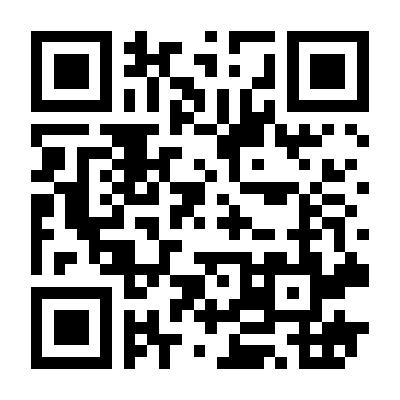
**Nerve RTOS开发指南**

**编辑：阿顿**

****

扫码，获开源代码

## NerveOS系统的历史版本

1. NerveOS1.0.0
2. NerveOS1.1.0
3. NerveOS1.5.1
4. NerveRTOS1.5.2
5. NerveRTOS1.5.2.5
6. NerveRTOS1.5.3
7. NerveRTOS1.5.4
8. NerveRTOS1.6.0
9. NerveRTOS1.6.7
10. NerveRTOS1.6.8

## NerveOS系统的开发人员

开发：阿顿

测试：阿顿

**手册信息**

|  |  |
| --- | --- |
| **编写日期** | **2021年12月** |
| **手册版本** | **第一版修订版** |
| **编辑** | **阿顿** |
| **审核** | **阿顿** |

# 0.目录

[NerveOS系统的历史版本 1](#_Toc95317729)

[NerveOS系统的开发人员 1](#_Toc95317730)

[0.目录 2](#_Toc95317731)

[1. 了解 4](#_Toc95317732)

[1.1什么是实时操作系统 4](#_Toc95317733)

[1.2 Nerve RTOS可以干什么 4](#_Toc95317734)

[1.3 Nerve RTOS的主要结构 4](#_Toc95317735)

[1.4工程文件结构 4](#_Toc95317736)

[1.5系统的主要文件 4](#_Toc95317737)

[1.6系统基本的外设支持 5](#_Toc95317738)

[1.7系统对内存的要求 5](#_Toc95317739)

[1.8 任务在系统中运行过程 5](#_Toc95317740)

[1.9 任务的类型 5](#_Toc95317741)

[1.10任务容量 5](#_Toc95317742)

[2.开始 7](#_Toc95317743)

[2.1 创建任务 7](#_Toc95317744)

[2.1.1创建任务时，注意事项 8](#_Toc95317745)

[2.2注销任务 8](#_Toc95317746)

[2.3手动启动任务 8](#_Toc95317747)

[2.4指定优先级启动任务 8](#_Toc95317748)

[2.5使用系统的延时功能 8](#_Toc95317749)

[2.6使用系统的区域保护功能 9](#_Toc95317750)

[2.6.1开启任务保护 9](#_Toc95317751)

[2.6.2取消任务保护 9](#_Toc95317752)

[2.7条件任务触发接口 9](#_Toc95317753)

[3.深入 10](#_Toc95317754)

[3.1了解基本的 10](#_Toc95317755)

[3.1.1系统的任务系统 10](#_Toc95317756)

[3.2系统的时钟系统 12](#_Toc95317757)

[3.3系统的移植与剪裁 13](#_Toc95317758)

[3.4系统的串口系统 13](#_Toc95317759)

[3.5绍系统串口的基本使用 14](#_Toc95317760)

[3.6其他文件的介绍 14](#_Toc95317761)

[3.6.1 Inc\_Basic.h文件 14](#_Toc95317762)

[3.6.2 Inc\_DataType.h文件 14](#_Toc95317763)

[3.6.3 Inc\_ActionValue.h文件 15](#_Toc95317764)

[3.6.4 NOP.h文件 15](#_Toc95317765)

[3.6.5 NOP.A51 文件 15](#_Toc95317766)

[3.6.6 STARTUP.A51 文件 15](#_Toc95317767)

# 了解

## 1.1什么是实时操作系统

实时操作系统（RTOS）是指当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统做出快速响应，调度一切可利用的资源完成实时任务，并控制所有实时任务协调一致运行的操作系统。提供及时响应和高可靠性是其主要特点。

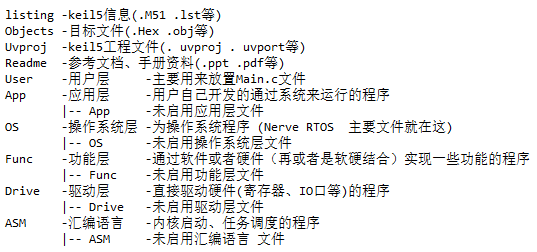
## Nerve RTOS可以干什么

主要实现多任务的运行

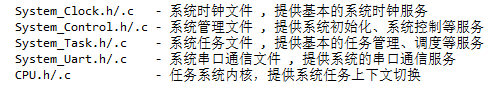
## Nerve RTOS的主要结构



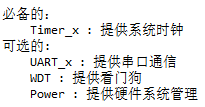
## 1.4工程文件结构



## 1.5系统的主要文件



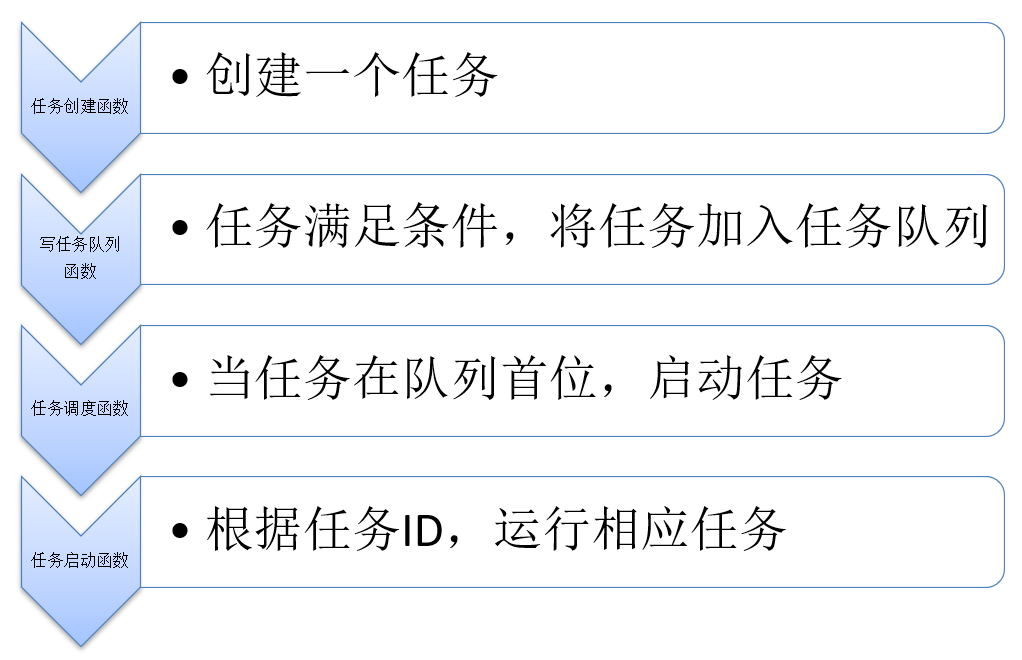
## 1.6系统基本的外设支持



## 1.7系统对内存的要求

|  |  |  |
| --- | --- | --- |
| 存储空间（FLASH/ CODE） | 随机存储器（RAM / DATA） | 扩展随机存储器（SRAM / XDATA） |
| 11Kbyte | 50Byte | 2200Byte |

## 1.8 任务在系统中运行过程



## 1.9 任务的类型

|  |  |  |
| --- | --- | --- |
| 类型 | 代号 | 说明 |
| 定时任务 | 0x00 | 定时触发 |
| 条件任务 | 0x01 ~ 0x1E | 中断或异常触发 |
| 空任务 | 0xFF | 手动触发 |

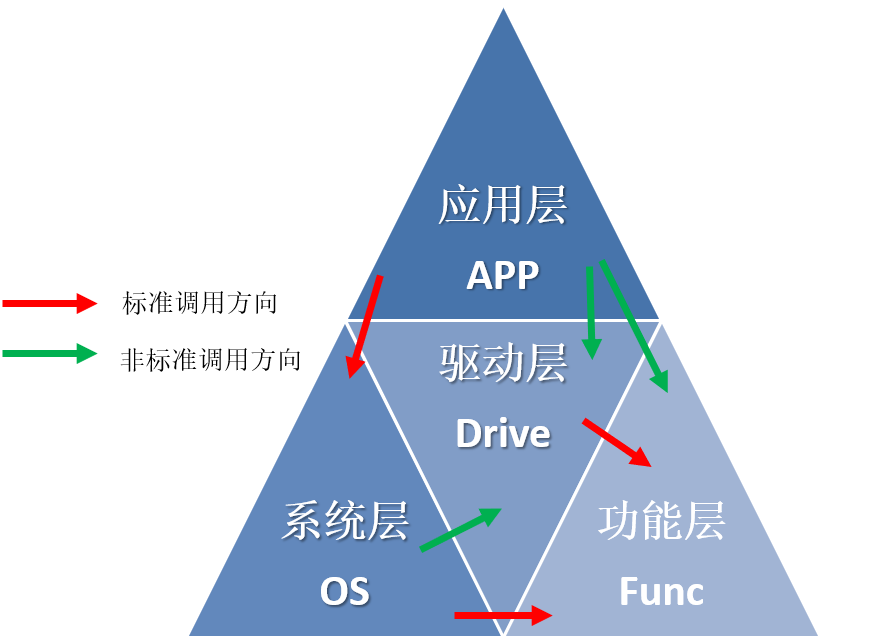
## 1.10任务容量

0x00 ~ 0x20 (共 32个任务)

