第一章 STM32 入门

参考资料：STM32 Reference Manual (RM0008)

1 STM32 简介

STM32 是由意法半导体推出的一款基于 ARM® Cortex - M 系列内核的高性能 32 位单片机。目前有 STM32F0，STM32F1，STM32F2，STM32F3，STM32F4，STM32F7，STM32L0，STM32L1，STM32L4 九个产品线。其中“STM32”表示基于 ARM®核心的 32 位微控制器，其余各个参数的差异，如表 1 所示。

名称	核心	产品特性
STM32F0	ARM® Cortex - M0	通用型产品
STM32F1	ARM® Cortex - M3	
STM32F2	ARM® Cortex - M3	
STM32F3	ARM® Cortex - M4	
STM32F4	ARM® Cortex - M4	
STM32F7	ARM® Cortex - M7	
STM32L0	ARM® Cortex - M0	超低功耗产品
STM32L1	ARM® Cortex - M3	
STM32L4	ARM® Cortex - M4	

表 1　STM32 系列微控制器产品线

根据 STM32F1 系列单片机的 Flash 大小，可以分成不同容量的产品，具体分类如表 2 所示。

小容量（LD）	16~32K
中等容量（MD）	64K~128K
大容量（HD）	256~512K

表 2　STM32F1 系列微控制器容量分类

通常在芯片选型的初期，首先要看数据手册以评估该产品是否能够满足设计上的功能需求；在基本选定所需产品后，需要察看技术参考手册以确定各功能模

块的工作模式是否符合要求；在确定选型进入编程设计阶段时，需要详细阅读技术参考手册获知各项功能的具体实现方式和寄存器的配置使用。 在设计硬件时还需参考数据手册以获得电压、电流、管脚分配、驱动能力等信息。

关于 STM32F1 系列产品命名规则，如图 1 所示。

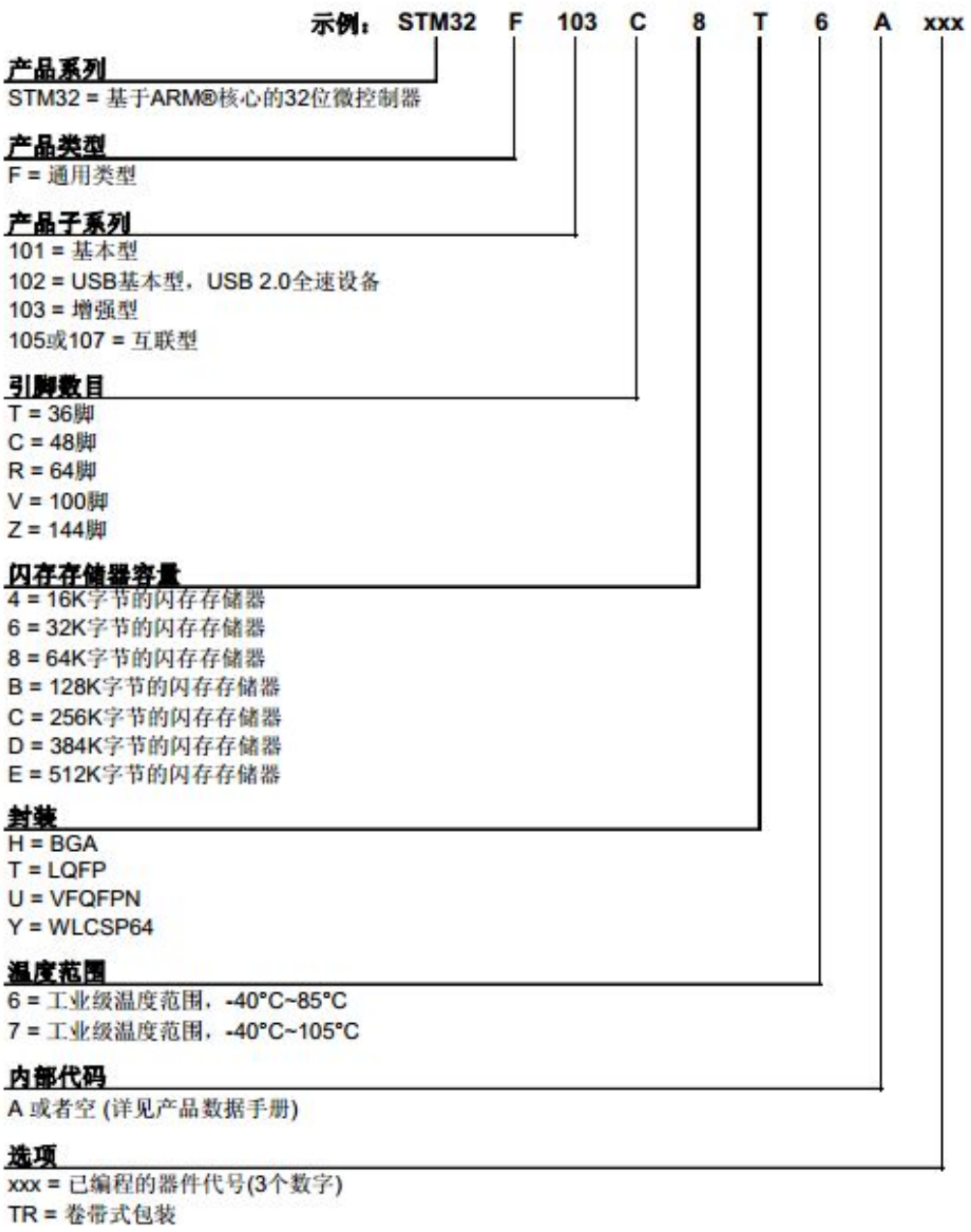


图 1　STM32F1 系列单片机命名规则

2 STM32F103 单片机内部外设（Peripheral）

2.1 ARM 内核处理器的设计流程

ARM 是全球领先的半导体知识产权（IP）提供商。他们致力于先进 RISC（精简指令计算机）内核技术的研发，并且以知识产权出售的形式，向各大半导体厂商出售 RISC 机内核代码（工业主流为 Verilog HDL 语言编写的 IP 核）。各大半导体厂商在标准的 ARM 内核基础上，外加各种外设，就组成了市场上各大类型的 ARM 内核微控制器（微处理器）。

当前的 ARM 系列内核，主要有 Cortex®-A，Cortex®-R，Cortex®-M 三个系列，Cortex 系列属于 ARMv7 架构。其中：“A”系列面向尖端的基于虚拟内存的操作系统和用户应用；“R”系列针对实时系统；“M”系列对微控制器。STM32 系列，即是基于 ARM Cortex®-M3 内核的单片机，除此之外，基于此内核的单片机市场上还有很多，因此只需要掌握一种，便可以一通百通。

关于 ARMv7 内核的具体介绍，本文就不详细介绍，推荐杜春雷的《ARM 体系结构与编程》，适当了解内核，对今后操作系统的学习，有些许帮助。

2.2 STM32 内部结构

STM32F103 内部集成了现在绝大多数的主流外设部件，如 USART，SPI，DMA 等等，不同产品系列的内部外设资源，如表 3 所示。

外设资源	STM32F103 小容量产品	STM32F103 中容量产品	STM32F103 大容量产品
GPIO 和 AFIO	√	√	√
中断和事件	√	√	√
DMA 控制器	√	√	√
ADC	√	√	√
DAC			√
高级控制定时器（TIM1 和 TIM8）	√	√	√

外设资源	STM32F103 小容量产品	STM32F103 中容量产品	STM32F103 大容量产品
通用定时器 TIMx	√	√	√
基本定时（TIM6 和 TIM7）			√
实时时钟 RTC	√	√	√
独立看门狗（IWDG）	√	√	√
窗口看门狗（WWDG）	√	√	√
灵活的静态存储器控制器（FSMC）			√
SDIO 接口（SDIO）			√
USB 全速设备接口（USB）	√	√	√
控制器局域网（bxCAN）	√	√	√
串行外设接口（SPI）	√	√	√
I2C 接口	√	√	√
通用同步异步收发器（USART）	√	√	√

表 3　STM32F103 系列微控制器内部外设

3 STM32F103 系统架构

STM32F103 在小容量、中容量和 大容量产品中，主系统由以下部分构成：

● 四个驱动单元：

— Cortex™-M3 内核 DCode 总线(D-bus)，和系统总线(S-bus)；

— 通用 DMA1 和通用 DMA2。

● 四个被动单元

— 内部 SRAM；

— 内部闪存存储器；

— FSMC；

— AHB 到 APB 的桥(AHB2APBx)，它连接所有的 APB 设备。

具体的系统架构，如图 2 所示

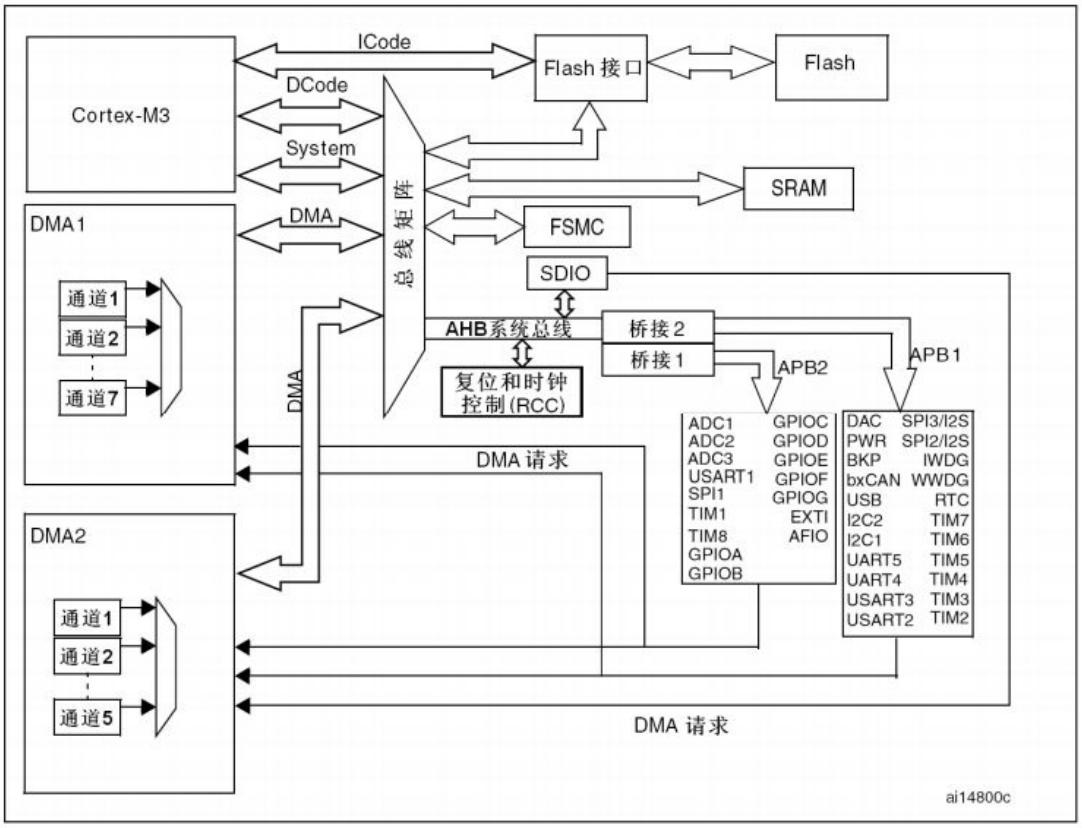


图 2 STM32F103 系统结构

在图 2 中，我们可以看到，Cortex-M3 核心，主要通过 ICode，DCode，System 三种总线，与外设进行连接。

ICode 总线直接连接到内部的 Flash 外设接口，指令预取在此总线上完成。

DCode 总线通过总线矩阵与闪存存储器的数据接口相连接(常量加载和调试访问)。

System 总线（系统总线）通过总线矩阵，协调着内核和 DMA 间的访问。

其他内部外设和 Cortex-M3 内核之间的数据和访问交换，都是需要总线矩阵进行访问仲裁。总线矩阵协调内核系统总线和 DMA 主控总线之间的访问仲裁，仲裁利用轮换算法。总线矩阵包含 4 个驱动部件（CPU 的 DCode、系统总线、DMA1 总线和 DMA2 总线）和 4 个被动部件（闪存存储器接口（FLITF）、 SRAM、 FSMC 和 AHB2APB 桥）。

DMA 总线将 DMA 的 AHB 主控接口与总线矩阵相联，总线矩阵协调着 CPU 的 DCode 和 DMA 到 SRAM、闪存和外设的访问。

AHB 外设通过总线矩阵与系统总线相连，允许 DMA 访问。两个 AHB/APB 桥在 AHB 和 2 个 APB 总线间提供同步连接。 APB1 操作速度限于 36MHz， APB2 操作于全速（最高 72MHz）。

JCT WORKROOM

4 基于 ARM® Cortex™-M3 架构的中等容量微处理器

- 内核： ARM 32 位的 Cortex™-M3 CPU

– 最高 72MHz 工作频率，在存储器的 0 等待周期访问时可达

1.25DMips/MHz(Dhrystone2.1)

– 单周期乘法和硬件除法

- 存储器

– 从 64K 或 128K 字节的闪存程序存储器

– 高达 20K 字节的 SRAM

- 时钟、复位和电源管理

– 2.0～3.6 伏供电和 I/O 引脚

– 上电/断电复位(POR/PDR)、可编程电压监测器(PVD)

– 4～16MHz 晶体振荡器

– 内嵌经出厂调校的 8MHz 的 RC 振荡器

– 内嵌带校准的 40kHz 的 RC 振荡器

– 产生 CPU 时钟的 PLL

– 带校准功能的 32kHz RTC 振荡器

- 低功耗

– 睡眠、停机和待机模式

– VBAT 为 RTC 和后备寄存器供电

- 2 个 12 位模数转换器， 1μs 转换时间(多达 16 个输入通道)

– 转换范围： 0 至 3.6V

– 双采样和保持功能

– 温度传感器

- DMA：

– 7 通道 DMA 控制器

– 支持的外设：定时器、 ADC、 SPI、 I2C 和 USART

- 多达 80 个快速 I/O 端口

– 26/37/51/80 个 I/O 口，所有 I/O 口可以映像到

16 个外部中断；几乎所有端口均可容忍 5V 信号

• 调试模式

– 串行单线调试(SWD)和 JTAG 接口

• 多达 7 个定时器

– 3 个 16 位定时器，每个定时器有多达 4 个用于输入捕获/输出比较/PWM 或脉冲计数的通道和增量编码器输入

– 1 个 16 位带死区控制和紧急刹车，用于电机控制的 PWM 高级控制定时器

– 2 个看门狗定时器(独立的和窗口型的)

– 系统时间定时器： 24 位自减型计数器

• 多达 9 个通信接口

– 多达 2 个 I2C 接口(支持 SMBus/PMBus)

– 多达 3 个 USART 接口(支持 ISO7816 接口，LIN， IrDA 接口和调制解调控制)

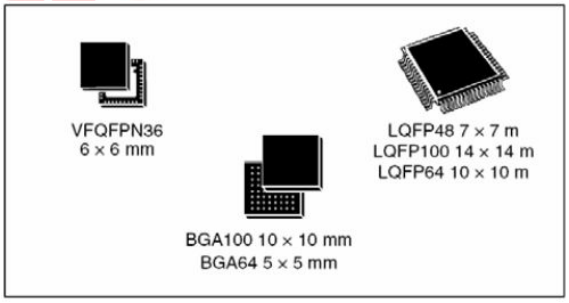
– 多达 2 个 SPI 接口(18M 位/秒)

– CAN 接口(2.0B 主动)

– USB 2.0 全速接口

• CRC 计算单元， 96 位的芯片唯一代码

• 封装



5 软件开发环境介绍

目前有很多工具可以进行 STM32 的开发，这些主流的工具有主要分成两类：1、独立的文本编辑器与 C 语言编译器的配合，例如“eclipse + arm-none-eabi-gcc”；2、集成的商业 IDE（Integrated Development Environment 集成开发环境），例如基于 RealViewMDK 的 Keil µVision x（最新版本 Keil µVision 5）以及基于 EWARM 的 Embedded Workbench for ARM （最新版本为 Embedded Workbench7）。第一种模式的开发环境，是属于 Linux 操作系统底下专有的开发环境，并且需要进行复杂的配置，我们会在以后的 Linux 课程中学习到。因此考虑到广大初学者，我们在这里只是基于 IDE 来开发。

RealView MDK 开发工具源自德国 Keil 公司，被全球超过 10 万的嵌入式开发工程师验证和使用，是 ARM 公司目前最新推出的针对各种嵌入式处理器的软件开发工具。RealView MDK 集成了业内最领先的技术，包括µVision 集成开发环境与 RealView 编译器。支持 ARM7、ARM9 和最新的 Cortex-M3 核处理器，自动配置启动代码，集成 Flash 烧写模块，强大的 Simulation 设备模拟，性能分析等功能，与 ARM 之前的工具包 ADS 等相比，RealView 编译器的最新版本可将性能改善超过 20%。

IAR Embedded Workbench 是一套用于编译和调试嵌入式系统应用程序的开发工具，支持汇编、C 和 C++语言。它提供完整的集成开发环境，包括工程管理器、编辑器、编译链接工具和 C-SPY 调试器。IAR Systems 以其高度优化的编译器而闻名。每个 C/C++编译器不仅包含一般全局性的优化，也包含针对特定芯片的低级优化，以充分利用您所选芯片的所有特性，确保较小的代码尺寸。IAR Embedded Workbench 能够支持由不同的芯片制造商生产，且种类繁多的 8 位、16 位或 32 位芯片。

有关于 IDE，MDK 和 IAR 各有千秋，主要双方的对比如下：

（以下为网络引用，原文地址：http://www.myir-tech.com/resource/508.asp）

（1）MDK 不支持层叠文件夹，在文件夹的下一级中必须为文件；IAR 支持层叠，可以比较方便管理代码，理清层次；

（2）MDK 连接 library，直接添加到文件夹即可；IAR 则需要从工程中选项中设置；

（3）IAR 中的相对文件路径，需要通过当前文件夹定位，即以“$PROJ_DIR$”开头

（4）MDK 默认只创建工程，工作区是不会直接创建。

（5）默认状态，MDK 的工具栏功能比较多，有点繁杂；IAR 的比较简洁，但相对，也比较单薄。

这两款 IDE 各有千秋，我亲身多年的使用经历，还是比较推荐 IAR 作为首选开发工具，主要原因有：1、IAR 的层叠文件夹，在建立大型工程的时候，简洁明了，逻辑清晰；2、IAR 在编译整个工程时候，速度比 MDK 快捷很多；3、目前 IAR 对 STM32 的支持已经非常完善了，举个例子，IAR 内部自带 STM32 固件库的启动文件。但是，总而言之，软件只是工具而已，优秀代码最终还是取决于开发者的水平。

JCT WORKROOM

6 IAR Embedded Workbench 安装

我们今后很长时间的学习开发都是基于 IAR IDE 的，但是基于某个 IDE 并不意味着代码不可移植，只要我们掌握了一种开发环境，并且代码足够优秀，那么移植到另一种真是几分钟的事情。因此我们这里只讲解 IAR 的安装和工程的建立，关于 Keil 的安装和工程建立，以及工程模板，我也会传上服务器，供大家下载和学习。

在正式的安装之前，我在这里还需要特别申明一点，我们乐创 DIY 学院和我要自学网是坚决拥护知识产权的，因此我们这里推荐大家使用正版软件，即使可以下载到破解版的 License，也不得用于商业用途，仅可作为自学交流。

IAR Embedded Workbench 7.2 的下载链接为：链接：http://pan.baidu.com/s/1cOMRM6k 密码：enp7。

具体安装流程如下：

（1） 双击 

图标，即可开始安装；

（2） 安装程序运行后，会出现图 1 的界面，单击图 3 中标红的选项，即可进行安装；

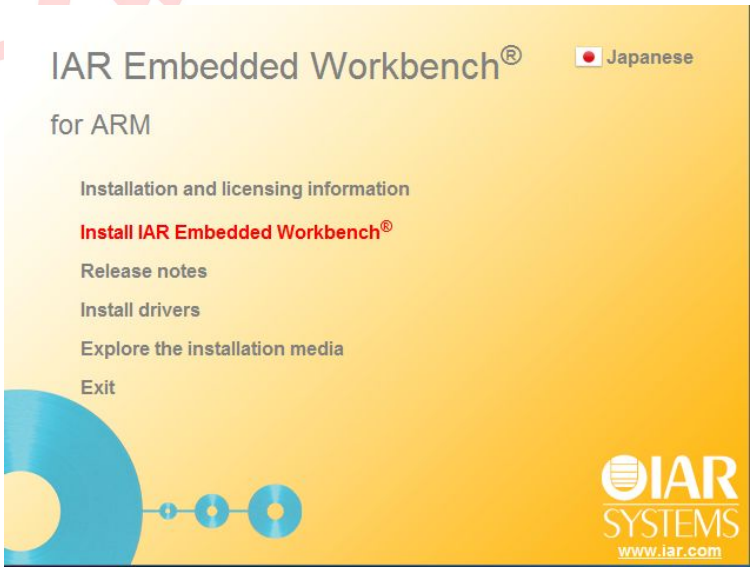


图 3 软件初始化

（3） 接下来，选定文件安装路径，直接点击“next”即可完成安装。

7 ST 单片机编程方式

相信很多学习过 51、AVR、PIC 或者 MSP430 的同学都会在脑海中形成一个固定的开发套路：设置寄存器->操作寄存器->写用户代码。STM32 当然也可以通过此方式进行程序编写，我们通常把这种编程方式成为“基于寄存器的编程方式”。“基于寄存器的编程方式”有利于大家更好地对一块单片机设备进行操作，但是这种方式也有以下几个不足：

（1） 移植性差。即当你想把 STM32F1 的程序移植到 STM32F0 的单片机上面时，由于两者内核已经内部外设有很多不同，因此，你需要修改很多地方才能完成移植；

（2） 效率低下。由于 STM32 整个系列的单片机具有非常多的外设，还有两个 8 位机从没有过的 DMA，因此如果基于寄存器编程时，需要设置的寄存器有很多，需要花很长时间来查询用户手册才能完成。

基于以上诸多问题，ST 公司推出了一个称为“基于固件库编程”的方式，即把所有 STM32 单片机的内部资源抽象成诸多函数，通过对这些函数的设置，即可完成相应固件的初始化以及操作。可以毫不夸张的说，STM32 可以这么广泛流行，很大的功劳归结于这个强有力的固件库。目前 STM32F10X 最新的固件库是“STM32F10x_StdPeriph_Lib_V3.5.0”，但可惜的是并不能和 STM32F0、STM32F3、STM32F4 公用，不过总体来说移植修改起来也很方便。因此我们课程中的程序都是采用基于固件库编程”的方式。我们这里着重讲的是“STM32F10x_StdPeriph_Lib_V3.5.0”。“STM32F10x_StdPeriph_Lib_V3.5.0”也在我要自学网上面可以下载。百度网盘 http://pan.baidu.com/s/1kTUPmgZ。

8 STM32 单片机的程序烧写以及 Debug

STM32 单片机支持 ISP 和 JTAG/SWD 两种编程方式。

ISP 即在线系统编程，使用时需要先设置 BOOT0 和 BOOT1 两脚，然后通过串口 1 进行程序烧写，烧写的软件为 Flash_Loader_Demonstrator。此方法的好处是，可以无需外加编程口，但是只可以烧写程序，无法进行在线仿真。

JTAG(Joint Test Action Group；联合测试工作组)是一种国际标准测试协议（IEEE 1149.1 兼容），主要用于芯片内部测试，早在 ARM7 上面就已经使用了 JTAG 作为调试和下载口。而 SWD 模式比 JTAG 在高速模式下面更加可靠。 在大数据量的情况下面 JTAG 下载程序会失败，但是 SWD 发生的几率会小很多。基本使用 JTAG 仿真模式的情况下是可以直接使用 SWD 模式的。

与 ISP 相比，SWD 接口可以与大多数 IDE 完美兼容，比如 IAR，并且可以通过 IAR 和调试器，进行程序的在线调试，这种调试方法可以直接追踪到当前代码处的各变量状态，寄存器状态等等，具体的使用方法我们会在以后的实践中详细解说。

关于 JTAG/SWD 的仿真器，市场上有好多，比如 ST 官方的 ST-Link,Segger 的 J-Link，这些编程器往往正版的很贵，在几千左右，网上盗版的也要 100 左右。为了使大家得到性价比高的开发套件，我们乐创软件工作室为大家开发了一套与 ST-Link 兼容的开发套件 Joy-Debugger。在后续的节目中，我们会为大家介绍。

微信扫码

JCT WORKROOM