工程师经验手记

轻松玩转
ARM Cortex-M4微控制器
——基于Kinetis K60

王日明　廖锦松　申柏华　编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

工程师经验手记

轻松玩转 ARM Cortex－M4 微控制器 ——基于 Kinetis K60

王日明　廖锦松　申柏华　编著

北京航空航天大学出版社

内 容 简 介

本书以野火 K60 开发板 V2 为实验平台，以 K60 的各个外设为主线，深入浅出地介绍了微控制器开发的各个步骤，重点强化嵌入式 C 语言、时序分析能力、寄存器配置思路、软件编程思想，力求让读者达到学一款微控制器而通各种微控制器的目的。

本书配套的例程还包含一些拓展实例，书中虽然没涉及此部分内容，但拓展例程都具有实用的参考价值，尤其适合参加智能车比赛的同学使用。本书的例程都是基于寄存器开发的，对于有简单的 C 语言基础的读者即可轻松上手此书。如果对书中内容有任何疑问，可以到野火初学 123 论坛交流(http://www.chuxue123.com)。

图书在版编目(CIP)数据

轻松玩转 ARM Cortex－M4 微控制器 ：基于 Kinetis K 60 系列 / 王日明等编著. -- 北京 ：北京航空航天大学出版社，2014.9

ISBN 978－7－5124－1537－9

Ⅰ. ①轻… Ⅱ. ①王… Ⅲ. ①微控制器－系统设计 Ⅳ. ①TP332.3

中国版本图书馆 CIP 数据核字(2014)第 094309 号

轻松玩转 ARM Cortex－M4 微控制器——基于 Kinetis K60

王日明　廖锦松　申柏华　编著

责任编辑　董立娟　陈　旭　敖惠珍

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191)　http://www.buaapress.com.cn

发行部电话：(010)82317024　传真：(010)82328026

读者信箱：emsbook@gmail.com　邮购电话：(010)82316524

北京楠海印刷厂印装　各地书店经销

*

开本：710×1 000　1/16　印张：29.75　字数：634 千字

2014 年 9 月第 1 版　2014 年 9 月第 1 次印刷　印数：3 000 册

ISBN 978－7－5124－1537－9　定价：59.80 元

前　言

目前微控制器的性能越来越好，集成的模块也越来越多，内部自带的寄存器越来越多，整体的架构也越来越复杂，因此库开发成为微控制器开发的主流。通过库开发，可以在不了解微控制器底层寄存器配置的情况下，快速地玩转单片机的各个模块资源。本书的目的就是带领读者深入了解开发 K60 库的过程，最终达到灵活快速上手其他人的库，甚至自行开发属于自己的库的目的。

笔者作为一名老师，每年都接触到各种各样零基础的学生。初学者学习微控制器，主要会遇到以下几种困难：

➢ **看不懂 Datasheet**

目前市场上大部分的微控制器书籍虽然都讲解寄存器的配置，但寄存器的说明都不按普通 Datasheet 的格式去排版，也没有介绍如何看 Datasheet，导致学生看不懂 Datasheet。例如寄存器里灰色表示不可操作或者保留，w1c 表示写入 1 就会清 0 而写入 0 是无效等，如果这些内容不加以说明，很多初学者就不知道原来有这么一回事。

➢ **看不懂时序图**

微控制器总是需要使用特定的时序来与各种外部模块进行通信，时序图最能描述时序的细节。很多书仅讲解寄存器的配置，直接忽略了时序图的讲解。如果初学者在开发中遇到通信失败的问题，那是硬件问题还是寄存器配置问题呢？唯一的解决方法就是测时序（示波器或逻辑分析仪），判断时序是否正确。但如果连时序图都看不懂，又怎么能看懂测出来的时序呢？

➢ **看不懂代码**

目前的 C 语言书基本上都是针对计算机系统的，几乎没有专门针对嵌入式系统的。而微控制器嵌入式的书几乎都仅局限于寄存器配置，其他必须的编程语言知识、编程思想几乎不讲。初学者不会无师自通，因此自然看不懂代码。

考虑到大部分的读者都缺乏嵌入式 C 语言的知识，本书的第 1 章就先给读者补充相关的嵌入式 C 语言知识。这些内容都是根据学生的问题和野火初学 123 论坛的网友咨询整理来的，相信大部分的读者都是第一次接触这些内容。

➢ **不懂调试**

程序跑飞了，如何解决？单步调试、看调用栈，这些都是基本的调试技巧。虽然

一般的教程都会介绍编译器创建工程的方法、工程设置的方法，但几乎都省略了编译器的调试工具介绍，似乎调试工具可有可无。虽然各个编译器的调试工具都大同小异，但初学者如果没接触过，则更需要学习调试工具来锻炼调试代码的能力。

➢ **不懂模块化编程**

随着 CPU 资源的丰富和需求的提升，嵌入式软件的复杂性越来越高，因此对开发人员的程序架构要求也越来越高。目前市场上几乎没有嵌入式的书籍会谈及模块化、软件分层这些内容，而这些却往往是项目开发的最基本要求。

模块化编程最常用的就是头文件，而头文件有什么用？别说是初学者，哪怕是工程师也难以准确地了解头文件的使用。一般也仅仅知道是用于函数声明、变量声明这类的，却不知道头文件是不参与编译的，对内容是没有要求的，仅仅在预处理阶段把内容插入到包含该头文件的相应文件里。

初学者是一张白纸，是不懂模块化编程的，而嵌入式这类书就应该在例程中讲解相应的知识。

➢ **不懂软件编程思想**

目前的嵌入式开发招聘，哪怕是不涉及 Linux 系统的纯单片机开发，一般也会注明会 Linux 系统者优先。原因就在于学习过 Linux 系统的人都会接触 Linux 系统分层的架构，了解应用层是如何通过硬件抽象层的 API 接口来调用硬件，学习设备注册（回调函数）、消息机制（队列）、线程机制（时间片）等思想，而这些思想都可以用在普通的单片机开发中。本书在编写代码的过程中已经融入了很多编程思想，例如按键的消息机制，状态机的思想，软件分层的思想等，力求让读者能够掌握常用的编程思想。

➢ **缺乏真正的项目经验**

既然是初学者，那么缺乏项目经验是必然的。作为教程的作者，如果没有把项目经验讲出来，那么会让初学者错失了一个学习开发技巧的机会。例如项目开发中最常用的是定时按键扫描，而不是查询按键扫描，但几乎全部的教程都仅讲解查询按键扫描，笔者也是直到参与真正的公司项目时才了解到原来还有这样一种按键扫描方式。

➢ **不懂微控制器的内存分布**

开发微控制器程序，如果连程序的内存存储位置都不知道，那么出现各种 bug 的时候都难以找出问题，因此嵌入式招聘的笔试面试几乎必考微控制器的内存分布。另外，目前市场上很多产品都是通过 ISP 或者 IAP 下载更新固件，或者动态从 SD 卡、nand Flash 里加载代码映像到 RAM 里运行相应的程序，这些功能都需要开发者对微控制器的内存分布十分熟悉。即使是初学者，一时间难以消化这些知识，但在初学阶段也应该了解有这么一回事，在后续的开发中慢慢领悟。

初学者在学习微控制器开发的时候，之所以觉得微控制器很难学，实际上是因为大部分的微控制器教程往往忽略了讲解微控制器的基础知识，如同没学会走路就来

学跑步那样。

本书采用 IAR 开发环境，全部例程都整合成代码库的形式，但目的不在于给读者简单地调用库，而是试图通过与读者一起编写代码库，介绍各种编程知识和思想，从而让读者在初学阶段就形成良好的编程习惯，具有良好的编程思想，进而让读者达到了解库的实现过程，快速地上手其他各种各样的代码库的目的。

本书以 Kinetis 系列 K60 微控制器外围模块为轴线，从简单的 GPIO 点亮 LED 来了解 K60 的编程步骤，到 GPIO 按键的定时扫描了解按键消息机制，再到 UART、I^2C、SPI 的时序分析学会看时序图、接着到系统时钟的设置和定时器的使用来熟悉微控制器的时钟模块，再学习模数转换模块、DMA 模块、Flash 模块、CAN 总线、外部总线 flexbus、SDHC 总线和 USB 总线等各种模块，从易到难逐步推进，中间补充各种相关的拓展知识，从而让读者熟悉库开发的各个细节。

配套资料

书配有全部案例的开发工具、完整源程序、数据手册、原理图，读者可到北京航空航天大学出版社网站(www. buaapress. . com. cn)的“下载专区”免费下载。

致 谢

首先，感谢野火初学 123 论坛的网友，他们不断地反馈使得本书的内容更加充实，更能解答初学者的疑惑。其次要感谢陈杏飞等人提供的翻译、对书稿内容的校正和建议。

由于本书内容涉及的知识面广，时间又仓促，限于笔者的水平和经验，疏漏之处在所难免，欢迎各位工程师、老师和读者批评指正，可以发邮件到 minimcu@foxmail. com 与作者进行交流，或者登录野火初学 123 论坛 http://www. chuxue123. com 进行讨论。

王日明 廖锦松
2014 年 8 月

目 录

第 1 章

ARM 嵌入式系统之路

1.1 嵌入式开发经验谈

1. 嵌入式技术知识结构

嵌入式技术是专用计算机系统技术，以应用为核心，以计算机技术为基础，软硬件均可裁减，适用在对功能、稳定性、功耗有严格要求的系统之中。嵌入式技术的开发人员需要对整个计算机体系从底层硬件到软件操作系统都有了解，而在这个体系之中，每个部分都可以分出一些小领域，因而对技术要求很高，如图 1-1 所示。

图 1-1 只是粗略地概括了嵌入式技术的知识结构，但从中已经可以看出它涉及的知识面非常多，难怪学生甚至技术人员总是迷茫。不少电子专业出身的嵌入式技术人员主要从事硬件抽象层（中间层）的开发，这一层是沟通嵌入式系统的硬件层和软件操作系统的桥梁，因而主要的工作是开发驱动程序、板级应用支持、协调软硬件的开发，因而对软硬件都要有深入的了解。

2. 嵌入式成长之路——从学生成为工程师

若希望从事硬件抽象层的开发，应该如何学习才能从学生成长为工程师呢？图 1-2 可以参考。从图 1-2 可以看出，越往后，就越接近于纯软件开发，但这并不代表嵌入式技术人员就不需要了解硬件，相反，上层的知识都是以下层为基础的，很多人说的做嵌入式软件开发，至少要读懂原理图就是这个道理。

3. 职业规划

在嵌入式技术领域的公司，除了工程师还分很多职业岗位。一般公司的研发部门职位如图 1-3 所示。一般需要 3～5 年过渡到下一级的岗位，小公司里项目经理一般也兼任部门经理。部门经理不一定要懂技术，并不是非由项目经理升职而成。直接与技术相关的是开发工程师和系统架构师，开发工程师会针对嵌入式技术的不同领域有不同的区分，在小公司里，熟悉软硬件的跨领域工程师很受欢迎，而大公司则区分明确，喜欢在某领域研究得深入的开发工程师。系统架构师需要熟悉整个嵌入式领域，能够协同不同领域的开发工程师进行项目开发。

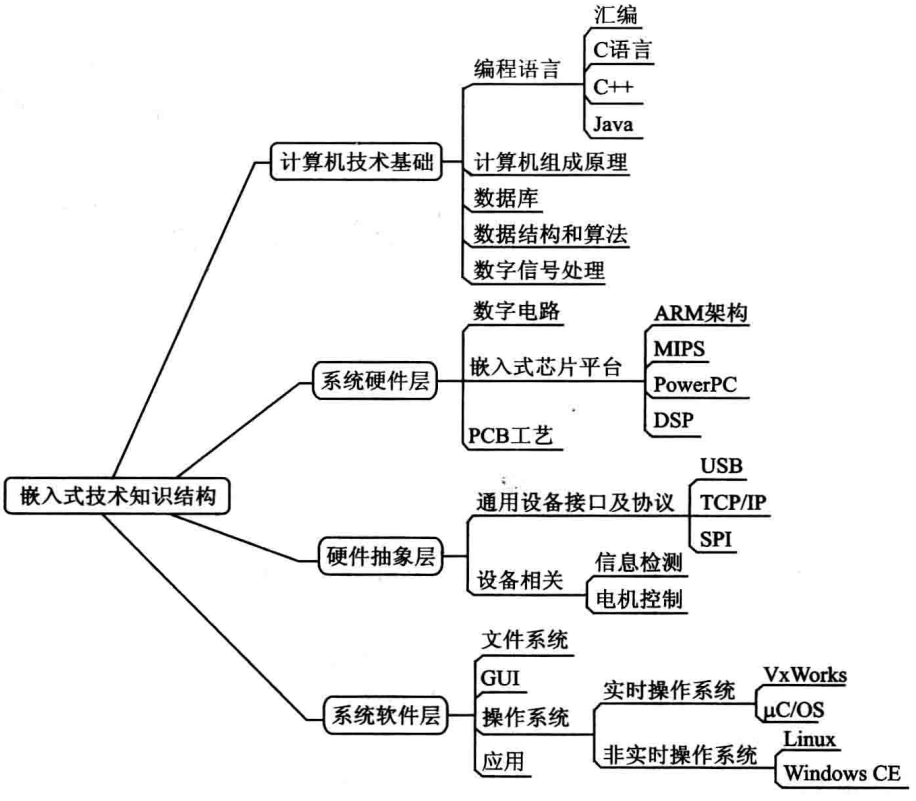

图 1－1　嵌入式技术知识结构

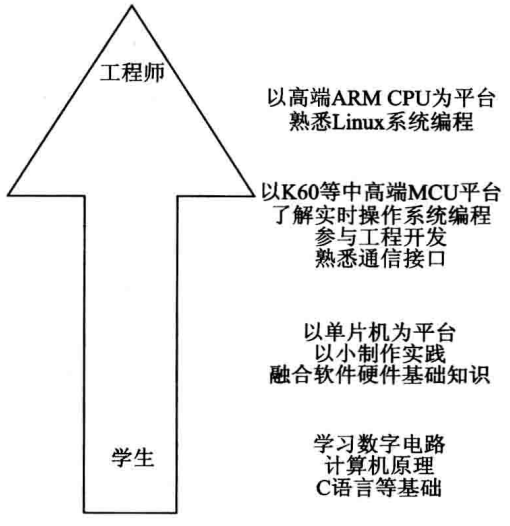

图 1－2　从事硬件抽象层开发的工程师成长之路

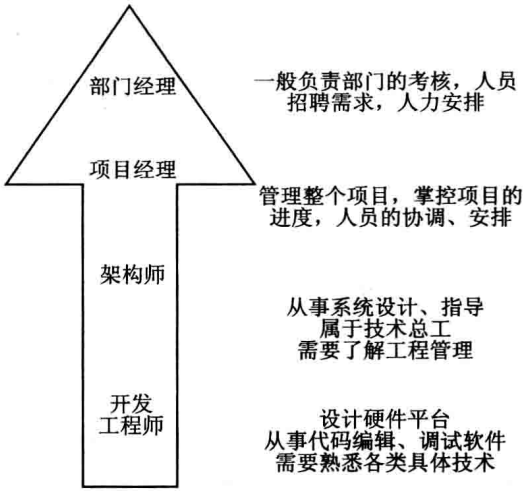

图 1-3 嵌入式技术人员职业成长

1.2 嵌入式开发进阶预备知识

本书的目的是力求用通俗易懂的方法带领初学者进入微控制器领域，学习嵌入式技术，因此，本书的读者对象为微控制器初学者，甚至没有接触过 51 系列单片机就直接来学习 ARM 的读者。

对于很多学习过 51 系列单片机的初学者而言，学习 51 后再来学习 ARM 时，都会有或多或少的恐惧，更不要说是没接触过 51 单片机的读者了。相比于 51，ARM 的模块很多，寄存器很多，用的编程语法也越来越陌生。当他们面对 ARM 的工程代码的时候就会难以接受，但当他们入门之后回头看，就会觉得 ARM 的编程太高效了、太便于移植与维护了。

可以说，在 51 编程里，整个 main 函数都是寄存器操作，那是很正常的事儿，甚至还有不少人在学 51 的时候还以为学会了 if-else 就是学会了 C 语言。但在 ARM 编程里，几乎全部都是软件分层与模块化编程，很少有人在 main 函数里进行寄存器操作，而且指针、枚举、结构体、宏定义、typedef 等语法用得非常多，在 51 编程里从来没见过的语法，在 ARM 编程里就是家常小菜。

另外，在 ARM 编程里，我们都会或多或少使用别人已经开发好的函数库来进行项目开发，而不再自己研究底层驱动、算法处理等，从而大大减少了开发时间，专注于自己的项目应用。例如 ARM 公司推出的 CMSIS 库、ST 公司的 STM32 驱动库、嵌入式实时系统 μC/OS、SD 卡上运行的文件系统库 FatFs，利用这些现有的函数库，我们可以从这些地方解放出来，专注于自己的项目开发。但这些函数库有个共同的特点是：库里的函数太多了，使学习难度大增。如果初学者不学好如何对待这些库，就

会产生恐惧，最终难以入门。

事实上，很多初学者对 ARM 感觉害怕、觉得难以入门，关键是基础不扎实，尤其是 C 语言知识和数据结构知识。为了让初学者更好地入门 ARM，本书专门针对初学者来讲解各种工程开发中常用到的知识点。

1.2.1 嵌入式 C 语言

目前，很多的 C 语言教材都针对的是计算机编程，而且内容非常浅显，很多工程开发用到的知识点都没讲到。为此，本书针对嵌入式软件的开发特点，讲解各种项目开发常用的知识点。

1. 关键字

很多人都学习了 C 语言，但又有多少人能熟练使用 C 语言里的关键字呢？能够明白每一个关键词的用法与作用呢？由 ANSI 标准定义的 C 语言关键字共有以下 32 个：

auto	double	int	struct	break	else	long	switch
case	enum	register	typedef	char	extern	return	union
const	float	short	unsigned	continue	for	signed	void
default	goto	sizeof	volatile	do	if	while	static

这里简单讲解各个关键字的用法。

(1) register 与 auto

相信很多读者都知道 MCU 的寄存器读/写速度远远快于 RAM 内存，换句话来说，对于一个需要频繁读/写的变量，将其放在寄存器里比放在 RAM 内存里的效率更高。为了提高效率，在 C 语言里，可以通过 register 关键字来声明变量，编译器编译的时候会尽可能地(因为 MCU 的寄存器是有限的，不一定能完全满足全部的要求)把变量放在寄存器里，例如：

```
register int i;
```

而 auto 关键字是用来声明自动变量的，由编译器自动优化，编译器一般默认为 auto(一般情况下都省略 auto)，例如：

```
auto int i;
```

等效于：

```
int i;
```

使用 register 须注意以下几点：

① register 变量必须是能被 CPU 寄存器接受的类型，这通常意味着 register 变量必须是一个单个的值，并且其长度应小于或等于整型的长度。但是，有些机器的寄存器也能存放浮点数。

② register 变量可能不存放在内存中，所以不能用取址符运算符“ & ”。

③ 只有局部变量和形参可以作为 register 变量，全局变量不行。

④ 静态变量不能定义为 register。

本来寄存器的数目就不多，如果全局变量和静态变量也可行，就意味着单片机运行全程中都少了一个可供使用的寄存器。

(2) continue、break 与 return

- continue：结束当前循环，开始下一轮循环。
- break：跳出当前循环。
- return：子程序返回语句，可返回值或者不返回值。

初学者容易犯的错误是分不清 continue 和 break 的区别，例如在 for、while 循环里，continue 是结束当前循环而已，并没有退出循环，而 break 则直接退出这个循环。

(3) extern 与 static

static 可用于修饰变量和函数，其中，修饰的变量可分为局部变量和全局变量，它们都存在内存的静态区。

① static 修饰变量

静态局部变量：出现在函数体内，生命周期是整个程序的执行过程，由于被 static 修饰的变量总是存在内存的静态区，即使该函数生命结束，其值也不会被销毁，同样要修改该变量，就要到函数内部完成，所以用起来比较安全，起到信息屏蔽的作用。

静态全局变量：出现在函数体外，作用域仅限于变量被定义的文件中，其他文件即使用 extern 声明也没法使用它。

② static 修饰函数

函数前加 static 修饰会使函数成为静态函数。此处 static 的含义不是指存储方式，而是指对函数的作用域仅局限于本文件(所以又称为内部函数)。使用内部函数的好处是：不同的人编写不同的函数时，不用担心自己定义的函数是否会与其他文件中的函数同名。

③ extern 声明外部定义

extern 的作用是声明函数和变量在外部定义，提示编译器遇到此函数或变量时在其他模块中寻找定义。注意，如果函数和变量定义时加了 static 修饰，那么即使用 extern 声明了外部定义，也不能在其他模块中调用此函数和变量。

(4) volatile 与 const

volatile 是 C 语言里的类型修饰符，本意是易变的。因为寄存器的读/写速度远快于内存单位，编译器一般都会把数据存放在寄存器而减少内存单位的读/写，从而有可能读取到脏数据，即错误数据。volatile 的作用准确来说是防止编译器对代码进行优化而导致没有执行指令或执行有误。

例如，在模拟时序的时候通常需要对 I/O 引脚输出高、低电平。假设没有 volatile 声明，例如如下的伪代码：

```
/ * volatile * / int * pPTA0_OUT = 0x400FF000u;      //pPTA0_OUT 指向 PTA0 的输出寄存器，
                                                     //注释了 volatile 的修饰
* pPTA0_OUT = 1;
* pPTA0_OUT = 0;
```

上面例程中最后两行的代码，不加 volatile 声明，编译器会认为两次对 0x400FF000u 地址进行写入操作，而且两次写入之间并没有读取该地址的数据，可认为第一次写入是无效的，则编译器就会忽略第一次写入的指令。因此，需要加入 volatile 来修饰，防止编译器对代码进行优化而忽略了这些指令。

一般说来，volatile 用在如下的几个地方：

① 在中断服务函数中需要访问的全局变量。

② 多任务环境中需要被多个任务共享的变量。

③ 硬件寄存器(例如：状态寄存器)。

const，只读变量(是变量，而不是常数)。编译时，如果直接尝试修改只读变量，则编译器会提示出错，就能防止误修改。对于非指针变量的修饰，const 的摆放位置可在数据类型的前或者后，两种摆放位置的意思都是一样的。如下面两条语句，两者的意思都是相同的。

```
const int a = 10;
int const a = 10;
```

对于指针变量的修饰，const 的摆放位置在数据类型的前和后的意思是不相同的，可参考下面的代码：

```
int me;
const int * p1 = &me;
          //p1 可变，* p1 不可变。const 修饰的是 * p1，即 * p1 不可变
int * const p2 = &me;
          //p2 不可变，* p2 可变。const 修饰的是 p2，即 p2 不可变
const int * const p3 = &me;
//p3 不可变，* p3 也不可变。前者 const 修饰的是 * p3，后者 const 修饰的是 p3，两者都不可变
```

前面说的直接修改 const 修饰的变量，编译器会报错。如果通过指针方式间接地修改，则编译器仅仅提示警告而不会报错。

```
int main()
{
    const int a = 10;
    int * b;
    b = &a; //此处编译器提示警告：C4090：'=' ：different 'const' qualifiers
    * b = 11;
    printf("a = %d\n",a);
    return 0;
}
```

volatile 易变，const 只读，那么能不能两者都用来修饰同一个变量呢？答案是肯定的，例如 I/O 端口的输入寄存器是只读的，数据是易变的。

(5) sizeof

sizeof 是 C/C++中的一个操作符(operator),作用就是返回一个对象或类型所占的内存字节数。sizeof 有 3 种语法形式:

① 用于数据类型。

```
sizeof( type_name );      //sizeof( 类型 );
```

② 用于变量。

```
sizeof ( object );        //sizeof( 对象 );
sizeof object;            //sizeof 对象;
```

根据上述的语法形式,可以容易判断下面哪个语句是错误的:

```
int i;
sizeof( i );              //ok
sizeof i;                 //ok
sizeof( int );            //ok   //一般建议用 sizeof(xxx)来避免错误
sizeof int;               //error
```

注意:sizeof 操作符不能用于函数类型、不完全类型或位字段。不完全类型指有未知存储大小的数据类型,如未知存储大小的数组类型、未知内容的结构或联合类型、void 类型等。

sizeof 是一个关键字,而不是函数。初学者比较容易搞混 sizeof 和 strlen 的区别。strlen 是 C 库提供的函数,用于计算有效字符串的长度,不包含'\0'。sizeof 是 C 语言的关键字,是一个运算符,用于计算占用空间的大小。

(6) typedef

typedef 用来为复杂的声明定义简单的别名,与宏定义有些差异。typedef 为 C 语言的关键字,作用是为一种数据类型定义一个新的别名,目的是给数据类型一个易记且意义明确的新名字,或简化一些比较复杂的类型声明。例如把 unsigned long 类型重命名为 uint32,那么程序员就容易知道 uint32 类型是 32 位的无符号整型,更易记且意义明确。

```
unsigned  long a;
```

等效于

```
typedef unsigned  long uint32;
uint32 a;
```

typedef 声明的方法如下:

① 先按定义变量的方法写出定义语句(如 unsigned long a;)。

② 将变量名换成新类型名(如将 a 换成 uint32)。

③ 在最前面加 typedef(如 typedef unsigned long uint32;)。

④ 然后可以用新类型名去定义变量(uint32 a;)。

例如把一个含 10 个整型元素的数组类型重命名为 array10,目的是简化比较复