

北京交通大学数字图像总结性报告

姓名：程维森 韦舒羽

学号：21231264 21281202

# 摘要

本论文系统地研究了二维码技术的核心内容，深入探讨了二维码的构成要素、生成过程以及识别流程。首先，对二维码的组成进行了详细解析，探讨了其由图案、定位图案、校正图案和封闭图案等元素组成的特征。其次，通过深入分析了二维码生成的流程，包括版本控制、数据编码和纠错等环节，重点介绍了不同参数对二维码生成质量的影响。在识别流程方面，论文详细讨论了二维码识别的基本原理，着重阐述了图像扫描、边缘检测和数据解码等关键步骤。

本文进一步探讨了开源二维码识别库的特点和功能。着重介绍了 ZBar 这一优秀的开源库，对其初始化、图像解析和识别过程进行了深入分析。通过对 ZBar 算法流程的解读，论文呈现了其底层实现原理和算法架构。

针对二维码的创造与识别程序，本文构建了一个基于 Python 的程序框架，通过图形用户界面实现了二维码的生成和识别功能。该程序利用 qrcode 和 pyzbar 库实现二维码的生成和识别，并提供了简洁的用户界面供用户操作。

最后，本文从实际应用的角度探讨了二维码在零售、移动应用、信息获取、物联网、教育和活动等领域的广泛应用。通过这些实例，展示了二维码技术在现实生活中的多样化应用场景。

本论文全面而深入地探讨了二维码技术的各个方面，旨在为读者提供对二维码技术的全面了解，并对其在不同领域的应用提供参考与启发。

# 二维码识别绪论

**2.1. 引言**

二维码（QR code）作为一种广泛应用于数据存储与传输的矩阵条码，已经成为了当今数字化社会中不可或缺的一部分。其具有信息容量大、扫描速度快以及易于识别等优点，在各种领域都得到了广泛应用，例如支付系统、物流跟踪、广告营销等方面。

随着移动设备和数字图像处理技术的快速发展，识别二维码的需求也日益增加。在识别二维码的过程中，采用了多种算法和技术。其中，ZBar 是一种常用的开源库，通过对图像进行扫描，能够识别并解码包含在图像中的二维码信息。

**2.2. ZBar 的工作原理**

ZBar 库是一个基于C语言的工具，具有跨平台性，并且提供了针对不同编程语言的接口，如Python、Java等，方便开发者进行二维码识别的集成。

其工作原理主要包括图像采集、预处理、定位定标、解码和识别等步骤。通过图像采集，ZBar 能够获取待识别的图像数据；随后，进行预处理操作，包括图像灰度化、降噪、边缘检测等操作，以提高二维码的识别率；接着，ZBar 利用图像中的定位标志定位二维码位置，并对其进行解码和识别，最终输出二维码所包含的信息内容。

**2.3. 应用场景与优势**

ZBar 的应用非常广泛，涵盖了多个领域。在商业领域，可以用于移动支付、产品追踪、广告营销等方面；在物流行业，可用于快递单号扫描、库存管理等场景；在生产制造中，可用于产品追溯与质量控制。ZBar 的优势在于其快速的识别速度、准确的解码能力以及稳定的性能表现。

**2.4. 结论**

ZBar 作为一种成熟、稳定的二维码识别工具，为解决现实生活中的二维码识别需求提供了便利。其高效的识别速度和准确性使得其在各种场景下广泛应用。随着图像处理和移动设备技术的不断发展，ZBar 在二维码识别领域有望进一步完善与发展。

# 二维码构成

**3.1 基础知识**

3.1.1. 二维码的大小尺寸

二维码的大小通常由版本（Version）和尺寸确定。版本表示二维码的大小，从版本1开始，每个版本增加4个模块（或称为“单位”）的宽度和高度。版本号越高，存储的数据量越大，但相应地，尺寸也会变大。例如，版本1的尺寸为21x21个模块，而版本40的尺寸为177x177个模块

3.1.2二维码区域的含义

如下图所示，为一个常见的二维码的定义图：

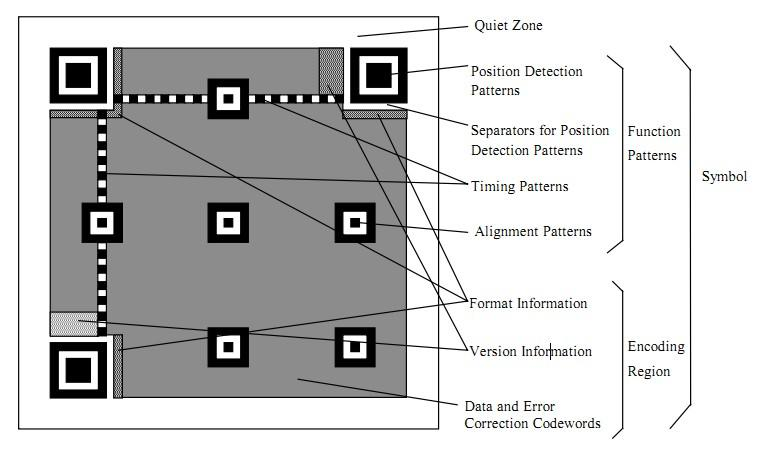


图3.1.1 二维码定义图

详细的介绍如下：

定位图案（Position Detection Patterns）：Position Detection Pattern是用于确定二维码的位置和大小的图案。通常有三个定位图案，位于二维码的角落。这些图案有白色的分隔边界，被称为Separators for Position Detection Patterns。它们三个的存在足以标识整个二维码的矩形边界。

定时图案（Timing Patterns）：Timing Patterns也是用于定位的。随着二维码尺寸的增大（高版本的二维码），需要定时图案来确保扫描设备能够准确读取二维码的网格结构，防止扫描时出现偏移或失真。

对齐图案（Alignment Patterns）：只有Version 2及以上的二维码才需要对齐图案。它们也是用于定位扫描位置的。这些图案的存在有助于更大尺寸的二维码在扫描时准确定位。

功能性数据区域：

格式信息（Format Information）：包含在所有尺寸的二维码中，存储着格式化的数据，如纠错等级和掩模模式等信息。

版本信息（Version Information）：在版本号大于等于7的二维码中，需要预留两个3 x 6大小的区域存放版本相关的信息。

数据码和纠错码：

除了上述区域外，剩余的区域用于存放实际的数据码（Data Code）和纠错码（Error Correction Code）。这些码用于存储用户想要嵌入的数据，以及为了提高二维码的容错性而添加的冗余信息。

3.1.3. 二维码的规范

二维码的规范是指其标准化的制定和遵循。主要的规范包括：

QR Code标准：由国际标准化组织（ISO）和日本工业规格（JIS）共同制定的二维码标准。规定了二维码的格式、尺寸、编码方式、纠错等级等信息。

容错性：二维码标准规定了不同级别的容错性（Error Correction Levels），包括 L、M、Q、H 四个等级，用以指示二维码纠错能力的强弱。高容错级别可以在一定程度上修复损坏或污损的二维码图像。

编码方式：规定了数据编码方式，包括字节编码、数字编码、混合编码等，以适应不同类型的数据。

版本和尺寸：规范了二维码的版本与尺寸范围，确保兼容性和可读性。

特定使用场景的规范：例如微信、支付宝等平台对于二维码的使用会有一些特定的规范，确保二维码在其平台上正常识别和使用。

**3.2 数据编码**

QR码支持多种数据编码方式，包括数字、字符、字节和日文编码。其中，数字编码可包含0至9的数字，字符编码允许数字、大写字母A到Z以及一些特殊符号。字节编码支持ISO-8859-1字符集中的0至255范围内的字符，使得QR码能够存储多种语言的文本和数据。此外，QR码还能处理日文双字节编码。

除了这些基本编码方式外，QR码还支持一些特殊模式。Extended Channel Interpretation (ECI)模式允许使用特殊字符集，但并非所有扫描器都支持。Structured Append模式允许在一个二维码中包含多个部分，每部分可以采用不同的编码格式，以便存储更大量的数据。FNC1模式则适用于特殊的工业或行业需求，类似于GS1条形码，用于区分不同的数据格式。

这些编码方式的组合使得QR码能够适应各种信息存储需求，并在不同领域中发挥重要作用。

下面是二维码的数据编码分类：

数字编码：从0到9；

字符编码：包括 0-9，大写的A到Z（没有小写），以及符号$ % \* + – . / : 包括空格；

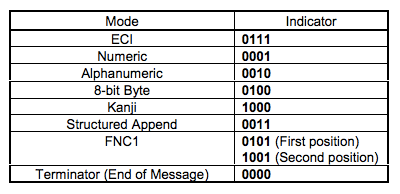
字节编码：可以是0-255的ISO-8859-1字符；

日文编码:也是双字节编码；

Extended ChannelInterpretation (ECI) mode： 主要用于特殊的字符集。并不是所有的扫描器都支持这种编码；

Structured Appendmode： 用于混合编码，也就是说，这个二维码中包含了多种编码格式；

FNC1 mode： 这种编码方式主要是给一些特殊的工业或行业用的。比如GS1条形码之类的

如下表所示：  


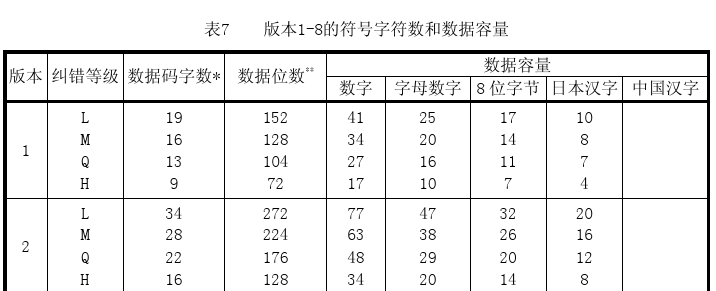
**3.3 结束符和补齐符**

在QR码的编码中，结束符和补齐符是两个特殊的符号，它们用于标识数据的结束和数据块的填充。

结束符：QR码使用结束符来表示数据的结尾。结束符告诉扫描器或解码器已经到达了数据的末尾，这样解码器就知道在这之后不应该再有数据。结束符在QR码中由特定的编码表示，以指示数据的结束位置。

补齐符：在QR码的数据填充中，为了满足特定的编码要求（如数据块大小等），有时候需要在数据末尾添加一些额外的位来填充数据。这些额外的位通常由补齐符填充。补齐符是特殊的数据位模式，用于填充数据块，使其达到特定的长度要求，从而保证QR码的正确解码和读取。

这两个符号都是在QR码的数据编码过程中用于保证数据正确性和完整性的重要组成部分。结束符指示数据的结束位置，而补齐符则用于填充数据块，确保满足编码格式的要求。

下图是结束符和补齐符的官方文档示例：  


**3.4 纠错码**

纠错码（Error Correction Code）是一种在数据传输或存储过程中用来检测和纠正错误的编码方式。在QR码中，纠错码被用来增加数据的可靠性，即使在存在某些损坏或干扰的情况下，也能够恢复原始数据。

在QR码中，纠错码是通过添加额外的数据来实现的。它采用了Reed-Solomon码，这种码能够在一定程度上纠正错误。具体来说：

纠错码的添加：QR码包含了数据区域和纠错码区域。数据区域存储了实际的用户数据，而纠错码区域存储了针对数据区域的纠错码。当创建QR码时，编码器会根据选择的纠错级别（如L、M、Q、H等级别）来确定添加的纠错码的数量。而下图则为二维码的纠错特性：



纠错级别：QR码有四个纠错级别，分别是 L、M、Q 和 H。不同级别的纠错能力不同，H级别的纠错能力最强。选择更高级别的纠错意味着在一定程度上可以纠正更多的错误，但也会占用更多的存储空间。

纠错码的作用：当QR码受到损坏、污损或部分遮挡时，解码器可以利用纠错码区域的信息来检测和修复受损的数据。纠错码允许解码器在接收到的数据中检测到错误并尝试对其进行修复，以恢复原始数据。

Reed-Solomon码的特性：Reed-Solomon码具有强大的纠错能力，即使在数据受损的情况下也能够恢复数据。它不仅可以发现错误，还能够纠正一定数量的错误。这种码的效率取决于添加的纠错码数量和QR码的整体大小。具体的纠错码特性如下表所示：



总体而言，纠错码在QR码中起着重要的作用，使得即使在有些数据受损或丢失的情况下，QR码仍能够被正确读取和解码，提高了数据的可靠性和稳定性。

**3.5 最终编码**

二维码的最终编码过程是一个将数据转换为可视化的二维矩阵的步骤。这个过程包括了数据分割、数据编码、纠错码生成和最终布局等步骤。

数据分割：首先，将要编码的数据按照QR码的容量限制和格式要求进行分割。根据QR码的版本（尺寸）、数据类型和纠错级别等因素，将数据分成适当大小的数据块。

数据编码：将分割后的数据块转换成二进制位序列，并将其映射到QR码支持的不同编码方式中，比如数字编码、字符编码、字节编码等。不同编码方式使用不同的编码规则将数据转换为QR码可以识别的格式。

纠错码生成：对于每个数据块，计算并添加相应的纠错码。使用Reed-Solomon编码生成纠错码，这些码将与数据一起编码到QR码中，提高了QR码的容错能力，使其在一定程度上能够纠正和恢复受损的数据。

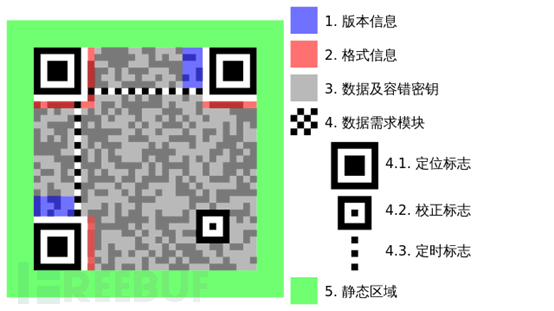
布局和排列：将数据块和纠错码块按照特定的布局算法排列成二维矩阵。这个过程包括了数据块的位置定位、对齐图案的添加、数据与纠错码的交错排列等步骤。QR码的版本和级别会影响最终二维矩阵的大小和布局。

添加特殊图案：在二维矩阵中加入定位图案、定时图案、格式信息、版本信息等特殊图案。这些图案用于帮助扫描设备定位和解码QR码。

生成最终的二维码：将排列好的二维矩阵转换为可视化的QR码图像。这个图像可以是黑白方块的矩阵，其中黑色方块代表数据，白色方块代表空白区域，形成了最终的二维码图案。

整个过程需要遵循QR码标准的规定和编码算法，确保生成的二维码能够被各种扫描设备准确读取和解码。

**3.6 二维码图分析**

二维码的格式示例如下：  


**有如下的几个特征点：**

Position Detection Pattern：

Position Detection Pattern（位置检测图案）是二维码（如QR码）中的一种特殊图案，用于帮助扫描设备确定二维码的位置和方向。这些图案通常以特定的规则和模式在二维码中标识出来，以便扫描设备能够快速、准确地识别和定位二维码的边界和位置。

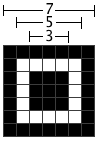
在QR码中，Position Detection Patterns 是由三个相同的方块图案组成的大正方形，分别位于二维码的左上角、右上角和左下角。这些图案用于帮助扫描设备识别二维码的位置和角度，以便进行准确的扫描和解码。

Position Detection Pattern的作用包括：

定位二维码位置：这些图案的存在和特殊的布局有助于扫描设备迅速识别二维码的位置，从而准确定位二维码的边界。

确定二维码的方向：通过检测这些特殊图案的布局和相对位置，扫描设备可以确定二维码的方向，确保在解码过程中数据的正确读取。

提高识别准确性：位置检测图案的存在有助于避免扫描过程中出现误差，提高了解码的准确性和稳定性。



Position Detection Pattern图示

Alignment Pattern：

Alignment Patterns（对齐图案）是QR码或其他二维码中的特殊图案，旨在帮助扫描设备在不同距离和角度下更准确地识别和解码二维码。这些图案通常位于QR码的内部，不同于定位图案（Position Detection Patterns），其作用更多是为了提高扫描设备在扫描大型或存在扭曲变形的二维码时的准确性。

Alignment Patterns通常以小的黑白圆点或方块的形式存在，由一系列同心圆圈组成。它们分布在二维码的不同区域，通常在定位图案的附近或之间。对齐图案的作用主要包括以下几点：

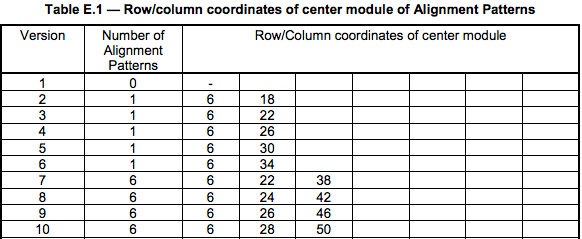
提高扫描精度：当二维码被放大、缩小、旋转或存在畸变时，对齐图案可以帮助扫描设备更精确地确定二维码的位置和方向，从而提高扫描的准确性。

指示二维码版本：QR码有不同版本，每个版本都具有不同数量和位置的对齐图案。通过检测和识别对齐图案，扫描设备可以确定二维码的版本，以便进行适当的解码处理。

优化解码算法：对齐图案提供了可靠的参考点，帮助解码算法更有效地处理二维码中的数据，提高解码的效率和准确性。



Alignment Pattern图示



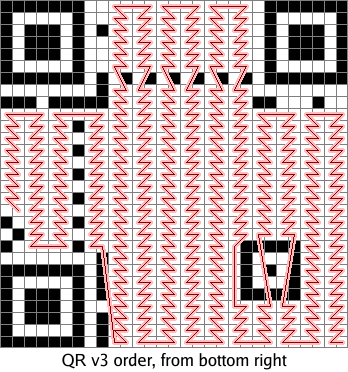
Alignment Pattern定义表

数据和数据纠错码

在 QR 码或其他类似的二维码中，数据和数据纠错码是编码和存储在码中的两个重要组成部分。这两者协同工作，以确保二维码数据的完整性和可靠性。

数据：这是要在二维码中嵌入的实际信息，可以是文本、链接、数字等。QR 码通过不同的编码方式（数字、字符、字节等）来表示这些数据。数据可以被分为数据块，并根据 QR 码的容量限制进行划分和编码。

数据纠错码：为了提高二维码的容错能力，对数据添加了纠错码。在 QR 码中，使用的纠错码通常是 Reed-Solomon 算法生成的。这些纠错码被添加到数据块中，用于检测和纠正二维码数据中的错误。



掩码图案：

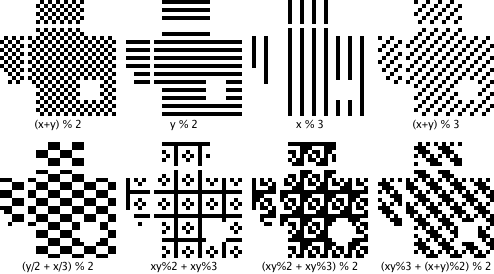
在 QR 码中，掩码图案（Mask Pattern）是一种用于增加二维码扫描可靠性的技术。掩码图案通过对二维码模块中的数据模式进行特定规则的处理，使得生成的二维码更具鲁棒性。

掩码图案的主要目的是改变二维码中数据模块的布局，使得扫描设备在解码时更容易区分模块之间的信息，从而提高扫描准确性。掩码图案通过以下方式应用于数据模块：

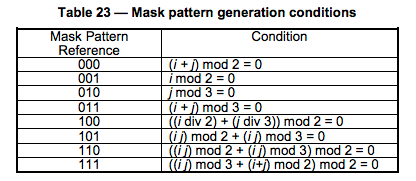
数据模块反转：掩码图案根据特定规则对二维码中的数据模块进行反转，即将一些特定模块的黑白颜色进行反转。这种反转操作有助于减少相邻模块之间的相似性，使得扫描设备更容易识别和处理二维码。

减少视觉相似性：掩码图案的应用可以使得二维码的视觉模式更加均匀、减少重复规律。这样做有助于防止扫描设备因为模块之间太过相似而产生混淆，提高了解码的准确性。

在生成 QR 码时，可以选择不同的掩码图案规则。QR 码的生成器会尝试不同的掩码图案规则，并评估生成的二维码对扫描的可靠性，然后选择最优的掩码图案。这种方法使得二维码在不同环境下的可扫描性更强。

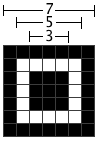


掩码图案



# 二维码生成流程

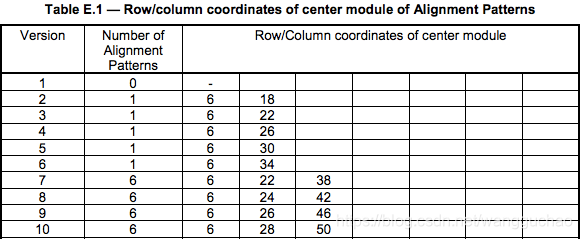
首先画出Position Detection图案，在二维码的三个角上

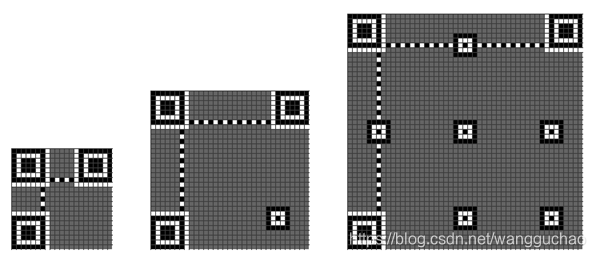


其次画出Alignment图案

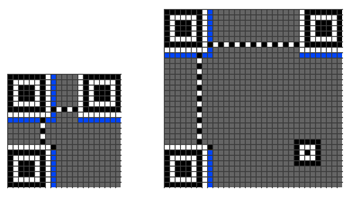


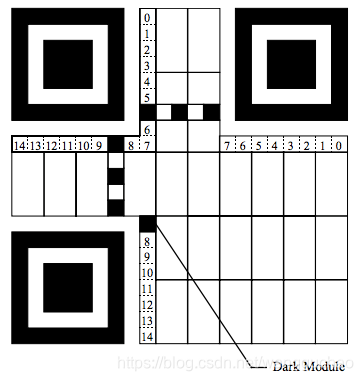
关于Alignment的位置，可以查看QR Code Spec的第81页的Table-E.1的定义表



接下来是Timing Pattern的线

再接下来是Formation Information，下图中的蓝色部分

  
Format Information是一个15个bits的信息，每一个bit的位置如下图所示：



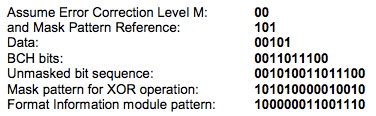
这15个bits中包括：

• 5个数据bits：其中，2个bits用于表示使用什么样的Error Correction Level， 3个bits表示使用什么样的Mask

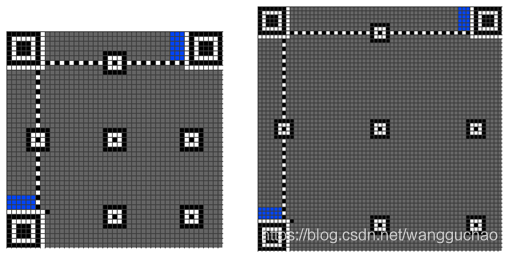
• 10个纠错bits。主要通过BCH Code来计算

然后15个bits还要与101010000010010做XOR操作。这样就保证不会因为我们选用了00的纠错级别和000的Mask，从而造成全部为白色，这会增加我们的扫描器的图像识别的困难。

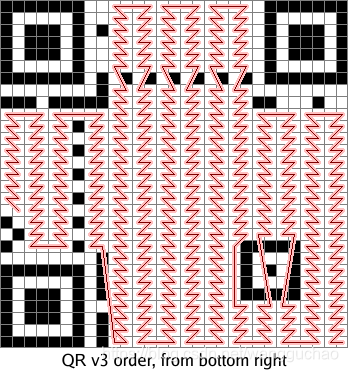
下面是一个示例：



Version Information  
再接下来是Version Information（版本7以后需要这个编码），下图中的蓝色部分。

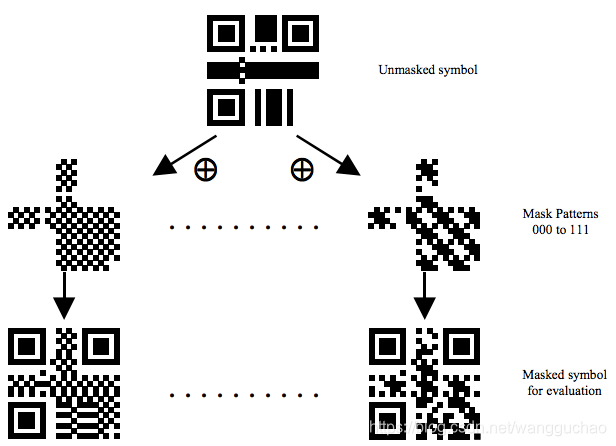


 然后是填接我们的最终编码，最终编码的填充方式如下：从左下角开始沿着红线填我们的各个bits，1是黑色，0是白色。如果遇到了上面的非数据区，则绕开或跳过。

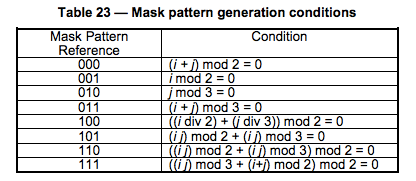


这样图就填好了，但是，也许那些点并不均衡，如果出现大面积的空白或黑块，会告诉我们扫描识别的困难。所以，我们还要做Masking操作,QR的Spec中说了，QR有8个Mask你可以使用，如下所示：其中，各个mask的公式在各个图下面。所谓mask，说白了，就是和上面生成的图做XOR操作。Mask只会和数据区进行XOR，不会影响功能区。

下面是Mask后的一些样子

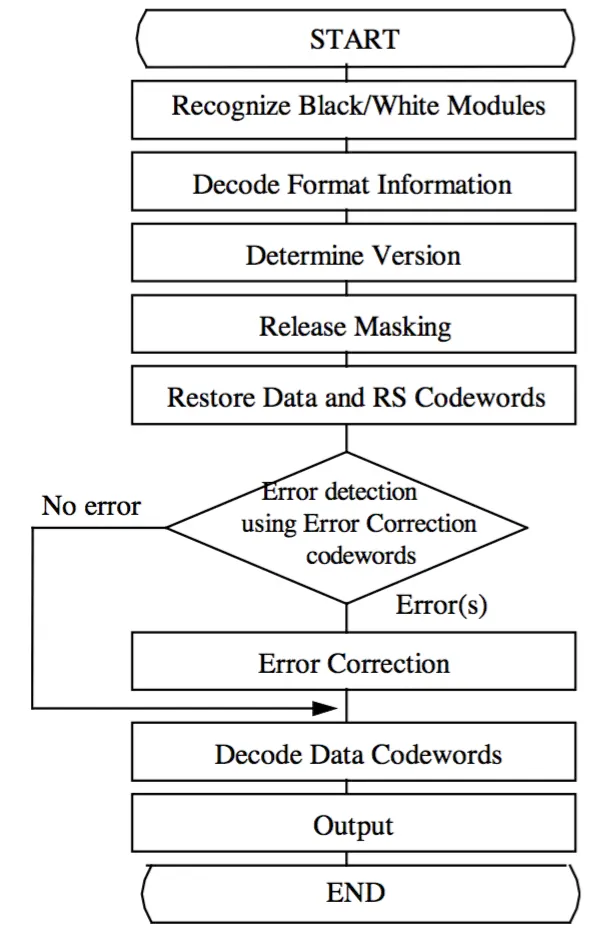


其Mask的标识码如下所示：



# 二维码识别流程

二维码的识别流程就是译码的流程，也就是编码的反过程，下图介绍了译码的流程：



译码流程

大概的流程可以分为八步：

定位并获取符号图像。深色与浅色模块识别为“ 0”与“1”的阵列。

识读格式信息（如果需要，去除掩模图形并完成对格式信息模块的纠错，识别纠错等级与掩模图形参考。）

识读版本信息，确定符号的版本。

用掩模图形掩模图形参考已经从格式信息中得出对编码区的位图进行异或处理消除掩模。

根据模块排列规则，识读符号字符，恢复信息的数据与纠错码字。

用与纠错级别信息相对应的纠错码字检测错误，如果发现错误，立即纠错。

根据模式指示符和字符计数指示符将数据码字划分成多个部分。

最后，按照使用的模式译码得出数据字符并输出结果。

# 开源二维码识别库

**6.1综述**

开源二维码识别库是一种允许开发人员在其应用程序中集成二维码扫描和识别功能的工具库。这些库提供了一组功能和算法，可用于检测、解码和识别二维码图像，使开发人员能够轻松地在他们的应用程序中实现二维码识别功能。

这些开源库通常提供简单易用的API，可以在各种编程语言和平台上使用，如Python、JavaScript、Java、C++等。它们可能基于不同的算法和技术，包括但不限于基于图像处理、机器学习或计算机视觉的方法，以确保对不同类型和质量的二维码图像进行准确识别。

一些流行的开源二维码识别库包括ZBar、ZXing (Zebra Crossing)、OpenCV、Quirc等。这些库通常提供了广泛的文档和示例，使开发人员能够快速集成和定制二维码识别功能，以满足他们应用程序的需求。

**6.2Zbar开源识别库**

ZBar 是一个流行的开源条码（包括二维码）识别库，旨在为开发人员提供跨平台、多语言的条码识别功能。它能够在不同操作系统上运行，包括Linux、Windows和macOS，并支持多种编程语言，例如C、Python和Java。

以下是关于 ZBar 开源识别库的详细介绍：

特点和功能：

1. 多格式支持：

ZBar 支持多种条码格式的识别，其中包括了二维码（如 QR 码）、Data Matrix、PDF417 等等。

2. 跨平台性：

ZBar 具有良好的跨平台性，能够在多个操作系统上运行，这使得开发者可以在不同环境中使用它。

3. 多语言支持：

ZBar 提供了多种编程语言的接口和库，包括 C、Python、Java 等，方便开发者根据自身需求选择合适的语言进行开发。

4. 高效识别：

它具有高效的识别能力，能够处理不同角度、尺寸和质量的图像，快速准确地识别出条码信息。

5. 简单易用的 API：

ZBar 提供了简单易用的 API 接口，使得开发者能够轻松地在自己的应用程序中实现条码扫描和识别功能。

工作原理：

ZBar 的工作原理大致分为以下几个步骤：

图像获取：

首先，ZBar 获取图像数据或视频流，这可以是通过摄像头采集的图像或是已存在的图像文件。

预处理：

对获取到的图像进行预处理操作，如灰度化、降噪等，以准备进行条码识别的进一步处理。

条码定位：

通过边缘检测等技术，ZBar 定位可能包含条码的区域。

条码解码：

使用特定的解码算法对定位到的条码区域进行解码，提取其中的信息内容。

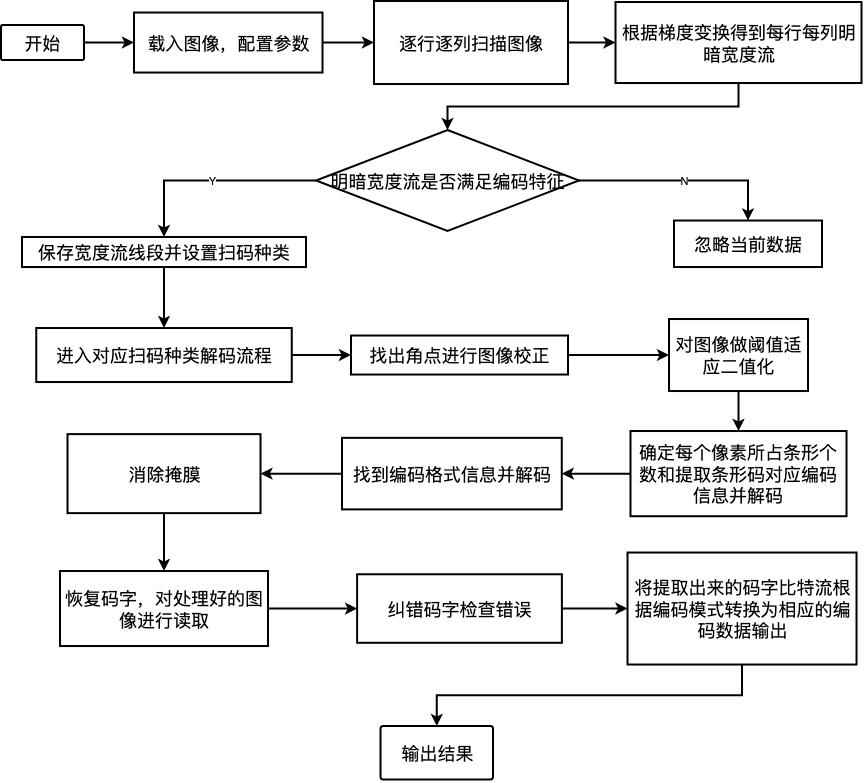
信息处理：

对解码得到的信息进行处理，输出识别到的条码内容或执行相关操作。

在下文中，将会对Zbar算法的流程做详细的介绍。

**6.3Zbar算法讲解**

以下的流程框图详细介绍了Zbar算法的识别与分析流程：



图Zbar识别的流程

算法流程介绍：

1. ImageScanner scanner;
2. *// configure the reader*
3. scanner.set\_config(ZBAR\_NONE, ZBAR\_CFG\_ENABLE, 1);

首先，算法进行初始化，创建了一个名为 ImageScanner 的扫描器对象，并使用 set\_config() 方法对扫描器进行初始化设置，确保扫描器能够启用识别功能。

1. IplImage \*img = cvLoadImage("./test.png");
2. IplImage \*imgGray = cvCreateImage(cvGetSize(img), 8, 1);
3. cvCvtColor(img, imgGray, CV\_RGB2GRAY);

接下来，载入图像的过程展示了使用 OpenCV 库读取图像文件并将其转换为灰度图像的步骤。通过 cvLoadImage() 读取图片文件，然后利用 cvCreateImage() 创建灰度图像，并通过 cvCvtColor() 实现 RGB 到灰度图像的转换。

1. int width = imgGray->widthStep;
2. int height = imgGray->height;
3. Image image(width, height, "Y800", imgGray->imageData, width \* height);

构建了一个名为 Image 的图像对象，并调用其构造函数对其进行初始化。这一步骤中，获取了图像的宽度、高度、像素格式以及像素数据。

1. while(y < h) {
2. iscn->dx = iscn->du = 1;
3. iscn->umin = 0;
4. while(x < w) {
5. uint8\_t d = \*p;
6. movedelta(1, 0);
7. zbar\_scan\_y(scn, d);
8. }
9. quiet\_border(iscn);
10. movedelta(-1, density);
11. iscn->v = y;
12. if(y >= h)
13. break;
14. iscn->dx = iscn->du = -1;
15. iscn->umin = w;
16. while(x >= 0) {
17. uint8\_t d = \*p;
18. movedelta(-1, 0);
19. zbar\_scan\_y(scn, d);
20. }
21. ASSERT\_POS;
22. quiet\_border(iscn);
23. movedelta(1, density);
24. iscn->v = y;
25. }

随后的图像解析过程调用了图像扫描器对象的 scan() 方法，对图像对象进行处理。在内部，scan() 方法主要调用了 zbar\_scan\_image() 函数，该函数首先对传入的图像进行配置校验，然后按照 Z 字型路径逐行扫描传入的图像。

1. #ifdef ENABLE\_EAN
2. if((dcode->ean.enable) &&
3. (sym = \_zbar\_decode\_ean(dcode)))
4. dcode->type = sym;
5. #endif
6. #ifdef ENABLE\_CODE39
7. if(TEST\_CFG(dcode->code39.config, ZBAR\_CFG\_ENABLE) &&
8. (sym = \_zbar\_decode\_code39(dcode)) > ZBAR\_PARTIAL)
9. {
10. dcode->type = sym;
11. }
12. #endif
13. #ifdef ENABLE\_CODE128
14. if(TEST\_CFG(dcode->code128.config, ZBAR\_CFG\_ENABLE) &&
15. (sym = \_zbar\_decode\_code128(dcode)) > ZBAR\_PARTIAL)
16. dcode->type = sym;
17. #endif
18. #ifdef ENABLE\_I25
19. if(TEST\_CFG(dcode->i25.config, ZBAR\_CFG\_ENABLE) &&
20. (sym = \_zbar\_decode\_i25(dcode)) > ZBAR\_PARTIAL)
21. dcode->type = sym;
22. #endif
23. #ifdef ENABLE\_PDF417
24. if(TEST\_CFG(dcode->pdf417.config, ZBAR\_CFG\_ENABLE) &&
25. (sym = \_zbar\_decode\_pdf417(dcode)) > ZBAR\_PARTIAL)
26. dcode->type = sym;
27. #endif
28. #ifdef ENABLE\_QRCODE
29. if(TEST\_CFG(dcode->qrf.config, ZBAR\_CFG\_ENABLE) &&
30. (sym = \_zbar\_find\_qr(dcode)) > ZBAR\_PARTIAL)
31. dcode->type = sym;
32. #endif

接着，对扫描器结构变量 scn 中保存的明暗宽度流进行处理，调用了 zbar\_decode\_width(scn->decoder, scn->width) 函数。这个函数主要处理当前行目前保存下来的宽度流，通过计算各宽度之间的宽度信息来提取扫码特征。依次进行了一维码和二维码的检测标准，寻找符合标准的扫码种类，并更新了扫描器结构变量 scn 中的 type 成员，以及更新了 lock 成员以增加当前种类判断的置信度。针对不同的条码类型（如 EAN、Code39、Code128 等），进行了特定的解码方法。

以 QR 码为例，函数 \_zbar\_find\_qr(dcode) 内部对当前行的宽度流进行计算，判断其是否符合特定的特征。如果符合特征，则判定其不是 QR 码；否则将当前宽度流描述为一个自定义的线段结构，并将满足条件的横向线段结构变量存入名为 lines 的横向线段集合中。

整幅图像的逐列扫描同逐行扫描一样，扫描路径为 N 字型，通过函数 zbar\_scan\_y() 和 process\_edge() 进行处理以找到边缘，并最终求取出纵向的明暗高度流。通过 zbar\_decode\_width(scn->decoder, scn->width) 函数处理后，将符合 QR 码特征的纵向线段存入 lines 的纵向线段集合中。

1. qr\_finder\_t \*qrf = &dcode->qrf;
2. qrf->s5 -= get\_width(dcode, 6);
3. qrf->s5 += get\_width(dcode, 1);
4. unsigned s = qrf->s5;
5. if(get\_color(dcode) != ZBAR\_SPACE || s < 7)
6. return ZBAR\_NONE;
7. int ei = decode\_e(pair\_width(dcode, 1), s, 7);
8. if(ei)
9. goto invalid;
10. ei = decode\_e(pair\_width(dcode, 2), s, 7);
11. if(ei != 2)
12. goto invalid;
13. ei = decode\_e(pair\_width(dcode, 3), s, 7);
14. if(ei != 2)
15. goto invalid;
16. ei = decode\_e(pair\_width(dcode, 4), s, 7);
17. if(ei)
18. goto invalid;
19. invalid:
20. return ZBAR\_NONE;

以 QR 码为例子，函数\_zbar\_find\_qr(dcode)内部对当前行的宽度流进行计算，判断是否符合以上特征

1. int nqrdata = 0;
2. qr\_finder\_edge\_pt \*edge\_pts = NULL;
3. qr\_finder\_center \*centers = NULL;
4. if(reader->finder\_lines[0].nlines < 9 ||
5. reader->finder\_lines[1].nlines < 9)
6. return(0);
7. int ncenters = qr\_finder\_centers\_locate(¢ers, &edge\_pts, reader, 0, 0);
8. if(ncenters >= 3) {
9. void \*bin = qr\_binarize((unsigned char\*)img->data, img->width, img->height);
10. qr\_code\_data\_list qrlist;
11. qr\_code\_data\_list\_init(&qrlist);
12. qr\_reader\_match\_centers(reader, &qrlist, centers, ncenters,
13. (unsigned char\*)bin, img->width, img->height);
14. if(qrlist.nqrdata > 0)
15. nqrdata = qr\_code\_data\_list\_extract\_text(&qrlist, iscn, img);
16. qr\_code\_data\_list\_clear(&qrlist);
17. free(bin);
18. }
19. if(centers)
20. free(centers);
21. if(edge\_pts)
22. free(edge\_pts);
23. return(nqrdata);

符合当前特征的即判断其不为 QR 码，如果不符合，将当前宽度流描述为一个自定义的线段结构，包含两端端点及长度等信息，并将满足条件的横向线段结构变量存入一个容器lines的横向线段集合中。 对整幅图像的逐列扫描同逐行扫描一样，扫描路径为 N 字型，同样通过函数zbar\_scan\_y()和process\_edge()进行处理找边缘最后求取出纵向的明暗高度流，通过zbar\_decode\_width(scn->decoder, scn->width)函数进行处理，将符合 QR 码的纵向线段存入lines的纵向线段集合中。

ZBar 开源条码识别库的流程是一个多阶段的图像扫描和解析过程。其核心算法涉及图像初始化、灰度处理、边缘检测、宽度流解析以及特定条码类型的识别。

该算法首先通过初始化扫描器对象并加载图像，然后进行灰度处理和边缘检测。在确定边缘后，进行宽度流解析，提取扫码特征，并依据特定的一维码和二维码检测标准，对各种条码类型进行识别。例如，对于QR码，会检查图像中的特定特征以判断其是否为QR码。

整个流程包含了对图像进行不同路径的扫描，并通过对图像的逐行和逐列扫描，收集明暗宽度流，最终提取出符合特定条码特征的线段集合。

ZBar算法的关键在于其对边缘的处理、宽度流的解析，以及对不同类型条码的特征识别。这个流程有助于准确地识别图像中的各种条码类型，为后续的解码提供了重要的特征信息。

# 二维码的创造与识别程序详解

二维码的生成与识别是现代信息技术中广泛应用的重要环节。以下是一个使用 Python 编写的基于 tkinter、PIL、qrcode 和 pyzbar 库的简易二维码生成与识别程序。该程序提供了一个用户友好的 GUI 界面，允许用户输入数据生成二维码，并可以通过图像识别功能解码二维码中的信息。

7.1. 二维码生成

生成二维码功能允许用户输入数据，并生成相应的二维码图像。程序利用 qrcode 库创建了一个 QRCode 对象，并通过设定版本、纠错级别、盒子大小和边框等参数生成二维码图像。生成的图像以 Image 对象的形式呈现在 GUI 界面上。

1. def generate\_qr\_code():
2. # 获取用户输入的数据
3. data = entry.get(）
4. if data:
5. # 创建QRCode对象并生成二维码图像
6. qr = qrcode.QRCode(
7. version=1,
8. error\_correction=qrcode.constants.ERROR\_CORRECT\_H,
9. box\_size=10,
10. border=4,
11. )
12. qr.add\_data(data)
13. qr.make(fit=True)
14. img = qr.make\_image(fill\_color="black", back\_color="white")
15. img = img.resize((200, 200), Image.ANTIALIAS)
16. img\_tk = ImageTk.PhotoImage(img)
17. qr\_label.config(image=img\_tk)
18. qr\_label.image = img\_tk
19. status\_label.config(text="QR Code generated")
20. return img

7.2. 二维码识别

二维码识别功能允许用户从生成的二维码图像中解码信息。程序使用 pyzbar 库对图像中的二维码进行解码，并将解码结果显示在 GUI 界面上。

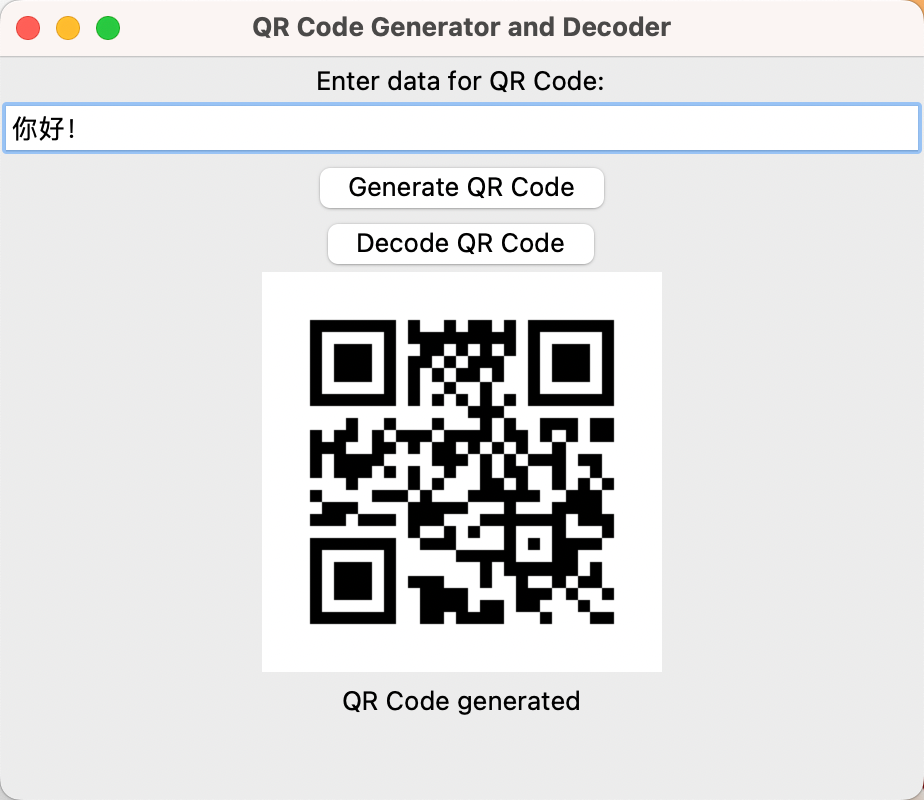
1. def decode\_qr\_code\_from\_img(img):
2. # 将图像保存为二进制流
3. buffered = io.BytesIO()
4. img.save(buffered, format="PNG")
5. buffered.seek(0)
6. # 解码二维码
7. decoded\_objects = decode(Image.open(buffered))
8. if decoded\_objects:
9. decoded\_text = decoded\_objects[0].data.decode('utf-8', 'ignore')
10. return decoded\_text
11. return None

7.3. GUI界面

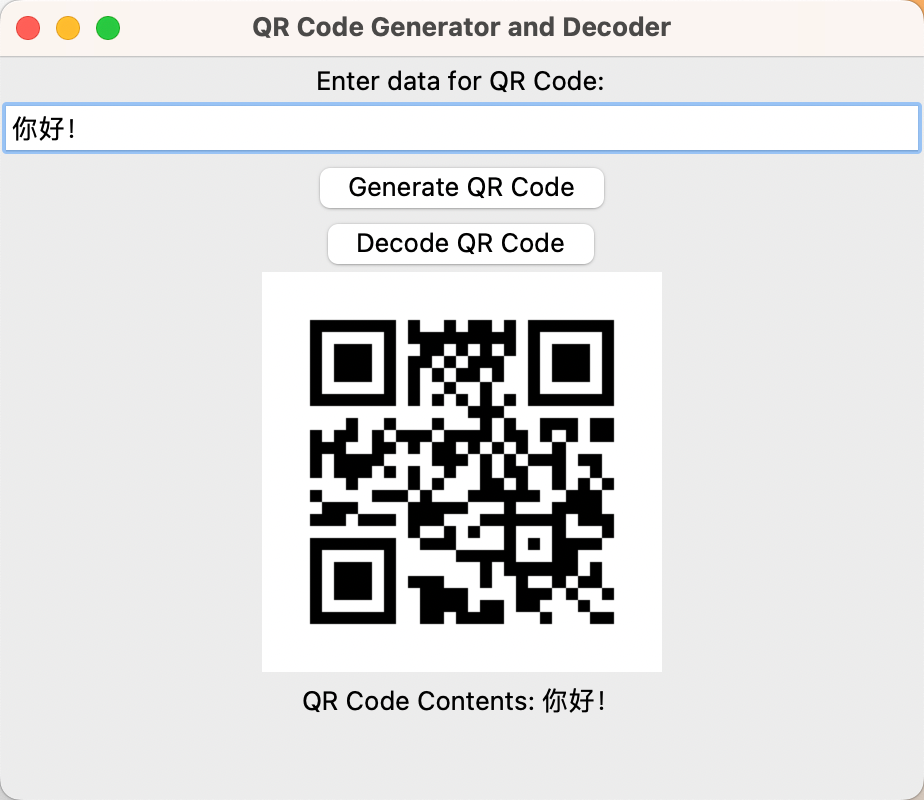
程序使用 tkinter 构建了一个简单的 GUI 界面，包括输入框、生成二维码按钮、解码二维码按钮、二维码图像显示区域以及状态信息显示区域。用户可以在输入框中输入数据，点击相应按钮进行二维码生成和识别操作。

以上是该程序的基本功能和实现原理，通过简单易懂的界面，用户可以方便地进行二维码的生成和识别操作。

下面是这个程序的运行输出示例：



产生二维码



解析二维码

使用手机扫码后，得到了同样的结果，证明该二维码确实可以被扫描。



# 实际应用

ZBar是一个强大的开源条码扫描和识别库，其灵活性和多功能性使得它在各种实际应用中被广泛采用。以下是ZBar实际应用的几个领域：

8.1. 零售与商业

收银系统和库存管理： 在零售行业中，商家可以使用ZBar库来构建收银系统，帮助快速扫描商品条码以完成结账流程，并自动更新库存信息。

商品跟踪和物流： 用于跟踪和管理商品、扫描包装标签，加速仓库管理和物流流程。

8.2. 移动应用与扫码支付

移动应用集成： 许多移动应用集成了ZBar功能，例如扫码支付应用、票务预订应用等，用户可以利用摄像头扫描二维码完成支付或获取信息。

活动门票与抽奖： 活动中使用二维码作为门票或抽奖凭证，并通过扫描识别来管理参与者或领取奖品。

8.3. 信息获取和访问控制

身份识别与考勤系统： 在企业内部使用二维码作为员工身份识别标识，或者作为考勤系统的一部分来记录员工工作时间。

链接访问： 在广告、宣传品或文化场所中，提供二维码以便快速获取相关信息或访问网页链接。

8.4. 物联网与智能设备

智能家居和物联网设备： 在智能家居控制中，二维码用于快速添加设备、配置网络等，简化设备的安装和连接流程。

生产追溯与产品信息： 生产过程中使用条码标识物品，利用ZBar识别追踪产品生产流程或获取产品信息。

8.5. 教育和活动

教育应用： 在教育领域，老师可以使用二维码分享课件、作业等信息，学生可以快速扫描获取相关资料。

会议与活动管理： 在会议或活动中，通过二维码进行签到、获取议程或相关资料。

ZBar的应用领域非常广泛，它提供了一个便捷的方式来处理和识别条形码和二维码，有效地应用于商业、科技、教育和生活等多个领域。

# 分析总结

论文全面剖析了二维码技术的核心方面，从构成要素到生成流程再到识别流程，以及开源库的解析和实际应用，都进行了详实的探讨。以下是对论文内容的分析总结：

1. 全面性和深入性： 论文对二维码技术进行了全面而深入的研究，包括了二维码的构成要素、生成流程和识别流程，为读者提供了系统性的知识框架。

2. 技术细节的展示： 论文针对二维码生成和识别过程中的关键环节，展示了大量的技术细节和算法原理，例如数据编码、纠错编码、图像扫描和边缘检测等，有助于读者深入理解二维码技术的实现原理。

3. 开源库的解析： 对于开源库 ZBar 的解析，论文深入剖析了其初始化、图像解析和识别流程，为读者呈现了一个典型的二维码识别库的实现方式和内部工作原理。

4. 程序实践与应用案例： 论文不仅在理论层面阐述了二维码技术，还通过实践构建了一个基于 Python 的二维码生成与识别程序。通过具体的应用案例，为读者提供了实际操作的参考，使得理论知识能够转化为实际应用能力。

5. 实际应用的探讨： 论文对二维码在多个领域的实际应用进行了广泛探讨，从商业应用到教育领域，从移动应用到物联网，展示了二维码技术在现实生活中的多种实际应用场景。

综上所述，论文内容详实全面，深入浅出地介绍了二维码技术的方方面面，不仅有助于读者全面理解二维码技术的原理与实现，也为读者提供了在实际应用中探索和运用二维码技术的指导和参考。