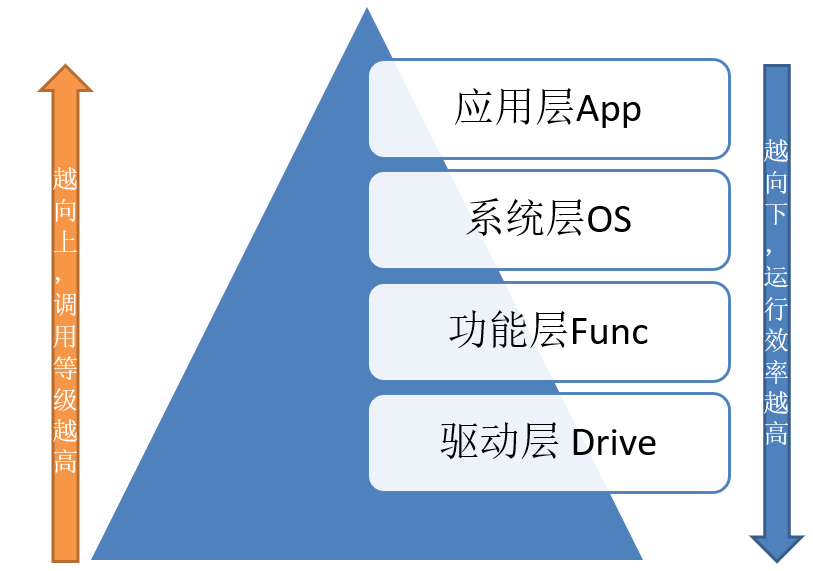
系统任务也占用了一些任务ID

|  |  |
| --- | --- |
| 任务ID | 用途 |
| 30 | 重置看门狗 |
| 29 | 系统重启并下载模式 |
| 28 | 串口接收完成 |
| 27（临时占用） | 上电测试 |
| 0 | Main函数 |

1.11各部分程序的调用关系与架构



上图为各部分之间的程序的调用关系



上图为系统基本架构

# 2.开始

## 2.1 创建任务

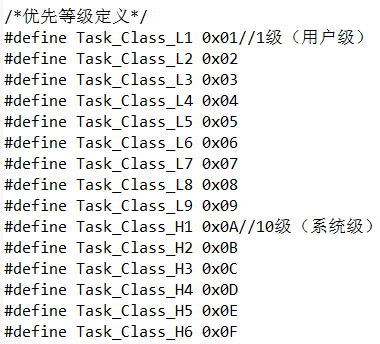
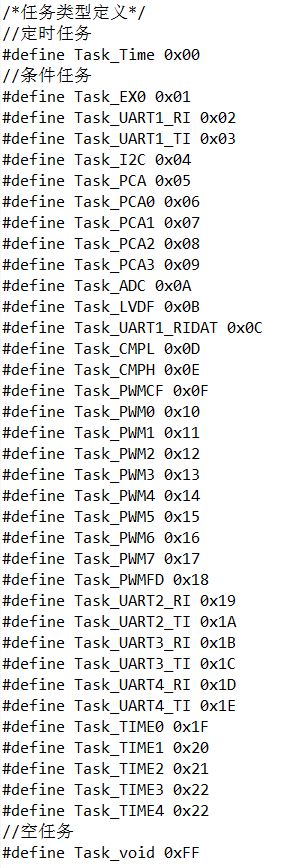
创建任务，需要System\_Task.h/.c文件中TaskLoad 函数

函数原型：TaskLoad(u8 TaskId,u8 TaskType,u8 PY,u16 TaskM,u16 TaskPC)

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| TaskId | 任务ID | 0x00 ~ 0xFF |
| TaskType | 任务的类型 | 0x00 ~ 0xFF |
| PY | 任务的类型优先级 | 0x01 ~ 0xFF |
| TaskM | 任务的参数 | 0x0000 ~ 0xFFFF |
| TaskPC | 函数地址 | 0x0000 ~ 0xFFFF |

示例： TaskLoad(1,0,1,10,test);//为test函数开启了，时间为10ms，优先级为1，ID为1定时任务

为便于标记与使用，任务的类型与优先级都已使用宏定义，具体内容如下图所示



任务优先级

注：系统级与用户级无区别

任务类型

使用宏定义，优化后的代码 : TaskLoad(1,Task\_Time, Task\_Class\_L1,10,test);

### 2.1.1创建任务时，注意事项

1.对于周期循环定时型的任务，单次周期时长不要低于10ms

2.对于实时性要求很高的程序，不建议通过系统来运行

## 2.2注销任务

注销任务，需要System\_Task.h/.c文件中TaskRemove函数

函数原型：TaskRemove(u8 TaskId)

