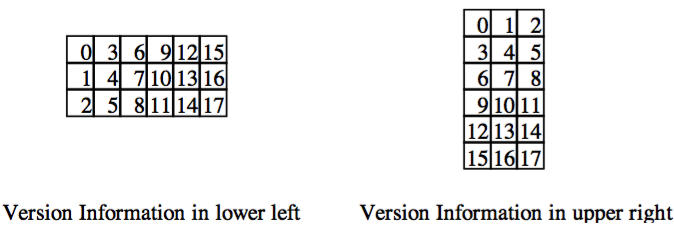
12位的就错位放在版本信息之后。

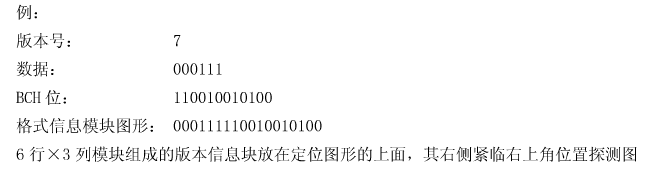
**7-40**的版本，都包含了版本信息，没有版本信息的全为0。**不用对版本信息做掩码**。

例如：如版本号8，数据位的信息为001000

二维码上一共保存了**两份** 版本信息，它们是冗余的，如下：









## 实例分析

### 使用数字模式进行编码

例如：01234567编码为(版本1-H)

1、分组：

每三个数字为一组，压缩成10bit，为012 345 67

1. 转成二进制：

012 → 0000001100

345 → 0101011001

67 → 1000011

1. 转成序列：

→ 0000001100 0101011001 1000011

4、字符数长度，数字模式为10bit，转成二进制：

数字长度是8 → 0000001000

5、加入模式指示符（上图数字）0001：

→ 0001 0000001000 0000001100 0101011001 1000011

对于字母、中文、日文等只是分组的方式、模式等内容有所区别。基本方法是一致的。

### 使用字母数字模式进行编码

例如：ABCDE12编码为(版本1-H)

1、字符数长度编码：字母数字模式为9bit，转成二进制：

字母数字长度是8 → 000001000

1. 加入模式指示符（上图数字）：0010：

总的编码为 → 0010 000001000

1. 数据内容编码：

→ 00111001101 01000101001 01001110111 00001011101

4、完整的编码：

→ 0010 000001000 00111001101 01000101001 01001110111 00001011101

### 使用字节模式进行编码

例如：01234567编码为(版本1-H)

### 使用混合模式进行编码

例如：ABCDE12编码为(版本1-H)

# 中文编码原理

中文汉字的与日文汉字转换步骤相似：

1、对于**第一字节**为**0xA1~0xA9**之间，也就是01到09区之间，第二字节在0xA1~0xFE之间字符：

a)第一字节减去0xA1；

b)上一步结果乘以0x60;

c)第二字节减去0xA1;

d)将b)步骤的结果加上c步骤的结果;

e)将结果转换为13位二进制串。

2、对于**第一字节**为**0xB0~0xFA**之间，第二字节在0xA1~0xFE之间字符：

a)第一字节减去0xA6；

b)上一步结果乘以0x60;

c)第二字节减去0xA1;

d)将b)步骤的结果加上c步骤的结果;

e)将结果转换为13位二进制串。

# 其他

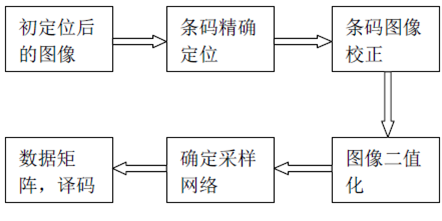
## 添加logo图片

为了美观，可以在二维码的中间添加一张logo图片。但是添加了logo之后，二维码的一些数据会被破坏，由于二维码本身具有容错能力，例如L级别可以纠正7%的错误，所以添加了logo之后的二维码依然可以识别。

注意：如果我们添加了logo之后，破坏的模块超过当前纠错级别，那么这个二维码就无法识别了。所以我们一般添加logo占二维码的长宽比例，设置为20%，纠错级别选择M以上，测试后发现不影响识别。

# 解码流程

## QR码定位校正流程



## QR码译流程

### 过程解析

1. 定位并获取图形，根据黑白色块转成0, 1组成的数组
2. 识别 **格式信息**

2.1、获取**纠错等级**，2bit

2.2、获取**掩码标示符**，3bit

3、识别 **版本**信息，确定当前版本

4、去掉编码区域的掩模，即从格式信息中得到编码区使用的掩码，然后对编码区进行异或运算，消除掩码

5、根据模块排列规则，识别符号字符，恢复数据码字和纠错码字

6、使用纠错码字进行错误检查，如果发现错误，立即纠错

7、解码数据码字

7.1、获取**模式**指示符

7.2、获取**字数**指示符

7.3、按照所使用的模式进行译码

# 参考文档

Wiki：

<https://en.wikipedia.org/wiki/QR_code>

二维码标准：

<http://www.doc88.com/p-9982811145395.html>

<http://www.doc88.com/p-9813410333607.html>

二维码标准中文：

<http://www.doc88.com/p-743823956683.html>

已经check：

<https://blog.csdn.net/liulina603/article/details/42099121>

<https://blog.csdn.net/u012611878/article/details/53167009>

<https://baike.baidu.com/item/QRCode/10336647?fr=aladdin>

<https://www.cnblogs.com/wangkaining/p/6829327.html>

<https://blog.csdn.net/carson2005/article/details/37519819>

编码流程：

<https://www.jianshu.com/p/3cf1862552f8>

使用python创建二维码：

<https://www.cnblogs.com/sddai/p/5675041.html>

电装的官网：

<https://www.qrcode.com>

格式信息的填充算法BCH

<https://en.wikipedia.org/wiki/BCH_code>

里德-所罗门纠错算法(RS)

<https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction>