摘要

全国代码管理中心为了提高代码信息化服务水平，提高代码业务处理效率，需要建立一套新的组织机构代码业务信息采集系统。在目前仍然使用的旧系统当中，组织机构信息的识别、录入、管理依然是人工方式，存在着效率低下，人力成本高，容易出错的问题，也存在着信息易泄漏、易伪造的安全隐患。

本文结合这个项目的实际需求，对该系统中的组织机构代码自动识别子系统进行了设计和实现，旨在解决旧系统中上述的种种问题。本文对二维码的编码解码技术进行了探究和分析，对其中的中文识别存在的问题进行了解决，以此来实现高效的组织机构信息的识别、录入；并对文本加密进行了探究，选择了几种有代表性的算法进行了分析，结合他们的优点缺点，尝试对其进行了结合和改进，以此来解决信息易泄漏、易伪造的安全隐患；对于加密后出现的新问题——本文长度太长，不利于二维码的生成和解码，本文为此也研究了字符串压缩，并作相应结合，以解决二维码容量有限的问题；为了实现高效信息管理目的，本文也对二维码批量生成、搜索功能进行了实现，为此，本文对文本多串模糊匹配，多线程同步、异步操作等相关算法和技术进行了探究，并作相应实现和结合。

该子系统在完成后，通过使用二维码存储信息的方式达到了组织机构信息高效高质量识别、录入的目的；通过加密实现了关键信息需要安全隐藏、不能被伪造的功能；虽然在加密后出现了文本长度过长的问题，但通过文本压缩的方式得到了解决；对于二维码信息的管理，该系统对组织机构的多项信息进行格式化，实现了快速的批量生成、搜索功能，为海量信息的管理提供了接口。

关键字：自动识别，二维码，加密，文本压缩，多串匹配

Abstract

National Code Management Centre in order to improve code information service levels, improve business efficiency code, need to establish a new organization code of business information collection systems.

Combined with the actual needs of the project, the organization code systems automatically identify subsystem design and implementation, aimed at resolving the old system to all these problems. In this paper, two-dimensional code codec technologies were explored and analyzed, on which the Chinese recognize the existence of problems were solved in order to achieve recognition efficient organization of information entry; were explored and text encryption, select several representative algorithm analysis, combined with their merits and demerits, and try to be a combination of improvements in order to solve the information easy to leak, easy to counterfeit security risks; new problems arise for encrypted - This paper length is too long, is not conducive to the two-dimensional code generation and decoding, the paper also studied this purpose string compression, with a corresponding combined to solve the problem of the limited capacity of the two-dimensional code; efficient information management in order to achieve the purpose, the paper also batch generate two-dimensional code, the search function has been achieved,this paper more than the text string fuzzy matching, multi-thread synchronization, algorithms and techniques were explored asynchronous operation, and accordingly implement and combine.

Upon completion of this subsystem by using two-dimensional code information stored way to achieve the organization information to identify high- quality, entry purposes; through encryption key information needed to achieve a safe hiding, cannot be forged function; Although encrypted the text is too long there is a problem, but by way of text compression has been resolved; dimensional code information for the management of the system to a number of formatting information organization to achieve a quick batch generation, search functions, to manage the vast amounts of information provides the interface.

Key word： Two-dimensional code, encryption, text compression, string matching.

第 1 章 绪论

1.1 研究背景与意义

全国代码管理中心目前有一套在使用的系统，现有系统于 2004 年建成，在该系统的使用过程中，建立并完善了各项业务流程及工作模式，对全国组织机构代码的收集、上报起到了极大的作用。随着 IT 技术的发展，出现了很多新的信息采集 、管理、应用技术，如何将这些新技术与代码业务进行结合，提高代码信息化服务水平，提高代码业务处理效率，是当前信息化环境下对代码技术发展提出的新的要求。通过本项目，探索新技术可能在信息采集过程中的应用，使代码采集工作更便捷、准确。

自动识别技术的信息是以特定信息载体的形式来传输，识别方式的不同，载体的形式也不相同。物品信息载体具有一般信息载体(信息传播中携带信息的媒介)的基本属性，同时也具有区别于一般信息载体的特性，例如容量相对较小、体积小成本低和易于识读等，正式因为具有了这些特性，物体信息载体主要用于存储对容量要求较低对、识别速度要求高的物品信息。常见的物体信息载体有：条码(Bar Code)、射频标签(RFID)、磁条卡智能卡(IC 卡)等，其中条码又可以分为一维码和二维码。

二维码[1]是指在一维条码的基础上扩展出另一维具有可读性的条码，使用黑白矩形图案表示二进制数据，被设备扫描后可获取其中所包含的信息。一维条码的宽度记载着数据，而其长度没有记载数据。二维条码的长度、宽度均记载着数据。二维条码有一维条码没有的“定位点”和“容错机制”。容错机制在即使没有辨识到全部的条码、或是说条码有污损时，也可以正确地还原条码上的资讯。二维条码的种类很多，不同的机构开发出的二维条码具有不同的结构以及编写、读取方法。二维条码通常有三个定位点，这三个定位点提供读码机辨识。因为有这些定位点，所以二维条码不管是从何种方向读取都可以被辨识。二维条码比一维条码记载数据量更多，而且可以记载更复杂的数据，比如图片、网络链接等。

随着二维码的迅速普及，从最开始用于信息存储、传递和识别技术，到现在用于网银、手机支付，伴随而来的安全隐患也日益增多。由于二维码内存储的是明文，如果被不法分子截获，便可以完全识别其中的内容。这将造成以下

的后果。

第一，不法分子可以将二维码用于捆绑病毒、伪造支付信息，进行钓鱼诈骗等，使得用户在扫码后遭受巨大的损失。

第二，二维码现在也用于身份检测，例如微信的网页登录方式，就需要用户通过扫描指定二维码的方式来验证用户身份的合法性。而这个二维码若被不法分子截获，则其可被用于伪造身份登录，进而造成用户隐私的泄漏，甚至可以用微信的支付功能使得用户“替他人买单”、造成经济损失。

第三，二维码除了实现开放式传播信息的功能外，也有一种需求是需要实现有针对性的传播。例如组织机构代码管理中心将启用采用了二维码的组织机构代码证，这个二维码除了实现便于企业信息录入的功能外，也要用于组织机构代码证的防伪检测。但由于二维码记录的是明文，使得该二维码也可以轻易的被伪造，从而无法起到防伪的作用了。

这些隐患，都使得一种更加安全，更加有针对性传播的二维码呼之欲出，本文将针对组织机构代码二维码安全识别进行探索和研究，在保障了信息畅通传递的同时，能实现更加安全、可靠的传输。

1.2 国内外研究情况

在国外，二维码编码解码的相关技术的研究起步于上世纪 80 年代。在二维码图形表示的编码技术研究方面，很多种码制已经被确立。而在二维码符号标准化研究方面，PDF417，QR Code，Code 49，Code 16K，Code One 等码制的符号标准已经由国际自动识别制造商协会、美国标准化协会已完成了。而对于本文着重探讨的 QR Code，其国际标准，已经由国际电工委员会第 1 联合委员会的第 31 分委员会，即条码自动识别技术委员会制定。在二维码识别设备的设计、研究、制造、生产等方面，美国、日本等国的设备制造商生产的图形识别设备、图形生成设备，已经广泛在各类二维码应用系统中应用[2]。

自 1993 年开始，中国物品编码中心便开始了对二维码技术的和研究，研究人员和专家对几种常用的二维条码的技术规范进行了翻译和跟踪研究。随着我国市场经济的不断完善和信息技术的迅速发展，国内对二维码这一新技术的需求与日俱增。

1.3 课题研究对象

基于组织机构代码中心对于二维码识别系统的需求，本课题主要对现在流行的 QR 二维码编码解码技术、对称非对称加密技术[3]、字符串压缩技术[4]、字符串匹配[5]、多线程技术[6]等进行研究和分析，然后结合这些技术进行组织机构代码二维码安全识别系统的实现。

1.4 本文组织结构

本文叙述了安全二维码识别系统的需求、分析、设计、实现，对于主要模块的主要算法或调用的方法进行了叙述。然后进行了功能的测试，然后进行了各方面的改进和完善。全文共分为 7 个章节。

第 1 章介绍了研究背景与意义、国内外研究情况。确定了研究对象。

第 2 章进行了需求分析，分析了系统所需的各个功能。

第 3 章介绍了系统功能模块的设计。

第 4 章介绍了系统功能模块的实现。

第 5 章介绍了系统测试和改进。

第 2 章 需求分析

2.1 需求简要概括

当前组织机构代码信息已经为我国的中国人民银行、税务总局、财政部、外汇管理局、银监会等金融系统，公安部、最高人民检察院、最高人民法院等法制系统，人力资源和社会保障部等社会保障系统，以及国家的统计、海关、外交、文化、建设、军队、铁路等二十多个政府部门提供了广泛的有关组织机构基本信息的服务。 随着组织机构代码信息量不断增加、应用范围不断拓展，为了给多领域多系统提供良好支撑，要求《组织机构代码管理系统》对数据质量提供保证；同时也对这一基础信息服务系统中海量数据检索、处理效率提出了更高的要求。为此对《组织机构代码管理系统》中的一个子系统组织机构信息的检验子系统进行前期的设计与实现。组织机构代码管理系统如图 2-1 所示。

图 2-1 业务信息采集系统结构

新疆维吾尔自治区组织机构代码申报表

机构代码	792424089	是否涉密	否	是否发卡	否		
机构名称	北京诚信能环						
法人代表	不知道	身份证号	372833197704113918				
经营或业务范围	I T						
主管部门	发改委					副本数量	1
成立日期	2011-10-13	职工人数	500	注册资金	0.0万元	货币种类	156
外方国别		行政区划	370100 山东省济南市				
注册地址	观湖国际						
邮政编码	200010	电话	12345677	批准文号或注册号	2213451234567		
批准机构	济南市工商行政管理局 004189090						
法人手机		E-mail		网 址			
经办人姓名		证件号码					
经办人电话	(手机)			经营期限			
经营地址							
经营邮编		经营电话		经费来源			
开户银行				开户帐号			
主要产品	产品1: 产品2: 产品3:						

图 2-2 组织机构代码申报表

图 2-3 带有二维码的组织机构代码证

根据图 2-2 和图 2-3 可知，需求方需要的使 用的二维码系统应具有如下功能。

1)二维码中的信息包括某个组织机构的基本信息，如组织机构代码，组织机构名称，机构类型，地址等。

2)有些机构的基本信息涉密，需要保护，不能向公众开放。如某些涉及到国家安全、国防的机构。

3)二维码是组织机构代码证的防伪标志，应具有防伪功能，不能被他人伪造。

4.需求方可以使用二维码进行组织机构的信息录入、信息核查、信息管理等操作。

5)该二维码应尝试存放尽可能多的有效信息，同时也尽可能的保障识别成功率。

6)二维码图片通过专用的扫描设备获得，本系统的识别部分是作为外设和数据间的接口，即输入图片，输出识别后的数据信息。

2.2 本章小结

本章进行了需求分析，分析了用户使用该系统希望达到的目的和效果：二维码中含有组织机构的基本信息，必要时需要对信息加密；该二维码应具有防伪功能，二维码所含信息要方便录入和核查，二维码的有效容量尽可能大。由此可见用户希望使用一种安全的灵活的二维码识别系统，并具有简单的信息管理功能。

以上根据这些，分析了系统所需的各个功能，具体分别是二维码编码、二维码解码、二维码信息加密、二维码信息解密、二维码搜索、二维码批量生成、组织机构信息数据结构化、信息识别、文本压缩、文本解压缩等，并对这些功能的效果进行了相应的描述。

最后，将这些功能进行细分为子功能，方便接下来的模块化设计和实现。由此也可以看出，该系统实现的重点在于二维码编码解码、加密解密、二维码搜索、二维码批量生成部分。这几部分相互独立，应分开设计，但它们之间也有重叠的部分，在设计和实现的时候应考虑模块的通用性，以提高开发效率。

第 3 章 系统功能模块的设计

3.1 系统模块划分

根据需求分析，可以将系统大致分为 4 个主要的功能模块，分别为二维码编码与加密，二维码解码与解密，二维码搜索，二维码批量生成。如下图 3-1 所示。

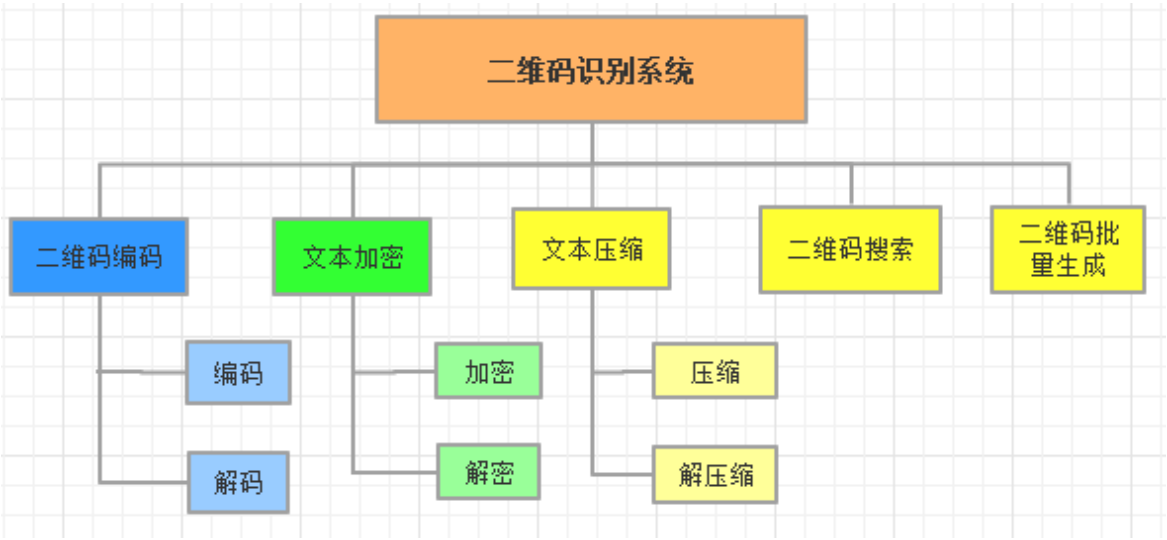


图 3-1 系统主要功能

各功能相应的描述如下表 3-1 所示。

表 3-1 功能描述

功能类别	功能描述
二维码编码	根据相关设置，对某文本进行二维码编码生成一张指定格式的二维码图片。
二维码解码	对某二维码图片进行解码，识别其中含有的文本信息。
二维码信息加密	对需要进行二维码编码的文本进行加密，达到安全保护信息的目的
二维码信息解密	对于二维码解码后的经过加密的信息进行解密，还原原有的文本。
二维码搜索	根据关键字在众多二维码中进行搜索，返回二维码信息中包含关键字的二维码
二维码批量生成	根据存有众多组织机构信息的文本，每一个组织机构生成一个对应的二维码。

表 3-1 功能描述（续）

功能类别	功能描述
组织机构信息数据结构化、信息识别	对于组织机构信息进行结构化，方便识别对应信息。
文本压缩	对于文本进行压缩，缩短其长度
文本解压缩	对于压缩文本进行解压缩，还原文本

3.2 系统功能分析

系统各功能的详细分析如下表 3-2 所示。

表 3-2 各大功能的细分

功能类别	子功能
二维码编码	设置需编码的文本
	设置二维码编码的选项
	设置加密选项
	生成二维码图片并显示
	保存二维码图片
	二维码图片打印
	设置文本信息按组织机构信息格式化
二维码解码	打开二维码图片并显示
	根据二维码图片解码并显示文本
	根据文本将格式化的组织机构信息对应识别
	显示识别后的组织机构信息
	设置解密选项
二维码信息加密	根据所选加密算法和密钥对文本进行加密
	不同加密算法
	密钥手动输入或自动输入（固定密钥和随机生成）
二维码信息解密	根据解密算法和密钥进行解密
	不同解密算法
	密钥手动输入或自动输入
二维码搜索	输入查找关键字
	从某二维码中识别关键字
	选择要进行搜索的存放二维码的路径
	根据关键字，多线程进行二维码解码、文本匹配、显示结果
	显示搜索所用时间
	从结果中移除不匹配项
	单击结果列表时，显示该项对应的二维码的信息
	双击结果列表时，显示该二维码文件所在文件夹并选中该文件

表 3-2 各大功能的细分（续）

功能类别	子功能
二维码批量生成	选择存放存有众多组织机构信息的文本的路径 按行生成对应的二维码信息，并显示生成结果 显示总共花费时间
组织机构信息数据结构化、信息识别	双击结果列表时，显示该二维码文件所在文件夹并选中该文件 将分隔开的各项信息进行格式化（在每项之间加入##） 将格式化的组织机构信息进行分项识别，显示在对应的栏目中
文本压缩	在二维码编码前进行文本压缩
文本解压缩	在二维码编码后文本解压缩

3.3 界面设计

本系统采用 C#语言编写，开发环境为 VS2013。在界面设计时，本文按照每一个主要功能模块对应界面中的一个页面的方法设计。其中每一个页面包含该功能对应的按钮、设置选项、输入文本框等。

可以将界面分为四个页面，每个页面对应一个系统主要功能模块。如下图 3-2、3-3、3-4、3-5 所示。

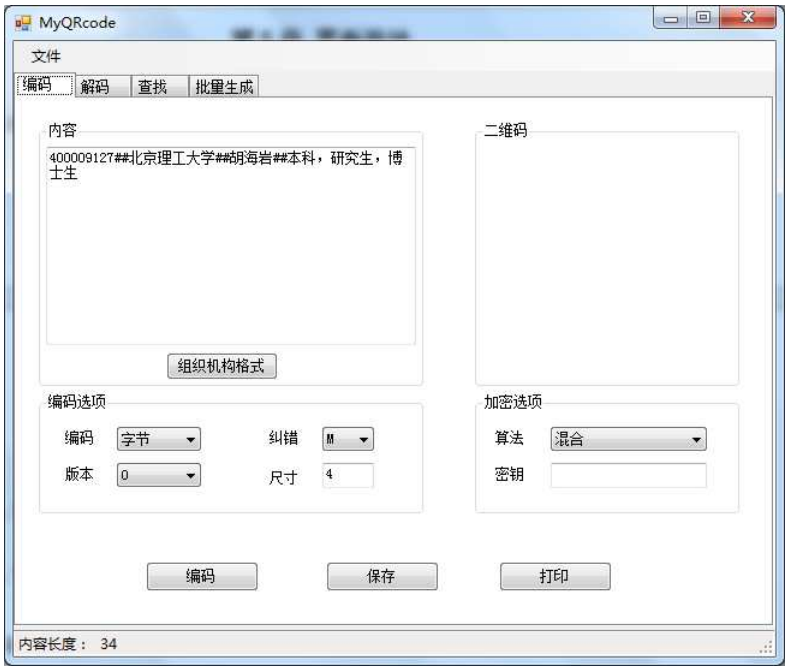


图 3-2 二维码编码页面

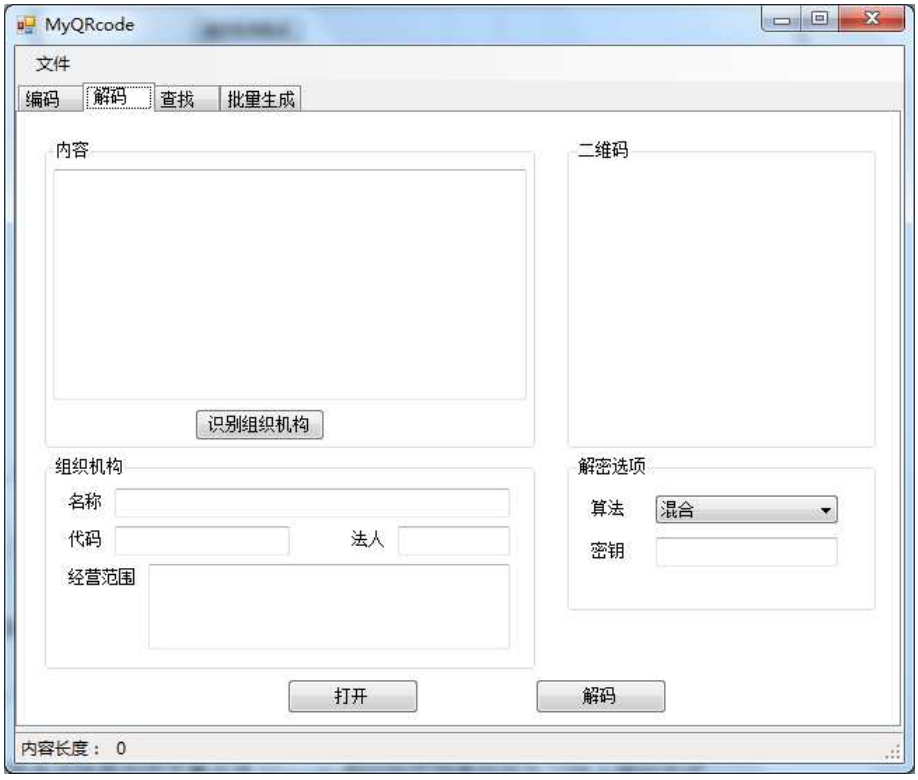


图 3-3 二维码解码页面

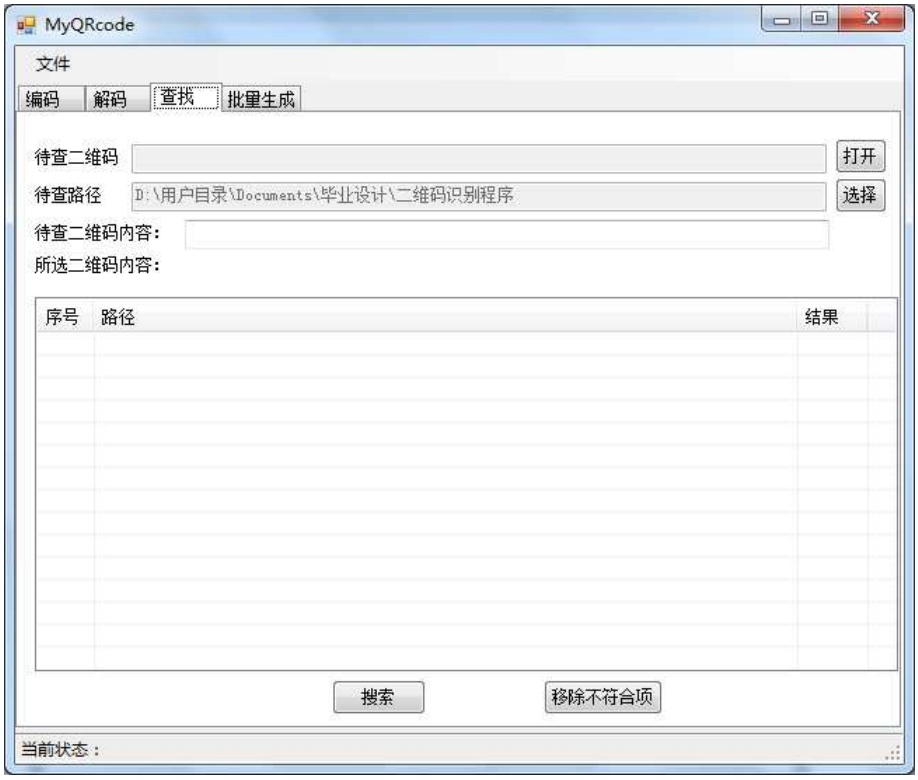


图 3-4 二维码搜索页面

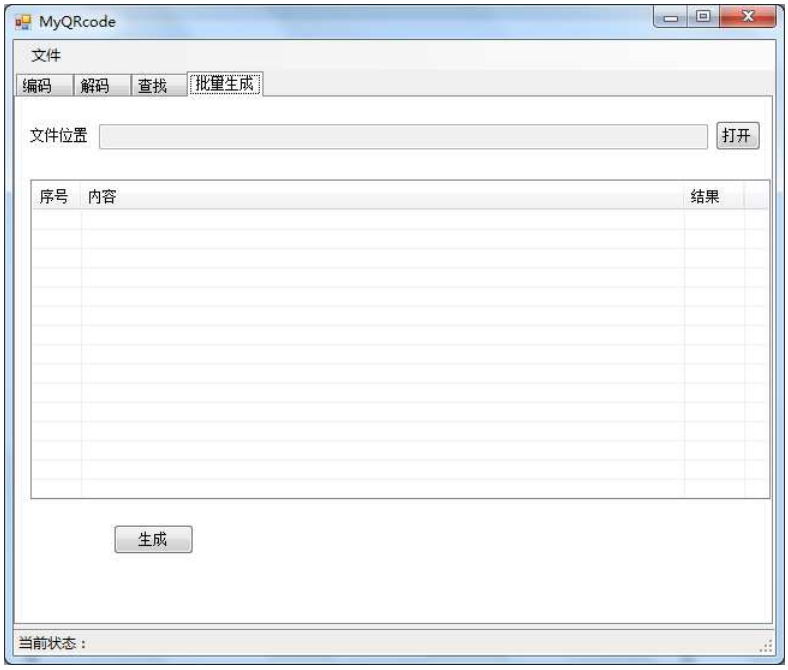


图 3-5 二维码批量生成页面

C#的默认控件搭建出来的界面非常朴素，可以使用开源类库 IrisSkin 美化界面。该类库中包含多种皮肤，可以在不该动原有设计的基础上实现简单的界面美化。例如，采用 MSN 皮肤后的界面如下图 3-6 所示：

图 3-6 采用 MSN 皮肤后的界面

3.4 二维码编码与加密模块设计

该部分设计的运行步骤如下。

1)用户输入一段文本。

2)然后根据需要选择是否进行格式化（例如：例如文本为：400009127 北京理工大学 胡海岩 本科，研究生，博士生；其中每一项用空格或回车分割开；格式化后为 400009127##北京理工大学##胡海岩##本科，研究生，博士生；即在每一项之间增##；这是为了读取时能够分别读取每一项）。

3)进行编码选项的设置，如果不进行设置则使用默认设置。

4)进行加密选项的设置，如果选择加密，则选择是否输入密钥；是则使用用户输入为密钥，否则使用系统生成的密钥。

5)如果进行了加密，则会自动进行压缩。（如果不加密则不进行压缩，因为文本内容不确定，无法针对性的压缩，压缩后效果有可能适得其反。）

6)点击编码，依次进行加密、压缩、二维码编码，并生成图片。

7)可以对图片进行保存或打印操作。

如下图 3-7、3-8 所示。

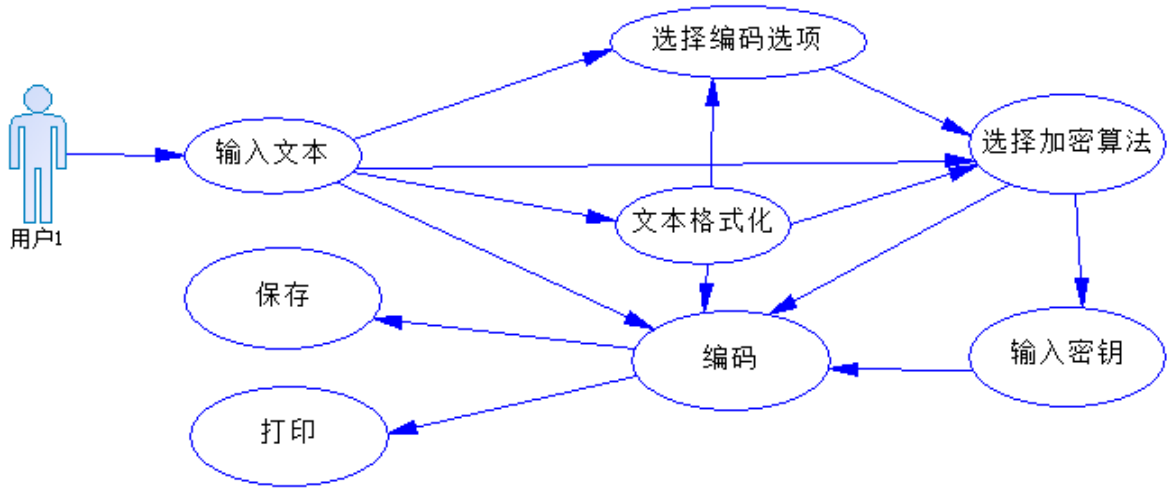


图 3-7 二维码编码用户操作逻辑图

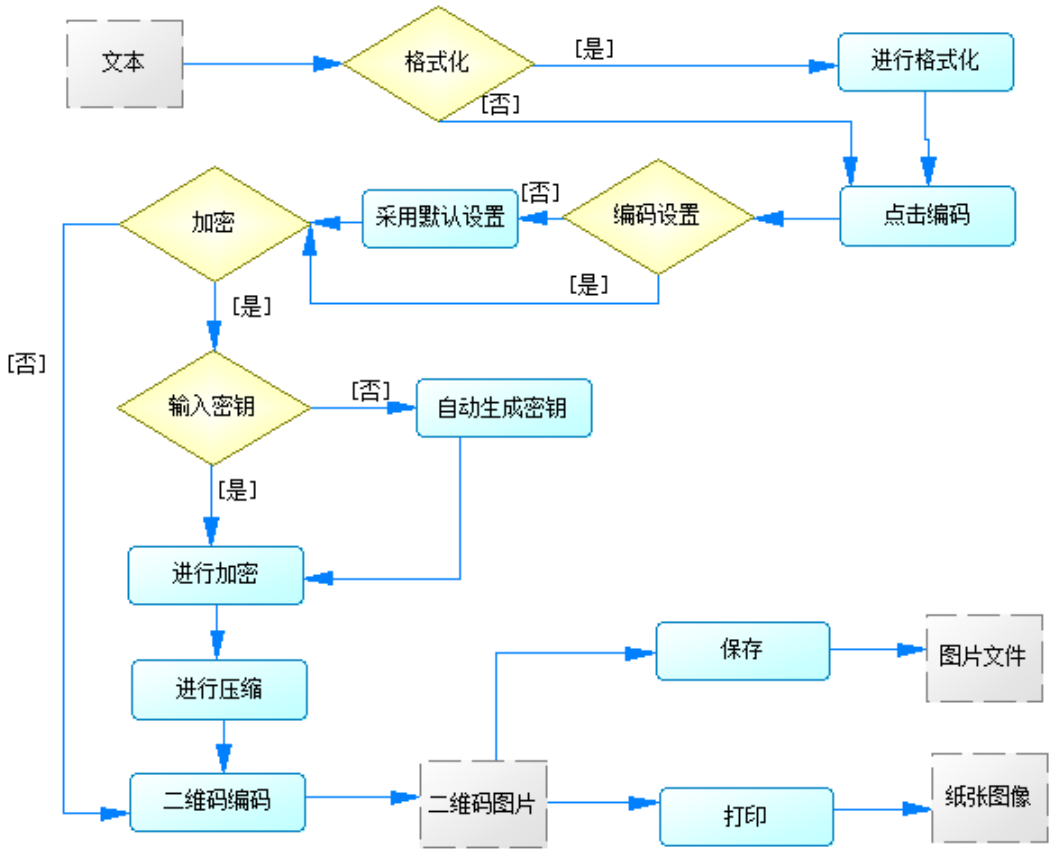


图 3-8 二维码编码系统处理逻辑图

3.5 二维码解码与解密模块设计

该部分设计的运行步骤是：

1)用户选择一个二维码图片并打开。

2)然后根据是否加密进行解密选项的设置，如果进行了加密，则选择是否输入密钥；是则使用用户输入为密钥，否则使用系统自动获取的密钥。

3)如果进行了加密，则需要在解密之前进行解压缩过程。

4)点击解码，依次进行解压缩、解密、二维码解码，并显示内容。

5)然后根据需要选择是否进行格式化读取(例如:若解码后内容为 400009127##北京理工大学##胡海岩##本科，研究生，博士生，可将该字符串按以##为分隔符进行分割，将得到的多项信息分别读取到相应的栏目内。

如下图 3-9、3-10 所示。

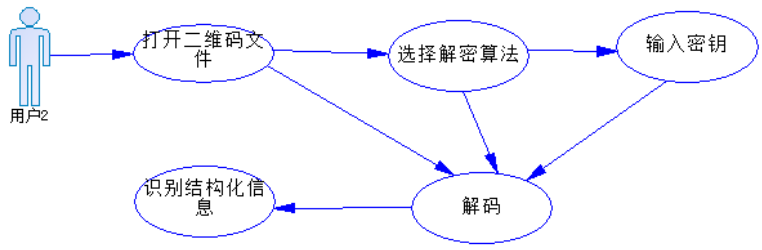


图 3-9 二维码解码用户操作逻辑图

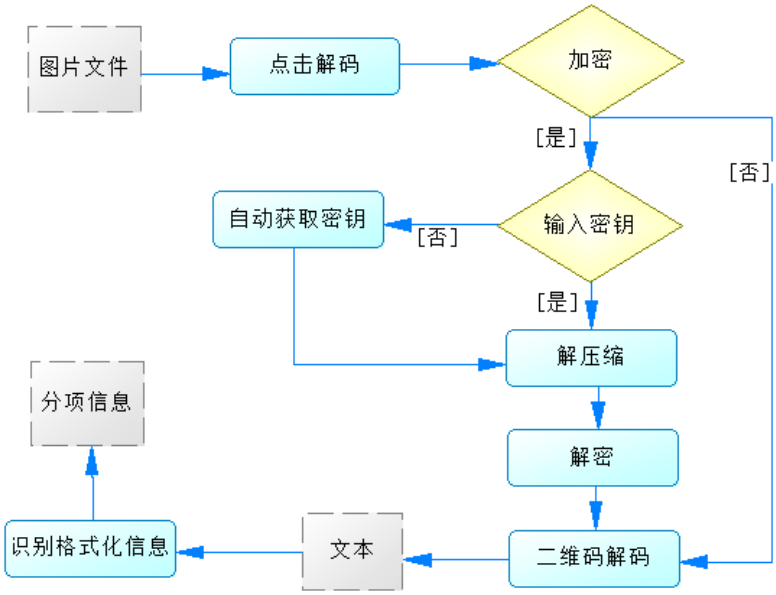


图 3-10 二维码解码系统处理逻辑图

3.6 二维码搜索模块设计

该部分设计的运行步骤是：

1)用户输入一个关键字；或打开一个二维码，并以二维码内容（或其中的一部分）为关键字。

2)选择待查的路径，此路径下保存着很多二维码图片。

3)点击搜索后，开始遍历所有图片，然后尝试解码，然后用关键字在解码后的内容中进行搜索，并在界面上显示结果。

4)点击去除不符合项，则去除除了结果为符合的其他所有项。

5)若单击结果中的某一项，则显示该二维码文件中的内容。

6)若双击某一项，则打开该文件所在文件夹并选中。

如下图 3-11、3-12 所示。

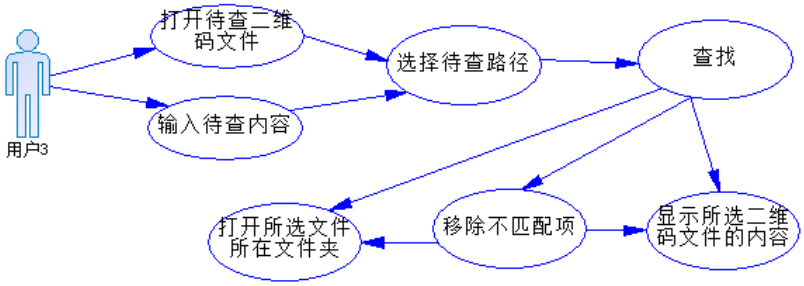


图 3-11 二维码搜索用户操作逻辑图

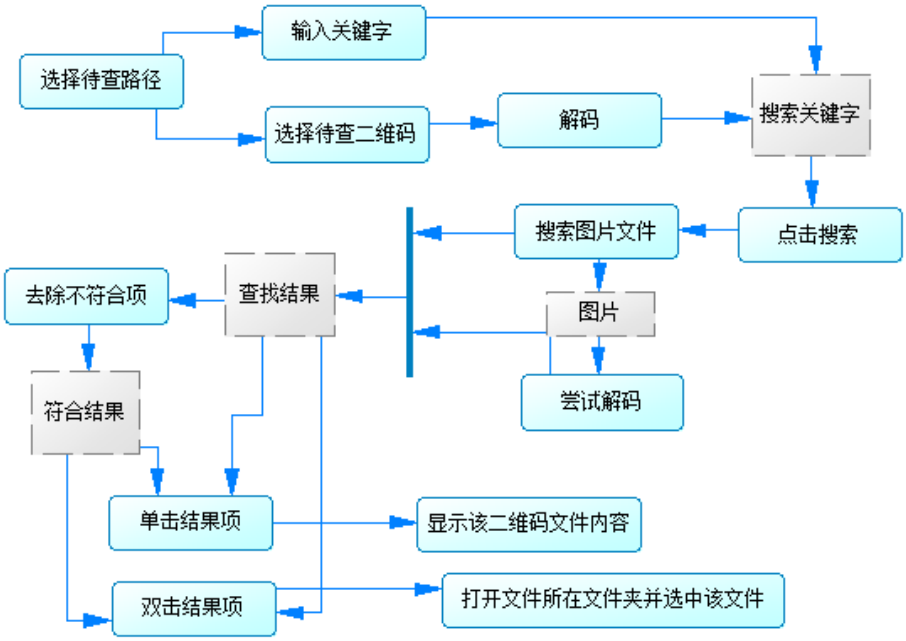


图 3-12 二维码搜索系统处理逻辑图

3.7 批量生成模块设计

该部分设计的运行步骤是：

1)选择文本所在路径，该文本内含有多行信息。

2)点击生成后，系统将文本按行读取，并按行尝试生成一个二维码文件。并在界面上显示结果。

4)若双击结果中的某一项，则打开该文件所在文件夹并选中。

如下图 3-13、3-14 所示。

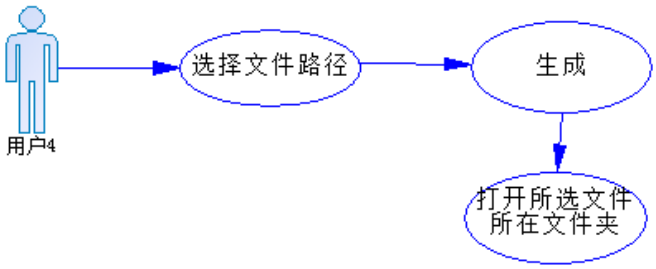


图 3-13 二维码批量生成用户操作逻辑图

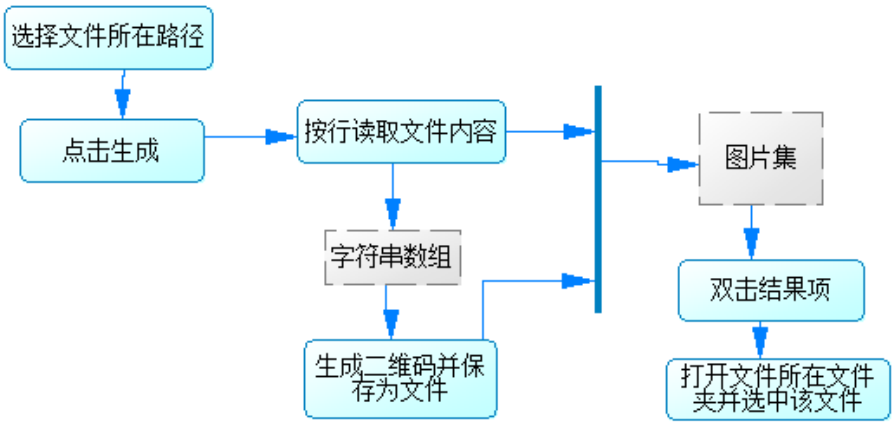


图 3-14 二维码批量生成系统处理逻辑图

3.8 本章小结

本章首先划分了主要功能模块，分析了系统所需的各个功能，具体分别是二维码编码、二维码解码、二维码信息加密、二维码信息解密、二维码搜索、二维码批量生成、组织机构信息数据结构化、信息识别、文本压缩、文本解压缩等，并对这些功能的效果进行了相应的描述。

接着徐叙述了系统的界面设计，根据不同的功能设计了不同的页面，用户可根据需要选择相应的页面，然后进行操作。然后又进行了简单的界面美化。然后叙述了系统各部分的功能模块的设计。系统主要分为四个部分：二维码编码与加密，二维码解码与解密，二维码搜索，二维码批量生成。本章分别从用户操作和系统处理的角度介绍了如何设计这四个模块。

第 4 章 系统功能模块的实现

4.1 二维码编码与解码

本系统在二维码的部分使用的 QR 二维码格式。此模块主要实现二维码的编码和解码过程。QR(Quick-Response)二维码是目前被广泛使用的一种二维码编码方式，其最大特点是解码速度快，识别成功率高。其标准 JISX0510 于日本在 1999 年 1 月发布，而其对应的 ISO 国际标准 ISO/IEC18004 于 2000 年 6 月获得批准。根据 Denso Wave 公司的资料，QR 码是属于开放式的标准，其专利虽然由 Denso Wave 公司持有，但不会被执行。QR 码呈正方形，只有黑白两色。在 3 个角落，印有较小，像“回”字的的正方形图案。这三个图案是帮助解码系统确定识别方向的图案，用户无需按照某一个特定方向对准，无论以任何角度进行图片扫描，二维码的内容均可以被正确读取。

4.1.1 QR 二维码图形分析

QR 二维码图形的结构主要包括如下几个部分：

1)位置探测图形、位置探测图形分隔符：用于对二维码的定位，对每个 QR 码来说，位置都是固定存在的，只是大小规格会有所差异；这些黑白间隔的矩形块很容易进行图像处理的检测。

2)校正图形：根据尺寸的不同，矫正图形的个数也不同。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。

3)定位图形：这些小的黑白相间的格子就好像坐标轴，在二维码上定义了网格。

4)格式信息：表示该二维码的纠错级别，分为 L、M、Q、H；

5)数据区域：使用黑白的二进制网格编码内容。8 个格子可以编码一个字节。

6)版本信息：即二维码的规格。QR 码符号共有 40 种规格的矩阵（一般为黑白色），从 21x21（版本 1），到 177x177（版本 40），每一版本符号比前一版本每边增加 4 个模块。

7)纠错码字：用于改正二维码模糊或损坏带来的识别错误。

如下图 4-1 所示。

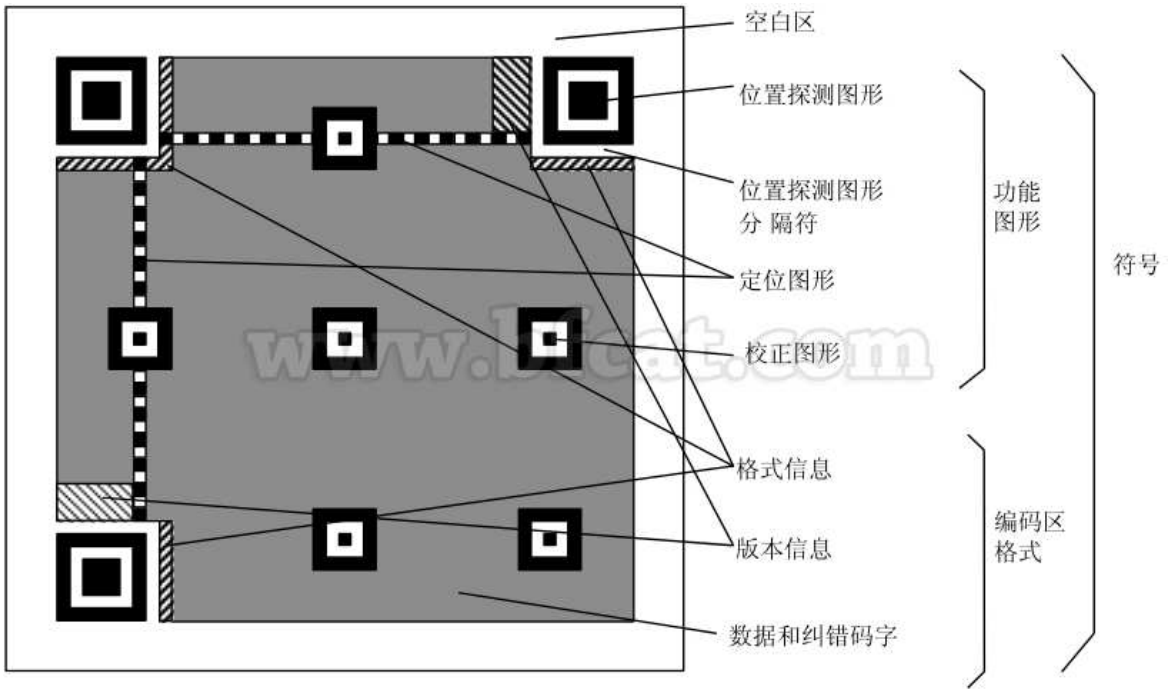


图 4-1 QR 码图形的结构

4.1.2 编码解码过程分析

编码过程可分为以下几个步骤。

1)数据分析：对编码字符类型进行分析，按照对应的字符集转换成符号字符。

2)数据编码：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。QR 码容量如下 4-2 图所示，编码模式与指示符如下图 4-3 所示

QR码资料容量	
数字	最多7,089字符
字母	最多4,296字符
二进制数（8 bit）	最多2,953 字节
日文汉字 / 片假名	最多1,817字符（采用Shift JIS）
中文汉字	最多984字符（采用UTF-8）
中文汉字	最多1,800字符（采用BIG5）

图 4-2 QR 码资料容量

模式	指示符
ECI	0111
数字	0001
字母数字	0010
8 位字节	0100
日本汉字	1000
中国汉字	1101
结构链接	0011
FNC1	0101（第一位置） 1001（第二位置）
终止符（信息结尾）	0000

图 4-3 QR 码编码模式与指示符

3)纠错编码：按需要将码字序列分块，并根据所选择的纠错等级，产生纠错码字，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。通常情况下，纠错等级越高，纠错信息所占空间越大，有效信息的实际容量就越小。QR 码错误纠错容量如下图 4-4 所示。

错误修正容量	
L水平	7%的字码可被修正
M水平	15%的字码可被修正
Q水平	25%的字码可被修正
H水平	30%的字码可被修正

图 4-4 QR 码错误纠错容量

4)构造矩阵：将探测图形、分隔符、定位图形、校正图形和码字模块放入矩阵中。把上面的完整序列填充到相应规格的二维码矩阵的区域中。

5)掩摸：将掩摸图形用于符号的编码区域，使得二维码图形中的黑色和白色区域能够具有最优的分布。

6)格式和版本信息：生成格式和版本信息放入相应区域内。版本 7-40 都包含了版本信息，没有版本信息的全为 0。二维码上两个位置包含了版本信息，它们是冗余的。版本信息共 18 位，其中 6 位时数据位，后面的 12 位是纠错位。QR 码编码图形如下图 4-5 所示。

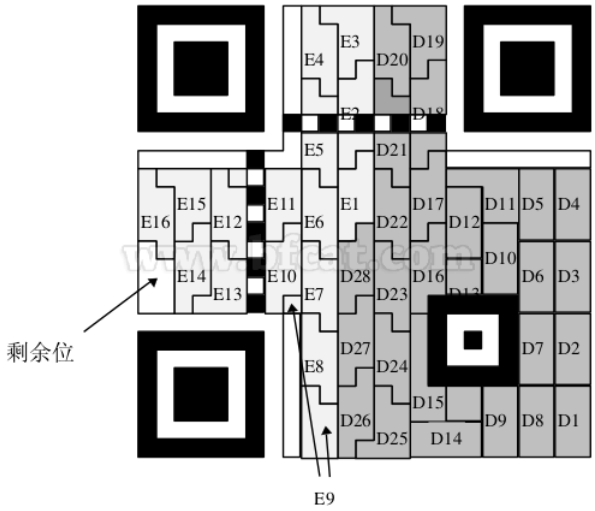


图 4-5 QR 码编码图形

4.1.3 实现方式和改进

QR 二维码现在已经是非常成熟的技术，所以本系统采用的实现方式是引用 ThoughtWorks 公司开发的 QRcode 开源类库，用于实现二维码编码和解码的功能。

该原始类库对于中文的支持不够好，所以需要对其进行改进，以便支持中文。简单来说就是内容凡是采用 Unicode 编码的字符串均改为 UTF-8 编码方式。判断字节数组数据是否为 Unicode 编码的方法如下图 4-6 所示。

```
public static bool IsUnicode(byte[] byteData)
{   //因为ascii编码当中的最大为127，这样判断后，
    //就能正确的判断是不是unicode，这样就能正确的解码中文了.
    bool isUnicode = false;
    try
    {
        foreach (byte value in byteData)
        {
            if (value > 128)
            {
                isUnicode = true;
                break;
            }
        }
    }
    catch (Exception)
    { //其中的是原有的代码 无法正确判断 中文
        string value1 = FromASCIIByteArray(byteData);
        string value2 = FromUnicodeByteArray(byteData);
        byte[] ascii = AsciiStringToByteArray(value1);
        byte[] unicode = UnicodeStringToByteArray(value2);
        if (ascii[0] != unicode[0])
            return true;
        return false;
    }
    return isUnicode;//返回是不是Unicode编码
}
```

图 4-6 判断是否为 Unicode 编码的方法

对类库中的 Encode 方法进行修改。如下图 4-7 所示。

```
public virtual Bitmap Encode(String content)
{
    if (QRCodeUtility.IsUniCode(content))
    {
        return Encode(content, Encoding.GetEncoding("UTF-8"));
        // 这样就支持中文了 （ 解码部分也要改下）
    }
    else
    {
        return Encode(content, Encoding.ASCII);
    }
}
```

图 4-7 修改后的支持中文的 Encode 方法

对类库中的 Decode 方法进行修改。如下图 4-8 所示。

```
public virtual String decode(QRCodeImage qrCodeImage)
{
    sbyte[] data = decodeBytes(qrCodeImage);
    byte[] byteData = new byte[data.Length];
    Buffer.BlockCopy(data, 0, byteData, 0, byteData.Length);

    Encoding encoding;
    if (QRCodeUtility.IsUnicode(byteData))
    {
        encoding = Encoding.UTF8;           //可以支持中文，或英文+中文混合
    }
    else
    {
        encoding = Encoding.ASCII;//英文字符
    }
    String decodedData;
    decodedData = encoding.GetString(byteData);
    return decodedData;
}
```

图 4-8 修改后的支持中文的 Decode 方法

至此，使用该类库已经支持中文编码解码了。

4.2 加密与解密

加密是保障信息系统安全的一个重要手段。加密技术已经广泛地被应用在各方面的安全技术中，例如：身份校验、信息传递、金融交易、数据保存等。在计算机中，尽管加密和解密的算法、方式多种多样各不相同，但整个加密系

统却都有类似的构成要素。

1)算法——对数据加密和解密的数学函数。

2)密钥——加密或解密所需要的除密码算法之外的关键信息。加密和解密算法的操作通常都是在一组密钥的控制下进行的，分别称为加密密钥(Encrypt ion Key) 和解密密钥(Decryption Key)。

3)明文——用户希望保密的信息原始文本。

4)密文——被加密后的信息文本。

加密系统分为专用密钥系统和公开密钥系统。

专用密钥系统是指用相同或两个等价的密钥对消息进行加密和解密。因此，该系统又被称为“对称密钥”系统。由于其加密和解密所用的密钥可以由一个推测出另一个，所以这两个密钥都不能对外公布，只能由通信双方保存。使用该系统的代表算法有 DES、AES、IDEA 等等。

公开密钥系统是指用一个公开密钥对信息进行加密，用另一个私有密钥对信息进行解密。在该系统中，用于加密的的密钥即使对外公开，也不会影响该信息本身和用于信息解密的私有密钥的保密性。这是由于，虽然私有密钥由公开密钥决定的，但却无法根据公开密钥计算出私有密钥。公开密钥系统也被称为“不对称密钥”系统。公开密钥系统常常被用于为数据建立“数字签名”，例如 Email、在线支付等，以证明数据是原始的而不是被伪造的。其模型如下图 4-9 所示。该系统的代表算法有 RSA 算法和椭圆加密算法。

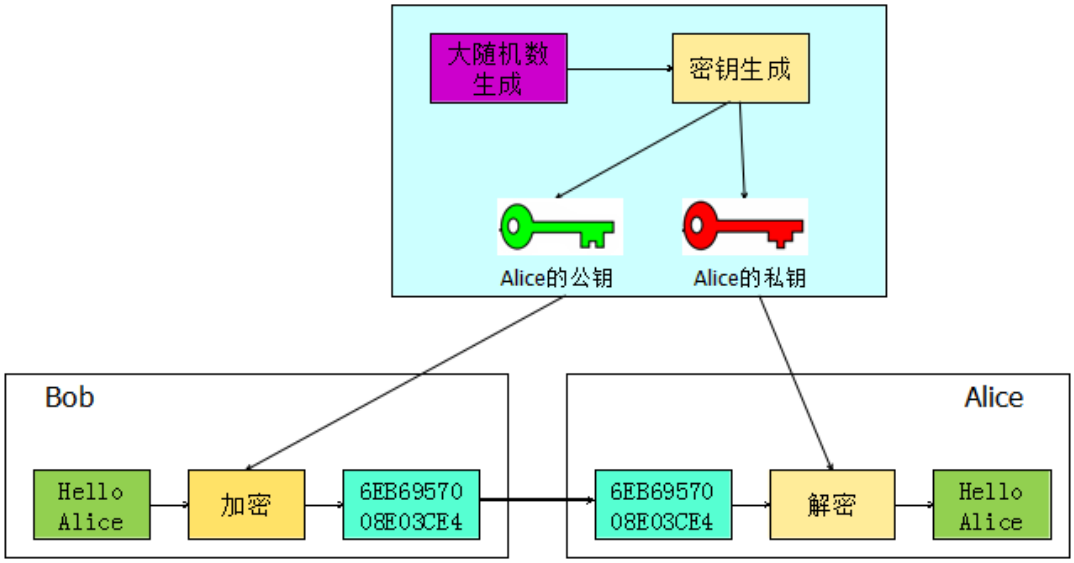


图 4-9 公钥加密模型

本系统将主要采用专用密钥系统和公开密钥系统的各自的代表算法 DES 和 RSA，以及改进后的混合加密方法，作为加密算法实现加密和解密的实现方式。

4.2.1 DES 算法

4.2.1.1 算法简介

1974 年，IBM 研究员以霍斯特.费斯妥（Horst Fiestel）提出的 Lucifer 算法为背景提出的。1976 年被确定为联邦标准，1977 被授权用于所有非机密资料，后来也用于银行的 ATM。2002，DES 发展到 3DES，DES-x，GDES 等。但由于加密算法自身的缺陷，后被高级加密标准 AES 所取代。

DES 算法是一种分组密码，就是将明文进行分组，然后分别进行加密的算法。分组密码的特点是分组密码加解密速度较快，现代分组密码发展非常快，技术较成熟，使用广泛。

DES 算法基于 Erokhoffs 假设：密码分析者已有密码算法及实现的全部详细资料。也就是说算法的安全性完全依赖于密钥。公开算法的意义在于，使用范围从军事到民用拓展；使用者须确认算法不存在陷门；可以由全世界的密码学家对其安全性评估，确保其足够的安全强度；可以进行标准化通信。

4.2.1.2 算法概述

1)明文和密文分组长度为 64 比特。

2)算法包含两部分：迭代加解密和密钥编排。

3)Feistel 结构（加解密相似）：加密和解密除密钥编排不同外， 完全相同。

4)密钥长度：56 比特（DES 的密钥空间：2^{56}），每 7 比特后为一个奇偶校验位（第 8 位），共 64 比特。

5)轮函数采用混乱和扩散的组合，共 16 轮。

4.2.1.3 加密步骤

1)设密钥 K(64 位) = 133457799BBCDFF1，即

K(64 位) = 0001001**1** 0011010**0** 0101011**1** 0111100**1**

1001101**1** 1011110**0** 1101111**1** 1111000**1**

其中加粗位为奇偶校验位，即实际密钥为 56 位。

2)生成 16 个子钥(48 位)。

3)用子钥对 64 位数据加密。

加密过程如下图 4-10、4-11 所示。

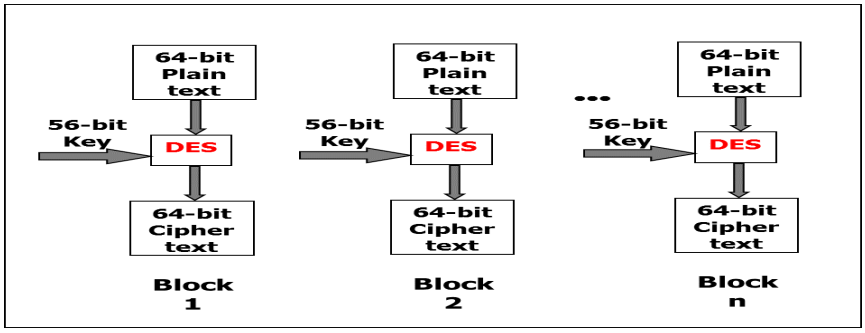


图 4-10 DES 算法全文加密过程

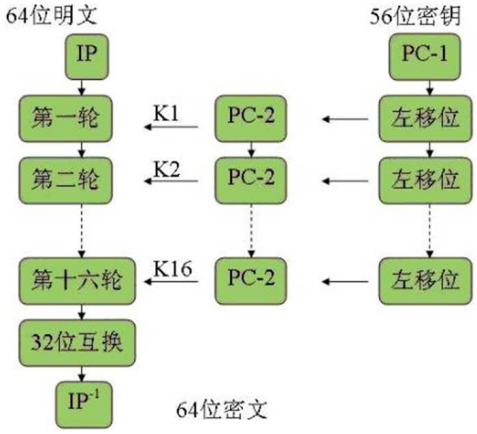


图 4-11 DES 算法单一块加密过程

4.2.1.4 安全性

DES 是从 1975 年被美国联邦政府确定为非敏感信息的加密标准，它利用 56 比特长度的密钥 K 来加密长度为 64 比特的明文，得到 64 比特长的密文。

1997 年，由于计算机技术迅速发展，DES 的密钥长度已经太短，NIST 建议停止使用 DES 算法作为标准。目前，二重 DES 和三重 DES 仍然广泛使用。

4.2.1.5 实现方式

引用由常熟理工的刘子谊、赵勇开发的加密类库，该类库包含 DES、3DES、AES、IDEA 四种加密算法。为了方便本系统引用，对于这四种算法进行了封装，编写相应的 Operate 类，该类含有两个静态方法：

加密方法：public static string Encrypt(string Source, string Key);

解密方法：public static string Decrypt(string Source, string Key);

DES 算法的封装方法如下图 4-12、4-13 所示。

```
public static string Encrypt(string Source, string Key)
{
    if (String.IsNullOrEmpty(Source))
    {
        throw new Exception("没有输入明文！");
    }

    if (Encoding.Default.GetBytes(Key).Length != 8)
    {
        throw new Exception("密钥个数必须为8个字符或4个汉字！");
    }

    int[,] cipherKey = new keyExpansion().GetKey(Key);
    int[] cipher = Encryption.GetEncryption(Source, cipherKey);

    StringBuilder str = new StringBuilder(64);
    for (int i = 0; i < cipher.Length; i++)
    {
        str.Append(cipher[i]);
    }
    //密文个数必须为64或64的倍数个二进制数
    return str.ToString();
}
```

图 4-12 DES 算法的封装后的加密方法

```
public static string Decrypt(string Source, string Key)
{
    if (String.IsNullOrEmpty(Source))
    {
        throw new Exception("没有输入明文！");
    }
    else if (Source.Length % 64 != 0)
    {
        throw new Exception("密文长度有误！（密文长度须为64的倍数）");
    }
    else
    {
        for (int i = 0; i < Source.Length; i++)
        {
            if ((!Source[i].ToString().Equals("1"))
                && (!Source[i].ToString().Equals("0")))
            {
                throw new Exception("密文内容有误！（密文内容仅含有0或1）");
            }
        }
    }

    if (Encoding.Default.GetBytes(Key).Length != 8)
    {
        throw new Exception("密钥个数必须为8个字符或4个汉字！");
    }

    int[,] cipherKey = new keyExpansion().GetKey(Key);
    DESDecryption decryption = new DESDecryption();
    return decryption.GetDecryption(Source, cipherKey);
}
```

图 4-13 DES 算法的封装后的解密方法

4.2.2 RSA 算法

4.2.2.1 算法简介

在 1978 年，来自美国麻省理工学院的 Ron Rivest、Ad Shamirh 和 Len Adleman 开发出了著名的 RSA 加密算法。RSA 取名来自他们三个人姓氏的首字母。到目前为止，RSA 算法是目前最常用最有影响力的公开密钥系统的算法，已被 ISO 推荐为公钥数据加密标准，因为它能够抵抗目前已知的所有密码攻击方式。

RSA 算法是第一个既可以用于信息加密又可以用于数字签名的算法。RSA 是被研究得最广泛的公钥算法，从提出到现在的三十多年里，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的公钥方案之一。

4.2.2.2 算法描述

RSA 算法基于一个十分简单的数论事实：将两个非常大的质数相乘十分容易，但若要对乘积进行质因数分解却极其困难，因此可以将乘积做某种运算后公开作为加密密钥，将两个乘数做某种运算后作为私有的解密密钥。

4.2.2.3 加密解密步骤

1)密钥的产生：

①选择两个的非常大的质数 p 和 q。

②计算这两个质数的乘积 n=p × q，并计算 n 的欧拉函数值 φ(n)=(p-1)(q-1)。

③选择一个整数 e，需满足 1<e< φ(n)，且最大公约数 gcd(φ(n)，e)=1。

④计算 e 在模 φ(n)下的乘法逆元，即满足等式 d • e≡1 mod φ(n)的 d。因 e 与 φ(n)互质，由模运算可知，它的乘法逆元一定存在。

⑤将{e，n}为公开密钥，{d，n}为私有密钥。

2)加密：加密的时候需要首先将明文进行分组，每组的长度需小于 $\log_2 n$。然后对每个明文分组 m，作加密运算：c≡m^e mod n。

3)解密：对每一个密文分组 c，作解密运算：m≡c^d mod n。然后将每一个 m 拼接，从而还原出明文。

加密过程如下图 4-14 所示。

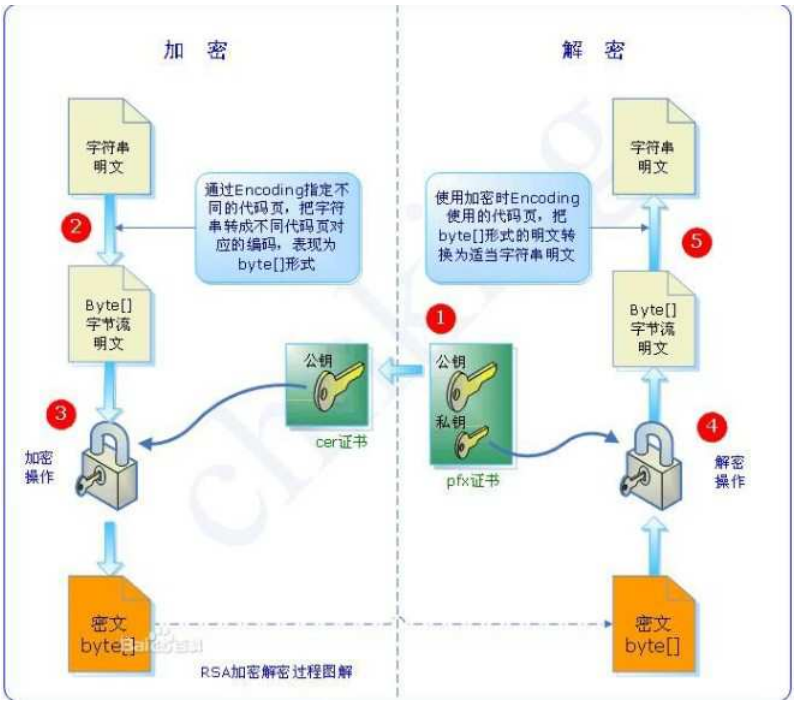


图 4-14 RSA 加密解密过程

4.2.2.4 安全性

RSA 的安全性基于数论中大整数分解的困难性。因为：如果 RSA 的模数 n 被成功地分解为两个质数相乘，即 p×q，则可以获得 n 的欧拉函数值 φ(n)=(p-1)(q-1)，如此破译者就能够从公共密钥 e 推导出私有密钥 d，即 d≡e-1 mod φ(n)，从而成功破译密文。

随着计算机的计算能力和速度的不断提高，原有被认为是不可能质因数分解的超大数已经陆续被成功分解。例如 RSA-129（它是一个 129 位的十进制整数特），在一次实验中已由各地的志愿人员采用二次筛法，在网络上通过分布式计算历时 8 个月于 1994 年 4 月被成功分解，此次实验中研究人员总共动用 1600 部电脑。而对于 RSA-130（130 位十进制数），研究人员采用了一种新算法，称为推广的数域筛法，已于 1996 年 4 月被成功分解。而 RSA-768（768 位二进制数, 232 十进制数）在 2009 年也依次法被分解，这一事件的发生被认为是威胁到了现通行的 1024bit 密钥的安全性，专家建议将数字证书的密钥升级至 2048bit 或以上。

综上所述，在使用 RSA 算法时对其密钥的选取要特别注意其长度。RSA 存在很多种攻击，并不是因为算法本身存在缺陷，而是由于参数选择不当造成

的。虽然随着分布式计算技术和量子计算机理论日趋成熟，RSA 算法的安全性受到了巨大的挑战，但是实际上只要其钥匙的长度足够长，用 RSA 加密的信息实际上是不能被解破的。估计在未来一段比较长的时间内，密钥的长度在 1024bit 至 2048bit 之间的 RSA 还是安全的。

4.2.3 实现方式

编写 RSA 类，该类中含有加密和解密方法。如下图 4-15、4-16、4-17 所示。

```
public byte[] EncryptToByte(string Source, string PublicKey)
{
    RSACSP.FromXmlString(DESManager.DesDecrypt(PublicKey));
    return RSACSP.Encrypt(Encoding.UTF8.GetBytes(Source), false);
}
```

图 4-15 RSA 算法的加密到字节数组方法

```
public string Encrypt(string Source, string PublicKey)
{
    RSACSP.FromXmlString(DESManager.DesDecrypt(PublicKey));
    byte[] tempByte = EncryptToByte(Source, PublicKey);
    StringBuilder sb = new StringBuilder();
    foreach (byte b in tempByte)
    {
        sb.Append(b.ToString().PadLeft(3, '0'));
    }
    return sb.ToString();
}
```

图 4-16 RSA 算法的加密到字符串方法

```
public string Decrypt(string Source, string PrivateKey)
{
    int ByteNum = Source.Length / 3;
    byte[] tempByte = new byte[ByteNum];
    StringBuilder sb = new StringBuilder();
    try
    {
        for (int i = 0; i < ByteNum; i++)
        {
            tempByte[i] = Convert.ToByte(Source.Substring(i * 3, 3), 10);
        }
    }
    catch
    {
        throw new Exception("密文内容有误");
    }
    RSACSP.FromXmlString(DESManager.DesDecrypt(PrivateKey));
    string result = Encoding.UTF8.GetString(RSACSP.Decrypt(tempByte, false));
    return result;
}
```

图 4-17 RSA 算法的解密方法

其中 RSACSP 是 RSACryptoServiceProvider 类的一个对象，该类是 C#自带的用于实现 RSA 算法的类。DESManager 是编写的 DES 算法类，用于将 RSA 的公钥私钥的加密和解密，以防止私钥泄漏造成的信息被盗。使用此方法时，公钥密钥在 RSACryptoServiceProvider 类构造后就已经生成。可调用相应方法获得公钥和密钥。其中，公钥和密钥也经过了加密。如下图 4-18、 4-19 所示。

```
//获取公钥
internal string GetPublicKey()
{
    string publicKey = RSACSP.ToXmlString(false);
    string encryptedPK = DESManager.DesEncrypt(publicKey);
    return encryptedPK;
}
```

图 4-18 获取公钥方法

```
//获取私钥
internal string GetPrivateKey()
{
    string privateKey = RSACSP.ToXmlString(true);
    return DESManager.DesEncrypt(privateKey);
}
```

图 4-19 获取私钥方法

4.3 加密模块的改进

通过将加密系统和二维码编码系统相结合的方式，二维码内容的安全性得到了保障，实现了信息的安全传播。但是由此也产生了一些新的问题，例如：加密后的字符串长度要比原文的长度还要长，这减少了二维码的实际的信息含量。还有，对于加密系统，无路是加密还是解密，都需要手动输入密钥，这显然降低了效率，也带来了很不好的用户体验，二维码的意义就在于信息的快速传播，这样的加密系统显然使得二维码犹如戴上了枷锁，束缚了行动。以上的问题，均需要一个解决方法，使得在安全的基础上，最大程度的能够保证二维码的容量大、灵活程度高、速度快等特点。

4.3.1 加密后进行信息压缩

4.3.1.1 DES 算法加密后压缩

将字符串用 DES 算法加密后，密文为只含 0 或 1 的二进制字符串。例如

将“北理工”（不含引号）使用 DES 算法加密（密匙为 dddddddd）后为：
10101111110101110101111010111101110100001100010110010001000110
01011110101101010011000110011000000001011101111011000100100110001000100，
长度为 128。

这显然造成了浪费，因为这个字符串 1 个字节才存了 1 位信息。而二维码能存储的信息长度有限。即使长度在最大信息长度内，若二维码内信息过多也会造成生成的图片过于复杂，加大识别成功的难度，所以有必要对加密后的字符串进行压缩。

压缩方法 1：将加密后字符串每 4 位二进制转为 1 为十六进制字符，这样长度将减少，依次法则起到了压缩的效果。将上述二进制字符串使用此方法压缩后为 afd75ebdd0c591197ad4c6602ef624c4，长度为 32，为原来的四分之一。

压缩方法 2：使用 c#提供的 ToBase64String 方法。在 ToBase64String 方法中，会对字节数组中的连续三字节（不足三字节时会用“=”补齐）进行一次编码，编码得的字符串长度为 4 位，而且得出来的 4 位的字符串里面的字符肯定是由大小写字母、数字（0~9）、+、/组成（末尾也许会含有“=”）。先将二进制字符串每 8 位转换为 1 个 byte 型数字形成一个 byte 数组，然后使用 ToBase64String 将其转换为 Base64 编码字符串。将上述二进制字符串使用此方法压缩后为 r9devdDFkRl61MZgLvYkxA==，长度为 24，为原来的近五分一。压缩效果比方法 1 更好。

4.3.1.2 RSA 算法加密后压缩

使用以上提供的 RSA 加密算法加密后的内容为每字节数字转换为 3 位（不足 3 位时高位补 0）字符串形成的十进制数字字符串，例如将“北理工”使用 RSA 算法加密后为：
093149144108196048050149082100028068153052048217202251210136191219245069054132180242038214209047221148113024106107116247203011070239207055126213019131070166166079158043125019128009213158030172051186056091082054031155198243147100254042128120137141158094227073026068236181066031153013014159073058138197130212142088122248143185195211104209051198025199084053032029106064131088244245248063，
长度为 384。

压缩方法：同样是使用 ToBase64String 方法。将上述十进制数字字符串压

缩后为 im1ikHSIIwr2WjvdtMESwWnXJ1bwdi43FnKyh0tM9ad3E6DiKlg+Bq6Hum84qPu6nbK8oJtCxjFr0Vn+tEpvhbYe7SZ+UYk4IV8wnmY18MjRreiHx0pmRHWA7MNmKqEgAuENlXK+7+ONmulmb8AVRbXwN/z8Cby4ST1XhGXHYR0=，长度为 172。长度缩减为原来的一半多。

4.3.2 加密解密自动化

4.3.2.1 对称加密算法

对于本系统使用的 DES、3DES、AES、IDEA 等对称加密算法，由于加密解密共用一个密钥，所以可以使用如下方法实现加密解密自动化：

方法 1：加密时自动随机生成密钥，并将密钥以明文的形式的保存为与二维码图片文件一一对应的文件；解密时直接读取存放密钥的文件获得密钥，然后解密。优点：简单容易实现。缺点：明文保存密钥失去了加密的意义，另外需要额外生成文件，造成了额外的时间和空间开销。

方法 2：将方法 1 中的密钥加密后再保存为文件。以此解决了密钥泄漏的问题。缺点：实现起来较为繁琐，因为对密钥加密又产生了新密钥的保存问题。

方法 3：将密钥内嵌于程序中，加密解密时直接调用。优点：容易实现且效率高，无需额外的开销。缺点：密钥需要固定不能更改，因为如果更改，则使用旧密钥加密的二维码就解密了。

对于方法 3，虽然有缺点，但因为简单快捷而且又不失安全性的特点，所以本系统就采用这个方法。当然，这个方法也有待改进，并不是最好的方法。

4.3.2.2 非对称加密算法

对于本系统使用的 RSA 非对称加密算法，由于解密需要私钥，所以私钥也需要进行保存。而 RSA 算法中密钥并不能固定，所以密钥内嵌的方法就不能使用了。一下探究了几种方法：

方法 1：将生成的密钥字符串进行导出然后保存为文件存储。需要指出的是，由于安全性方面的考量，不建议使用这种方法保存私钥，建议在使用这种方法的时候仅导出公钥。C#自带的 RSACryptoServiceProvider 类提供了一个 ToXmlString(bool)方法，可以使用此方法将密钥导出为一个 xml 格式的字符串，然后将其保存到一个文件中。这个方法的参数为 true 时会导出私钥，否则仅导出公钥。需要读取密钥的时候，我们可以使用 FromXmlString(string)方法，将文件内的密钥信息加载到 RSACryptoServiceProvider 类中。在实际的应

用中，出于安全方面的考量，很少会这种方法来保存密钥。

方法 2：将密钥保存到密钥容器(key container)中。Windows 系统提供两种密钥容器(key store)，分别是(User Key Store)和(Machine Key Store)，用来保存密钥。密钥容器包括了一对密钥（公钥和私钥），和关于密钥的一些其它的信息，例如是否允许导出密钥、密钥的种类等。当需要存储或获取密钥时，可以通过指定密钥容器名称的方式来实现。

例如：使用 CspParameters 类创建和使用密钥容器。如下图 4-20 所示。

```
//实例化CspParameters对象
CspParameters cspPara = new CspParameters();
//指定CspParameters对象实例的名称
cspPara.KeyContainerName = "key_container_test";
//设置密钥类型为Exchange
cspPara.KeyNumber = 1;
//设置密钥容器保存到计算机密钥库（默认为用户密钥库）
cspPara.Flags = CspProviderFlags.UseMachineKeyStore;
//实例化RSA对象的时候，将CspParameters对象作为构造函数的参数传递给RSA对象，
//如果名称为key_container_test的密钥容器不存在，
//RSA对象会创建这个密钥容器；
//如果名称为key_container_test的密钥容器已经存在，
//RSA对象会使用这个密钥容器中的密钥进行实例化
RSACryptoServiceProvider rsaPro = new RSACryptoServiceProvider(cspPara);
```

图 4-20 CspParameters 使用方法

当不再需要某密钥容器时，可以使用下面的方法进行删除。如下图 4-21 所示。

```
CspParameters cspPara = new CspParameters();
cspPara.KeyContainerName = "key_container_test";
cspPara.Flags = CspProviderFlags.UseMachineKeyStore;
RSACryptoServiceProvider rsaPro = new RSACryptoServiceProvider(cspPara);
//不在密钥库中保存此密钥容器
rsaPro.PersistKeyInCsp = false;
//释放rsaPro占用的所有资源，包括密钥容器。
rsaPro.Clear();
```

图 4-21 删除密钥容器方法

该方法的安全性在于，除非获取到了密钥容器的名称，否则无法从密钥库中提取到这个密钥容器。所以在本地加密中使用的密钥（特别是私钥）保存在密钥容器中是相对安全的方法。

方法 3：使用数字证书。如果密钥需要在不同的计算机上传递使用，那么可以使用将密钥保存在数字证书中的方法。或者更准确的说法是，首先生成一个新的数字证书，然后再使用其中的密钥。正式的数字证书需要到 CA 去申请。此外，在.Net 环境中，可以使用该平台提供的 MakeCert.exe 来生成一个临时的非正式的数字证书。然后可以使用 X509Certificate2 类来访问这个临时证书，可以用它来加载一个数字证书并获得数字证书中的密钥。若证书的保存方式是本地文件，则可以用如下图 4-22 所示的方法进行加载。

```
static byte[] EncryptDataByCert(byte[] data)
{
    //实例化一个X509Certificate2对象，并加载证书testCertificate.cer
    X509Certificate2 cert = new X509Certificate2(@"c:\testCertificate.cer");
    //将证书的公钥强制转换成一个RSACryptoServiceProvider对象，
    //然后可以使用这个对象执行加密操作
    RSACryptoServiceProvider rsa = (RSACryptoServiceProvider)cert.PublicKey.Key;
    byte[] enData = rsa.Encrypt(data, false);
    return enData;
}
```

图 4-22 本地证书加载方法

通常情况下，保存公钥所用的证书的扩展名为.cer，而保存私钥的证书的扩展名为.pfx。当加载一个保存私钥的证书时，还需要提供该私钥的保护密码，方法如下图 4-23 所示。

```
static byte[] DecryptByCert(byte[] endata)
{   //实例化一个X509Certificate2对象，并加载证书testCertificate.pfx。
    //由于证书testCertificate.pfx包含私钥，所以需要提供私钥的保护密码（第二个参数）
    X509Certificate2 cert = new X509Certificate2(@"c:\testCertificate.pfx", "123456");
    //将证书testCertificate.pfx的私钥强制转换为一个RSACryptoServiceProvider对象，用于解密操作
    RSACryptoServiceProvider rsa = (RSACryptoServiceProvider)cert.PrivateKey;
    byte[] data = rsa.Decrypt(endata, false);
    return data;
}
```

图 4-23 加载私钥数字证书方法

若证书保存在计算机的证书存储区（Certificate Store）中，就需要使用另一个类 X509Store 来访问证书存储区。证书存储区分为当前用户 Current User）和本地计算机（Local Machine）两个部分，这是根据访问权限而划分的，前者

用来保存当前登录用户所能访问到的数字证书，而后者用来保存登录到本机的所有用户所能访问的数字证书。不论是前者还是后者，均包括了多个逻辑存储区，它们通过不同的名称来区分。每个逻辑存储区可以保存多个数字证书。此外，可以通过 X509Certificate2Collection 类在当前存储区中添加删除证书。具体的访问证书存储区的方法如下图 4-24 所示。

```
private X509Certificate2 GetCertificate(string CertName)
{  //声明X509Store对象，指定存储区的名称和存储区的类型
    //StoreName中定义了系统默认的一些存储区的逻辑名称
    X509Store store = new X509Store(StoreName.My, StoreLocation.CurrentUser);
    //以只读的方式打开这个存储区，OpenFlags定义的打开的方式
    store.Open(OpenFlags.ReadOnly);
    //获取这个存储区中的数字证书的集合
    X509Certificate2Collection certCol = store.Certificates;
    //查找满足证书名称的证书并返回
    foreach (X509Certificate2 cert in certCol)
    {
        if (cert.SubjectName.Name == "CN=" + CertName)
        {
            store.Close();
            return cert;
        }
    }
    store.Close();
    return null;
}
```

图 4-24 访问证书存储区方法

对比以上 3 种方法，方法 3 是最安全可靠的方法，安全性得到了最好的保障，但是操作起来较为繁琐。方法 1 最易实现和操作，但是安全性不高。由于本系统的加密部分是在本地使用，所以可以将方法 2 作为一个折中的方法，该方法较好的统一了实用性和安全性。所以本系统采用方法 2 作为 RSA 算法的密钥保存方法。

4.3.3 自动混合加密解密

在该系统中，对称加密算法的密钥内嵌在程序内，并且固定不变，这就使得安全性有所降低，并且灵活程度也有所下降。

而 RSA 算法虽然比一般的对称加密算法安全，但是也有自身的局限性，例如对于明文长度有限制：明文的最大长度（字节数）=密钥的长度（字节数）-11，如果选用 1024bit 长度的密钥，那么明文的最大长度就是 117 字节。虽然

可以通过选用更长的密钥或将密文分组的方式来提高明文的最大长度，但是副作用是加密时间变的越来越长。所以 RSA 算法不利于长文本的加密，仅适合短文本的加密。

为了实现既能加密长文本，又能更好的保证其安全性，本系统采用一种混合加密的方法，即：用 DES 算法加密内容，用 RSA 算法加密 DES 算法的密钥。

4.3.3.1 基本原理

加密：用 DES 加密算法对明文进行加密，然后用 RSA 加密算法方法对 DES 算法所用的密钥进行加密。

解密：用 RSA 算法对 DES 密钥进行解密，然后使用该密钥用 DES 算法对密文解密。

4.3.3.2 加密步骤

1)随机生成 DES 加密密钥 K，每一个密钥只使用一次，以此来提高安全性。

2)利用生成的密钥 K 对明文 P 进行 DES 加密，生成密文。

3)利用 RSA 加密算法的公开密钥对 DES 的密钥 K 进行加密，形成 DES 加密的密钥 Ck，并保存为文件。

4.3.3.3 解密步骤

1)获取加密后的密钥 Ck。

2)利用 RSA 的私有密钥对 DES 的密钥 Ck 进行解密，形成 DES 解密的密钥 K。

3)生成明文 P，利用生成的密钥 K 对密文 C 进行 DES 解密，生成明文文本。

4.3.3.4 密钥的保存

为了实现加密解密过程的自动化，需要对两种算法的密钥进行保存。对于 DES 算法的密钥，被 RSA 加密后以文件形式保存，以生成二维码的文件名+扩展名作为文件名，扩展名为.key。对于 RSA 算法的密钥，将密钥保存到密钥容器 CspParameters 中。解密时，先从密钥容器 CspParameters 获得 RSA 算法解密所需私钥，然后从密钥文件中读取密钥密文，从而解密出 DES 算法所需密钥；然后使用 DES 算法解密出文本。

4.4 搜索部分主要模块

4.4.1 关键字匹配模块

关键字就是需要搜索的内容，是一个词语或者句子。关键字匹配是指关键字和搜索文本内容的匹配程度，匹配程度高于一定程度，系统就认为是匹配成功。实现关键字匹配的关键在于关键字匹配算法的实现，以及匹配成功的临界点的确定。可使用以下两种方法判断匹配。

1)关键字被包含：使用 String 类中的 Contains 方法，判断关键字是否被包含于被搜索字符串中，若包含，则匹配成功。但若要匹配成功只能是全部被包括，不支持模糊匹配。也就是说匹配度要么是 100%，要么是 0。

2)编辑距离算法：编辑距离，又称 Levenshtein 距离（也叫做 Edit Distance），指的是对于两个字符串，由其中一个变换成另一个所需最少的操作次数。编辑操作方式包括将一个字符替换成另一个字符、插入一个字符、删除一个字符。可以将编辑距离与字符串的长度进行某种运算，其结果作为源串与目标串的相似度。这种方法可以作为两个长度相近的字符串匹配程度计算的算法。

动态规划经常被用来作为这个问题的解决手段之一。其问题可以描述为：找出字符串的编辑距离，即把一个字符串 s1 最少经过多少步操作变成编程字符串 s2，操作有三种，添加一个字符，删除一个字符，修改一个字符。

算法过程可以如下描述。

1)str1 或 str2 的长度为 0 返回另一个字符串的长度。

2)初始化(n+1)*(m+1)的矩阵 d，并让第一行和列的值从 0 开始增长。

3)扫描两字符串（n*m 级的），如果：str1[i] == str2[j]，用 temp 记录它，为 0。否则 temp 记为 1。然后在矩阵 d[i，j]赋于 d[i-1，j]+1 、d[i，j-1]+1、d[i-1，j-1]+temp 三者的最小值。

4)扫描完后，返回矩阵的最后一个值 d[n][m]即是它们的距离。

5)计算相似度公式：1-它们的距离/两个字符串长度的最大值。

对于方法 1，存在的问题是不支持模糊匹配，只能全部匹配才算匹配成功。

对于方法 2，若相比较的两个字符串长度相差太大，会造成即使关键字被完全包含，算出来的相似度依然很小。

可以采用两种算法相结合的方式，只要其中一种匹配成功，那么就算匹配

成功。对于长度相差过大问题，解决方法是，将较长串分割为几个子串，然后再分别进行匹配即可。这样既提高了效率，又保证了一定的准确度。

编写方法将两种方法结合。如下图 4-25、4-26 所示。

```
public static bool Compare(string source, string target)
{
    bool CompareResult = false;
    Result LCompareResult = Operate.EditDistanceHelper.LevenshteinDistance(source, target);
    if (source.Contains(target) || LCompareResult.similarity > 0.5)
    {
        CompareResult = true;
    }
    return CompareResult;
}
```

图 4-25 字符串比较算法

```
public static bool Compare(string[] sources, string[] targets)
{
    foreach(string source in sources)
    {
        foreach(string target in targets)
        {
            if (Compare(source, target) == true)
            {
                return true;
            }
        }
    }
    return false;
}
```

图 4-26 多串匹配方法

其中 LevenshteinDistance(String,String) 是编辑距离匹配算法，返回值 Result 是一个结构体，包含编辑距离和相似度。在上图所示方法中，选择相似度大于 0.5 时就算匹配成功。

4.4.2 文件遍历模块

递归调用 C#提供的 string[] GetFiles(string path， string searchPattern)方法，实现文件遍历功能。其中参数 path 表示要搜索的目录。参数 searchPattern 表示要与 path 中的文件名匹配的搜索字符串。

4.4.3 字符串比对模块

对于搜索到的每一个图片文件，调用二维码解码模块的解码方法进行解

码，然后调用关键字匹配方法将关键字与解码后内容进行比对，返回匹配结果。

4.4.4 各模块间关系

各模块间逻辑关系可以描述为一个生产者-消费者模型，其中文件遍历模块为生产者，字符串比对模块为消费者，字符串比对模块等待文件遍历模块返回的结果，若返回，则进行解码，若未返回，则等待直到文件遍历模块返回新结果或者文件遍历模块执行结束。等到字符串比对模块执行结束，表示所有的搜索工作就结束了。他们的逻辑关系如下图 4-27 所示。

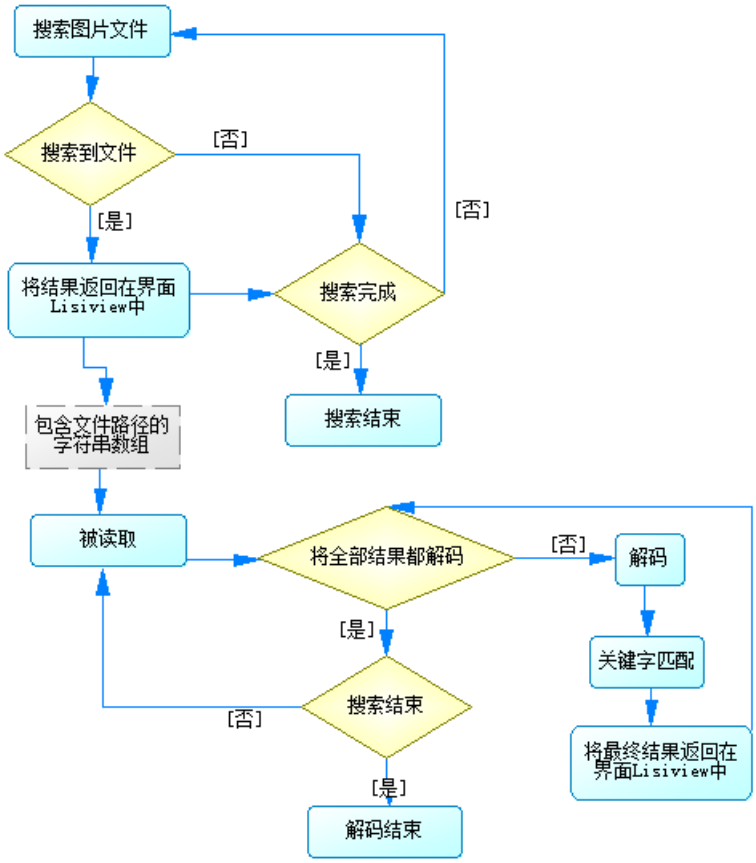


图 4-27 搜索部分各模块间的逻辑关系

由于他们有这样的逻辑关系，所以可以采用生产者-消费者模型进行多线程同步操作。执行搜索主线程所执行的方法如下图 4-28 所示。

```
public void Start()
{
    Searching = true;
    searchStart.Set();
    try
    {
        foreach (string SearchPattern in searchInput.SearchPatterns)
        {
            Thread searchThread = new Thread(()=>
            {
                GetFiles(Dir, Dir, SearchPattern);
                handler.Signal();
            });
            searchThread.Name = "搜索线程：" + SearchPattern;
            searchThread.IsBackground = true;
            searchThreads.Add(searchThread);
            searchThread.Start();
        }
        Thread searchTextThread = new Thread(new ThreadStart(FileContainsText2));
        searchTextThread.IsBackground = true;
        searchTextThread.Name = "解码线程";
        searchTextThread.Start();
    }
    finally
    {
        handler.Wait();
        Searching = false;
        searchFileFinish.Set();
        allFinish.WaitOne();
    }
}
```

图 4-28 执行搜索的方法

其中，searchInput.SearchPatterns 为搜索文件的扩展名集合，在本系统中，将其设置为：*.jpg，*.png，*gif，*.bmp；所以，搜索文件的线程一共 4 个，进行文本比对的线程 1 个。这是一个多生产者-单一消费者的模型，使用信号量和锁相结合的方法进行线程间同步。其伪代码如下图 4-29、4-30 所示。

```
private void GetFiles(string RootDir, string Dir, string SearchPattern)
{
    try
    {
        int resultNum = 0;
        搜索该目录下的匹配格式的文件；
        foreach (string File in files)
        {
            加锁；
            {
                在结果列表中增加搜索到的文件；
                在用户界面显示搜索到的结果（还没进行比对）
                发出搜索到新文件的信号；
            }
        }
        递归调用自身，搜索该目录的所有子目录
    }
}
```

图 4-29 搜索文件方法的伪代码

```
public void FileContainsText()
{
    等待搜索线程开始;
    for (int i = 0; ; )
    {
        start:
            加锁;
            {
                读取结果列表;
                如果没有新结果，则
                    如果正在搜索，则
                        等待搜索线程发来搜索到新文件的信号;
                        goto start;
                    或者搜索结束
                        通知搜索文件主线程全部结束;
                        自身结束;
                或者有新结果了
                    读取新结果;
            }
            对该文件解码;
            进行比对;
            更新比对结果到用户界面（完成了比对）;
            本次循环结束，i++;
    }
}
```

图 4-30 文本比对方法伪代码

4.5 批量生成部分主要模块

4.5.1 读取文件子模块

使用 C#自带的 FileStream 和 StreamReader 类即可完成文本文件的读取，将结果按行存入一个字符串的数组。

4.5.2 生成二维码子模块

调用二维码编码模块对结果字符串数组中的每一个字符串依次生成一个二维码图片，并保存为文件。

4.5.3 各模块间关系

与搜索部分的模块间关系几乎一样，也是一个生产者-消费者模型。其中读取文本模块是生产者，生成二维码模块为消费者，生成二维码模块等待读取文本模块返回的结果，若返回，则进行编码，若未返回，则等待读取文本模块返回新结果或者读取文本模块执行结束。等到生成二维码模块执行结束，表示

所有的批量生成工作就结束了。他们的逻辑关系如下图 4-31 所示。

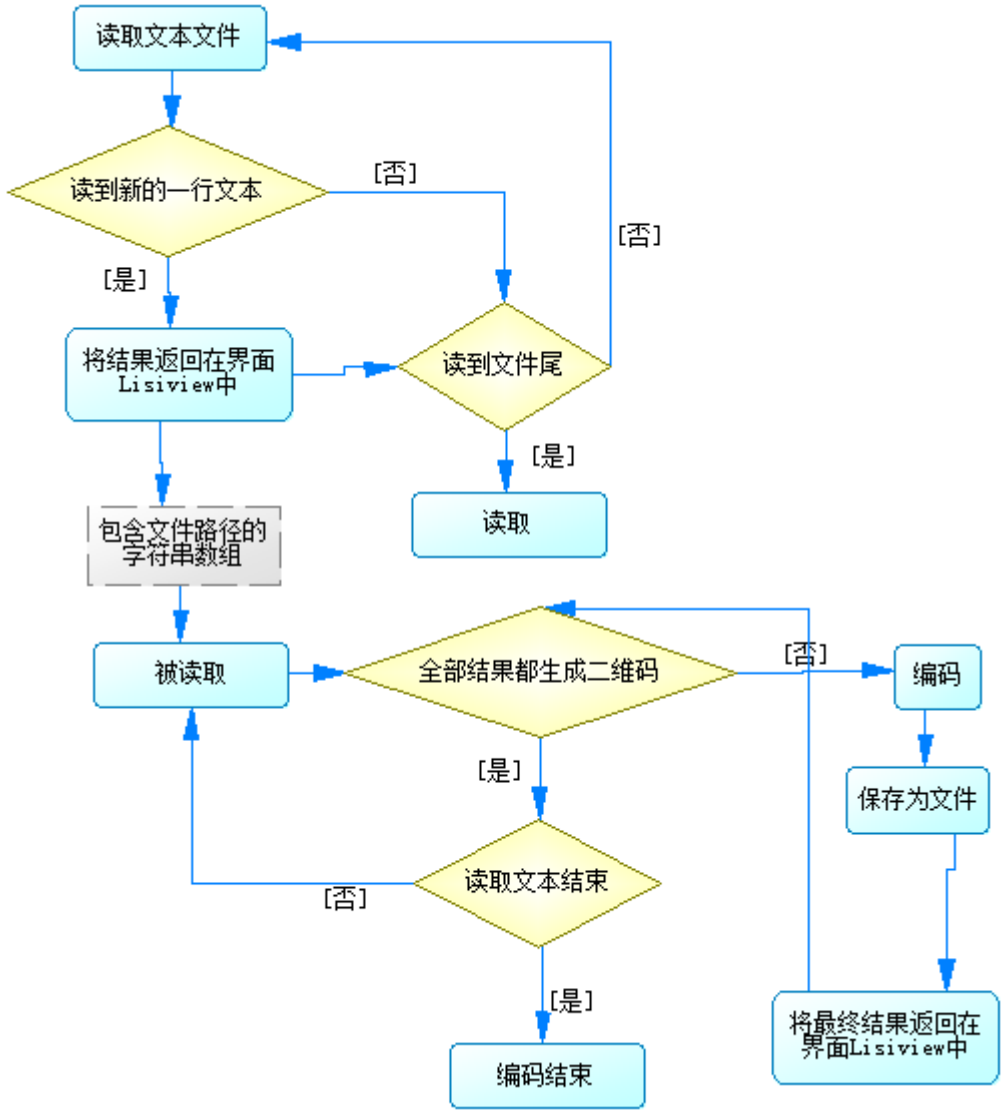


图 4-31 批量生成部分各模块间的逻辑关系

由于他们有这样的逻辑关系，所以可以采用生产者-消费者模型进行多线程同步操作。在批量生成的方法（如下图 4-32 所示）读取文本文件的线程一共 1 个（如下图 4-33 所示），进行二维码编码的线程 1 个（如下图 4-34 所示）。这是一个单生产者-单消费者的模型，使用信号量和锁相结合的方法进行线程间同步。

```
public void Start()
{
    reading = true;
    readStart.Set();
    try
    {
        Thread readFileThead = new Thread(() =>
        {
            ReadFile();
            readLineFinish.Set();
            reading = false;
            handler.Signal();
        });
        Thread encodeThread = new Thread(() =>
        {
            EncodeAll2();
            handler.Signal();
        });
        readFileThead.IsBackground = true;
        encodeThread.IsBackground = true;
        readFileThead.Name = "读取文件线程";
        encodeThread.Name = "生成二维码线程";
        readFileThead.Start();
        encodeThread.Start();
    }
    finally
    {
        handler.Wait();
    }
}
```

图 4-32 批量生成主线程所执行的方法

```
public void ReadFile()
{
    打开文本文件;
    while (没读到文件尾)
    {
        读取一行文本;
        加锁
        {
            保存到结果字符串数组;
            发出读取了行新文本的信号;
            在用户界面显示搜索到的结果（还没生成二维码）;
        }
    }
    关闭文件读取流;
}
```

图 4-33 读取文本文件方法伪代码

```
public void EncodeAll2()
{
    等待读取文本线程开始;
    for (int i = 0; ; )
    {
        start:
            加锁;
            {
                读取结果列表;
                如果没有新结果，则
                    如果正在读取，则
                        等待读取线程发来读取到新行的信号;
                        goto start;
                    或者读取结束
                        通知批量生成主线程全部结束;
                        自身结束;
                或者有新结果了
                    读取新结果;
            }
            对该字符串编码;
            保存二维码图片;
            更新比对结果到用户界面（完成了生成）;
            本次循环结束，i++;
    }
}
```

图 4-34 二维码编码方法伪代码

4.6 本章小结

本章根据系统的功能分析，叙述了每个子功能的实现方法和主要算法，然后根据功能逻辑，确定了了各方法之间的调用关系。然后详细分析了各部分模块间的内在逻辑关系，并根据这个依次实现了各部分的功能，从而完成了系统的全部设计和实现工作。

在二维码方面，采用 QR 二维码格式，编码和解码的实现主要采用开源的类库，为了支持中文对其进行了一些改进。在加密方面，其中对称算法 DES 的实现来自开源的类库，RSA 算法的实现则是对系统已有类库进行了封装。针对这两种加密方法，本系统进行了相应的改进。例如，对 DES 加密后的字符串进行压缩。DES 和 RSA 算法的密钥保存问题进行了分析和解决。然后根据这两种算法的优缺点的分析，本系统实现了这两个算法相结合的混合加密方法。该方法结合了二者的优点，兼顾了安全性和方便性。在搜索方面，主要靠编辑距离算法的具体应用来进行文本匹配，并且结合多线程同步技术，实现了文件遍历、解码、匹配等操作同步进行，保证了效率。在批量生成方面，主要是多线程同步技术的应用，实现了读取信息、编码、生成文件等操作的同步进行，保证了效率。

第 5 章 系统测试

5.1 编码解码部分

这一节主要介绍对二维码编码解码部分进行的测试。

5.1.1 无加密编码

打开界面，初始页面为编码页面，可以进行编码操作，如下图 5-1 所示。

图 5-1 软件运行后界面

其中，测试内容已经输入，编码选项是默认值，加密选项中算法是“无”表示编码时不会进行加密。

点击右下角“编码”，会生成一张二维码图片。运行结果如下图 5-2 所示。

图 5-2 点击编码后

然后，可以点击主界面下方的按钮“保存”，将该二维码图片保存为文件。在此处输入文件名为“无加密”，格式选择为“.jpg”。然后点击保存即可保存文件。

5.1.2 无加密解码

在页面标签中点击解码，即可跳转到解码页面。如下图 5-3 所示。

图 5-3 选择解码页面

点击打开，选择一张二维码图片的文件，可以进行解码操作。此处选择刚才生成的“无加密.jpg”，然后点击打开。该图片会显示在软件界面中。如下图 5-4 所示。

图 5-4 打开文件后

然后点击解码，即可对该二维码图片进行解码。如下图 5-5 所示。

图 5-5 点击解码后

解码后的内容会显示在内容框当中，与编码前的内容完全一致，说明在无加密情况下编码和解码功能都正常。

5.1.3 DES 对称加密后编码

在编码页面中的加密选项中，算法选择“DES”，密钥可以填写（需填写 8 位字符串）或者留空（将采用系统默认密钥），如下图 5-6 所示。

图 5-6 编码前进行加密设置

点击编码后，运行结果如下图 5-7 所示。

图 5-7 加密编码后

可以观察到，在内容框中显示的是，原文本进行 DES 算法加密并且进行 Base64 编码压缩后的字符串。然后点击保存，输入文件名为“DES.jpg”，保存此文件。

5.1.4 DES 对称加密后解码

在解码页面中，打开刚才保存的文件，对解密选项进行设置，算法选择“DES”，输入加密时的密钥。如下图 5-8 所示。

图 5-8 DES 加密图片解码前

然后点击解码，即可对该二维码图片依次进行解码、解压缩、解密操作。运行结果如下图 5-9 所示。

图 5-9 DES 加密图片解码后

解码后的内容会显示在内容框当中，与编码前的内容完全一致，说明在 DES 算法加密情况下编码和解码功能都正常。

5.1.5 RSA 非对称加密后编码

在编码页面的加密选项中的算法选择 RSA，密钥留空（因为密钥采用随机生成的，即使输入了也无效），如下图 5-10 所示。

图 5-10 RSA 加密编码前

此处待加密文本是“RSA 算法测试文本”，然后点击编码，如下图 5-11 所示。

图 5-11 RSA 加密编码后

可以观察到，文本内容长度变短了，但是加密并且压缩后长度超过 DES 算法加密压缩后的几乎一倍。此处将图片保存为“RSA.jpg”。

5.1.6 RSA 非对称加密后解码

在解码页面中，打开刚才保存的文件，解密选项中的算法选择 RSA，密钥留空。然后点击解码，结果如下图 5-12 所示。

图 5-12 RSA 加密后二维码解码

解码后的内容会显示在内容框当中，与编码前的内容完全一致，说明在

RSA 算法加密情况下编码和解码功能都正常。

5.1.7 混合加密后编码

在编码页面的加密选项中的算法选择混合，密钥留空，待加密文本是“混合加密算法测试”，进行编码、保存，文件名取为“混合.jpg”。与上述算法加密时不同的是，在保存二维码图片为文件后，还会生成一个存放密钥的文件。如下图 5-13 所示。

图 5-13 混合加密后生成的文件

5.1.8 混合加密后解码

在解码页面中，打开刚才保存的文件，解密选项中的算法选择混合，密钥留空。然后点击解码，结果如下图 5-14 所示。

图 5-14 混合加密后二维码解码

解码后的内容会显示在内容框当中，与编码前的内容完全一致，说明在混合算法加密情况下编码和解码功能都正常。需要指出的是，当密钥文件和二维码图片文件不在同一个目录时，如果进行解码就会失败。

将本例中的密钥文件用记事本打开后，里面的内容是：

SlnPwMVVOsAV5OzCn0W707Mr/1uqhw+qpR1DsfpRmIrFXU8s3SbfmWqoYU8
BbtlqnftdMHm74L53V+OBwAYpxo6uEDgtmLEh4RO9pBDj13A/ExDaemGTyyp

b9bh9oRZENyLlpYVHJqCVcjxRLeiydpwAhe65h1aInEyxxYioXNU=

这是 DES 算法的密钥进行 RSA 加密后的密文。

5.2 组织机构格式化

5.2.1 格式化

在编码页面有一个按钮为“组织机构格式”，其功能是将输入的文本格式化，如下图 5-15 所示。

图 5-15 点击组织机构格式后

可以看到，内容中的文本由“400009127（换行）北京理工大学（换行）胡海岩（换行）本科，研究生，博士生”变为了“400009127##北京理工大学##胡海岩##本科，研究生，博士生”。原来的每一行用“##”分隔。

5.2.2 识别组织机构

如果识别后的内容为组织机构格式，可以点击“识别组织机构”进行分栏识别。如下图 5-16 所示。

图 5-16 点击“识别组织机构”后

可以观察到此时组织机构一栏中，已经将内容中的信息分别填写到了对应的文本框中。

5.3 查找

切换至查找页面，在待查内容中输入“北京”（或者点击打开，选择一个二维码图片打开），然后点击搜索，即可开始查找操作。如下图 5-17 所示。

图 5-17 查找含有“北京”的二维码

单击结果列表中的一项，可以在“所选二维码内容”后看到该二维码中的内容；若双击该项，则可以打开该文件所在的文件夹，并且该文件被选中。如下图 5-18 所示。

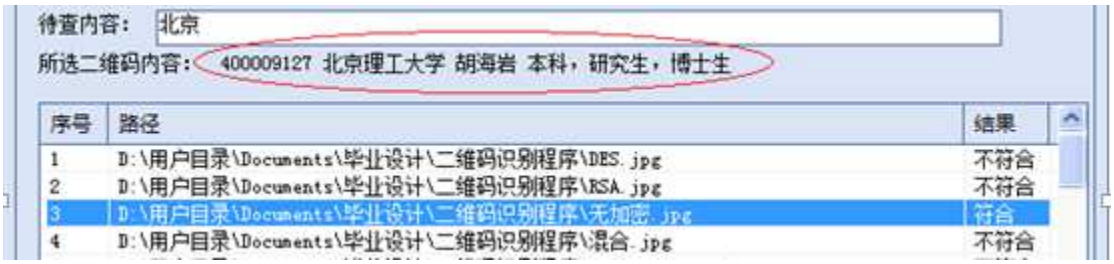
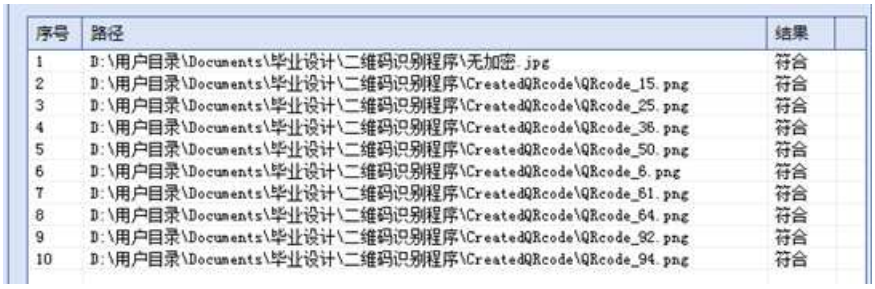


图 5-18 单击结果列表中的一项

若点击去除不符合项，则在结果列表中去除除了结果为符合的其他所有项。如下图 5-19 所示。



序号	路径	结果
1	D:\用户目录\Documents\毕业设计\二维码识别程序\无加密.jpg	符合
2	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_15.png	符合
3	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_25.png	符合
4	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_36.png	符合
5	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_50.png	符合
6	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_6.png	符合
7	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_81.png	符合
8	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_84.png	符合
9	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_92.png	符合
10	D:\用户目录\Documents\毕业设计\二维码识别程序\CreatedQRcode\QRcode_94.png	符合

图 5-19 点击去除不符合项后

通过以上测试，说明查找模块所包含功能都能正常使用并且结果符合预期。

5.4 批量生成

切换至批量生成页面，点击打开选择一个文本文件，该文本内含有多行信息。本例中选择打开“test100.txt”如下图 5-20 所示。

图 5-20 选择打开“test100.txt”

而 test100.txt 中含有的文本是 100 行组织机构信息。部分信息如下图 5-21 所示。

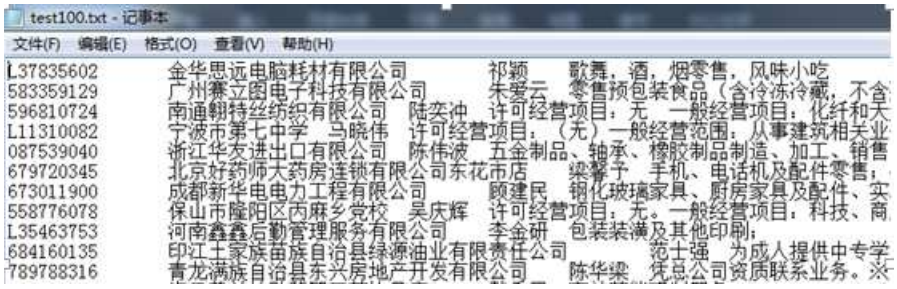


图 5-21 test100.txt 中的部分信息

然后点击生成：系统将文本按行读取，并按行尝试生成一个二维码文件。并在界面上显示结果。如下图 5-22 所示。

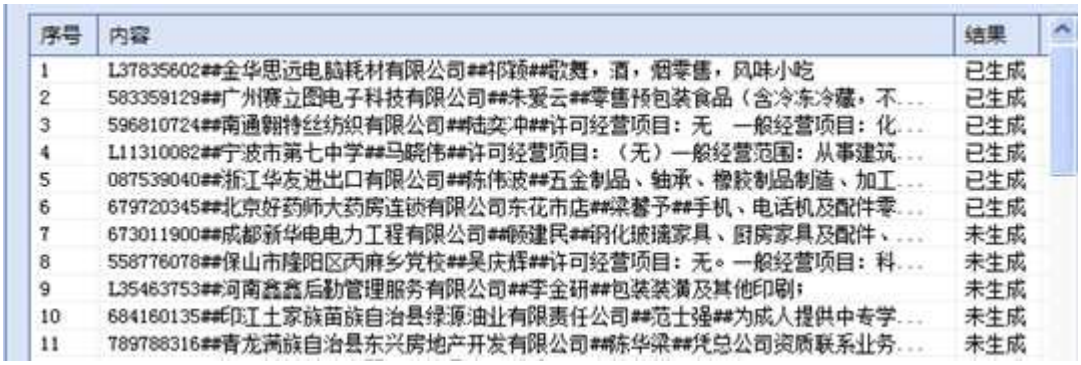


图 5-22 点击生成后

在 CreatedQRcode 文件夹下，可以看到生成所有的二维码，如下图 5-23 所示。

图 5-23 批量生成的二维码

通过测试，说明批量生成模块所包含功能都能正常使用。

5.5 存在的问题

问题 1：在进行查找或批量生成操作时，结果列表会一直闪烁，直到操作结束。

问题 2：在进行查找或批量生成操作未完成，用户可以进行页面切换，从

而有可能，进行其他操作，这有可能会造成异常，使程序崩溃。

5.6 改进方法

对于问题 1，重写一个支持双缓冲的 Listview，然后使用这个新的 Listview，将结果列表的 Listview 种类改为该新类型。如下图 5-24 所示。

```
public class DoubleBufferListView : ListView
{
    public DoubleBufferListView()
    {
        SetStyle(ControlStyles.DoubleBuffer
            | ControlStyles.OptimizedDoubleBuffer
            | ControlStyles.AllPaintingInWmPaint, true);
        UpdateStyles();
    }
}
```

图 5-24 支持双缓冲的 Listview

对于问题 2，重写一个可以禁止切换页面的 TabControl 用来禁止系统进行操作时进行页面切换，并且将页面内的按钮设置为不可点击状态直到操作结束。在进行查找等耗时长的操作时，利用此方法可以在 TabControl 通过鼠标点击切换页面的时候进行判断，若允许切换才可以进行切换，否则取消操作。如下图 5-25 所示。

```
public class NewTabControl : TabControl
{
    public bool AllowSelect;

    public NewTabControl()
    {
        AllowSelect = true;
    }

    protected override void OnSelecting(TabControlCancelEventArgs e)
    {
        e.Cancel = !AllowSelect;
    }
}
```

图 5-25 可以禁止切换页面的 TabControl

5.7 改进后效果

对于问题 1，结果列表在刷新的时候不再闪烁，问题得到解决。

对于问题 2，在进行查找等耗时长的操作时，点击其他页面的标签是没有任何效果的，直到该耗时长的操作结束，才可以进行其他的操作。

5.8 本章小结

本章介绍了对于该系统实现后各功能的测试，结果是各功能均能正常使用。并且对于测试过程中系统中存在的小问题和不足之处，尝试了相应的解决方法。然后对改进后的效果再次进行测试。

总结与展望

全文总结

本文首先介绍了组织代码中心的对于新系统中关于信息识别子系统的需求，然后介绍了二维码，以此来作为实现该需求的基本载体。结合当前二维码的使用情况，以及对使用过程中的安全隐患进行了简单的分析，得出二维码需使用加密的方式来规避信息泄漏的问题，同时这一措施也起到了组织机构代码证的防伪的重要功能。

基于以上基本的设想，本文接下来细致分析了该系统所应该具有的功能，并对各功能进行了细致的描述。大致主要分为二维码编码解码，文本加密，文本压缩，二维码搜索，二维码批量生成等部分。在确定了开发环境之后，系统设计实现部分总共分为三个阶段，分别是功能逻辑设计，界面设计，以及功能实现。

在功能逻辑设计部分，首先从用户的角度确定了各功能的用户操作和对应的系统响应的逻辑，基本可以概括为确定用户的输入和系统的输出。接着确定了在接受了用户输入后，系统内部的处理逻辑，也就是确定如何正确的输出。在界面设计部分，结合用户操作逻辑，将不同功能分到不同的页面，然后设计相应的功能页面。最后进行了界面的美化。在最终的功能实现部分，对每一个系统处理操作进行了相应方法的编写。然后结合功能逻辑设计，将这些方法进行相应的调用。从而逐步实现每一个功能。在二维码搜索，二维码批量生成部分，对多线程同步操作进行了应用，以此来提高效率。

在系统实现之后，进行了系统各功能的测试，对比实际输出和期望输出，得出了测试结果。然后整理在测试中发现的系统的问题和不足，并尝试进行解决，然后再次测试，得出最终结论。

后续展望

二维码作为一种流行的信息载体，还有非常多值得继续研究的地方。例如：容量的扩充，目前流行的二维码格式容量不过 2000 个字符，如何对该格式进行优化或者探究新格式以达到扩充容量的目的，这是一个很有意义值得深究的课题。另外，提高容量的过程中，或许会带来识别准确度、识别速度降低的副作用，如何规避这个问题，使得二维码能够容量、识别准确度、识别速度三方面兼顾，也是需要仔细考虑的问题。如果这方面有所突破，那么对于图片信息

传递效率又是一种突破。另外，二维码也有很多变种，例如内嵌 Logo 的二维码，彩色二维码。这对于二维码内容形式的丰富也是一种拓宽。其中对于彩色二维码，颜色或许可以作为一种信息的载体，对于容量扩大或许有深远的意义。当然，这或许又将带来了识别效率下降的问题。本文对于这些方面没有机会进行探究，不能不说是一种遗憾。

另外，由于二维码已经用于身份识别、快捷支付等领域，其安全性和保密性应越来越受到人们重视。如何将加密与二维码结合的更好，在保障安全性的同时又能不失去二维码的灵活性，虽然本文对这方面提出了解决方案，但是还有待改进，比如结合加密和压缩手段后，时间上会有额外开销，本文并没有对这部分开销进行探究。另外，加密和压缩算法的研究也不够多，对于这些算法和二维码的结合程度还需要进行大量实验，从而得出最优方法。

而关于二维码管理，虽然不是本系统的重点，但是也非常值得以后进行功能的拓展和现有功能的优化，例如文本匹配算法的优化，批量生成速度的优化等等。

另外考虑到方便性，手机端的实现也是很有必要的，虽然编写本系统的语言 C#的移植性受限于.net framework 环境，目前主流的安卓系统和苹果系统都还无法直接移植，但好在有 MonoDevelop 的出现，使得 C#跨平台移植变得可能，虽然比较繁琐，但是依然值得去探究。

本系统从大体上来说基本达到了当初的设想，但是还有有很多功能尚待开发和完善。

致谢

转眼间已经到了六月份，大学四年学习时光已经接近尾声，历时三个多月的毕业设计也完成了。这次毕业设计让我获益匪浅，通过查资料，学习理论，在编程中实践，在完成后测试等一系列过程，全方位的拓展了我的知识面，提高了我的分析解决问题的能力，和综合运用各种资源的能力。我过去学习过程中的不足之处和缺漏也得到了改正。伴随着毕业设计的圆满完成，我的本科生活也将画上一个完美的句号，在这里，我要衷心感谢帮助过我的人们。

首先，我要感谢我的毕业设计导师商建云老师。在我的毕业设计过程中，商老师一路给我指导和指正。最初面对毕设的时候，我心中充满了疑问，不知该从哪里着手去开始，是商老师给了我指导性的建议，使得我原本混沌的思绪逐渐变的明朗，让我知道了完成毕设的清晰脉络。在完成的过程中，商老师总是很耐心的指出我的不足，并且提出了修改意见，这让我进步很快，并且攻克了一个又一个阶段性的难题。老师的耐心指导和精益求精的治学态度让我非常钦佩。另外我要感谢张华平老师，张老师知识渊博、眼界开阔、思路新颖，对我毕设完成的系统的实现提出了很多巧妙的解决方法，让我的视野得到了很大的拓宽。我还要感谢我的学长王琦和吕笑，以及同一毕设小组的同学们，由于他们的帮助和鼓励，我的工作少走了很多弯路，也让我在遇到困难的时候能够有信心克服。

借此机会，在这里我向给予了我莫大帮助和指导的各位老师和关心我的同学们表示深深的感谢与敬意。同时，谨向审稿的老师们表示衷心的感谢！

参考文献

［1］张育绮．二维码营销［M］．北京：中信出版社，2013．7-40．
［2］中国物品编码中心．QR Code 二维码技术与应用［M］．北京：中国标准出版社，2002．23-95．
［3］杨波．现代密码学（第二版）［M］．北京：清华大学出版社，2007．1-80．
［4］(美)萨尤得．数据压缩导论（第 4 版）［M］．北京：人民邮电出版社，2013．80-98．
［5］张俊林．这就是搜索引擎：核心技术详解［M］．北京：电子工业出版社，2013．80-115．
［6］汤小丹．计算机操作系统（第三版）［M］．西安：西安电子科技大学出版社，2007．40-80．
［7］(美)沃森(Watson, K.) 等．C#入门经典（第 6 版）［M］．北京：清华大学出版社，2014．90-110．
［5］段钢．加密与解密（第三版）［M］．北京：电子工业出版社，2008．60-80．
［8］严蔚敏，吴伟民．数据结构（第二版）［M］．北京：清华大学出版社，2012．30-90．
［9］(加)斯廷森．密码学原理与实践（第三版）［M］．北京：电子工业出版社，2009．40-95．
［10］吴乐南．数据压缩（第 3 版）［M］．北京：电子工业出版社，2012．60-73．
［11］中国标准出版社．组织机构代码数字档案管理与技术规范［M］．北京：中国标准出版社，2012．10-69．
［12］(英)萨默维尔．软件工程（第 9 版）［M］．北京：机械工业出版社，2011．79-115．
［14］中国物品编码中心，中国自动识别技术协会．条码技术基础［M］．北京：武汉大学出版社，2008．1-31．
［15］(美)卡茨，(以色列)耶胡达·林德尔．现代密码学-原理与协议［M］．北京：国防工业出版社，2012．50-110．
［16］(美)Christian Nagel，Jay Glynn，Morgan Skinner ．C#高级编程（第 8 版）［M］．北京：清华大学出版社，2013．50-210．
［17］庞明．物联网条码技术与射频识别技术［M］．北京：中国物资出版社，2011．60-100．
［18］薛红．条码技术［M］．北京：中国轻工业出版社，2008．10-70．
［19］陈丹晖，刘红．条码技术与应用［M］．北京：化学工业出版社，2011．57-109．
［20］梁栋．加密与解密的艺术（第 2 版）［M］．北京：机械工业出版社，2013．70-160．