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| TaskId | 任务ID | 0x00 ~ 0xFF |

示例： TaskRemove (1);//注销ID为1的任务

## 2.3手动启动任务

手动启动任务，需要System\_Task.h/.c文件中OpenTask函数

函数原型：OpenTask(u8 TaskID)

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| TaskID | 任务ID | 0x00 ~ 0xFF |

示例：OpenTask(1);//启动ID为1任务

## 2.4指定优先级启动任务

指定优先级启动任务，需要System\_Task.h/.c文件中OpenTaskS函数

函数原型：OpenTaskS(u8 TaskID,u8 Py)

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| TaskID | 任务ID | 0x00 ~ 0xFF |
| Py | 任务的类型优先级 | 0x01 ~ 0xFF |

示例：OpenTaskS (1, Task\_Class\_L2);//以优先级为2的方式启动ID为1任务

## 2.5使用系统的延时功能

使用系统的延时功能，需要System\_Clock.h/.c文件中TaskDelayMs函数

函数原型：TaskDelayMs(u16 TimeMs)

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| TimeMs | 延时时长（单位：毫秒） | 0x00 ~ 0xFFFF |

示例：TaskDelayMs(20);//延时20毫秒

注意：些延时与软件延时不同，当使用延时后，当前的任务将会挂起，等待指定延时时长后，才会被启动

## 2.6使用系统的区域保护功能

### 2.6.1开启任务保护

开启任务保护，需要System\_ Task.h/.c文件中EnableProtection函数

函数原型：EnableProtection(u16 r1)

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| r1 | 延时时长（单位：毫秒） | 0x00 ~ 0xFFFF |

示例：EnableProtection (20);//当任务保护时间设为20毫秒

### 2.6.2取消任务保护

取消任务保护，需要System\_ Task.h/.c文件中DisableProtection函数

函数原型：DisableProtection()

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| void | 空 | 空 |

示例：DisableProtection ();//取消任务保护时间设为20毫秒

## 2.7条件任务触发接口

取消任务保护，需要System\_ Task.h/.c文件中TypeTaskStartupn函数

函数原型：TypeTaskStartupn (u8 TYPE)

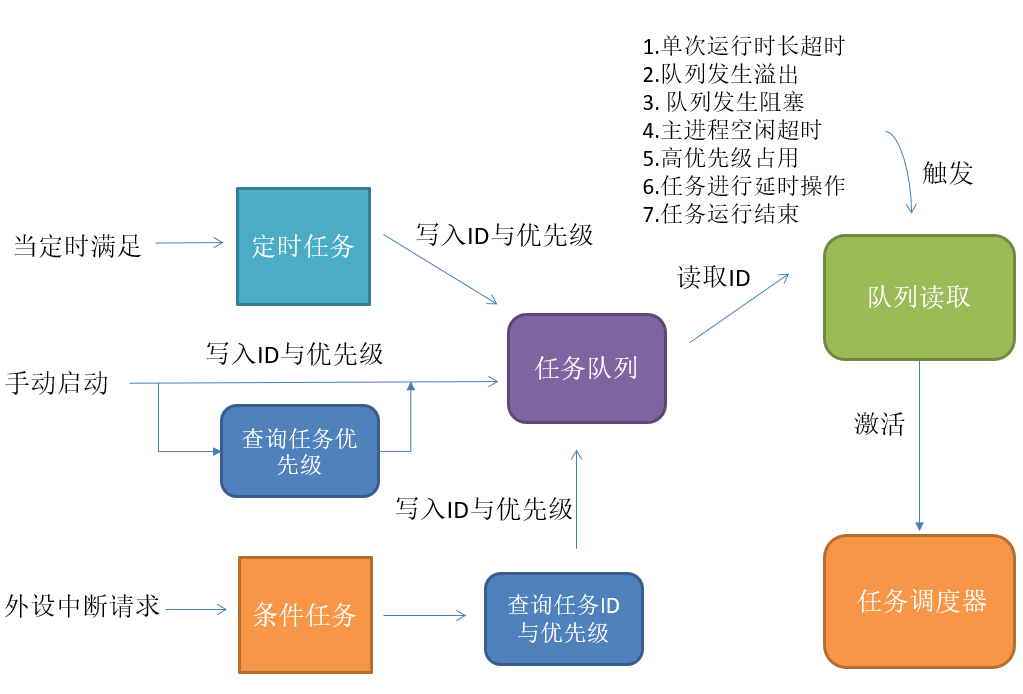
|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数范围 |
| TYPE | 任务类型 | 空 |

示例：TypeTaskStartupn (ADC);//启动与ADC相关的任务

# 3.深入

## 3.1了解基本的

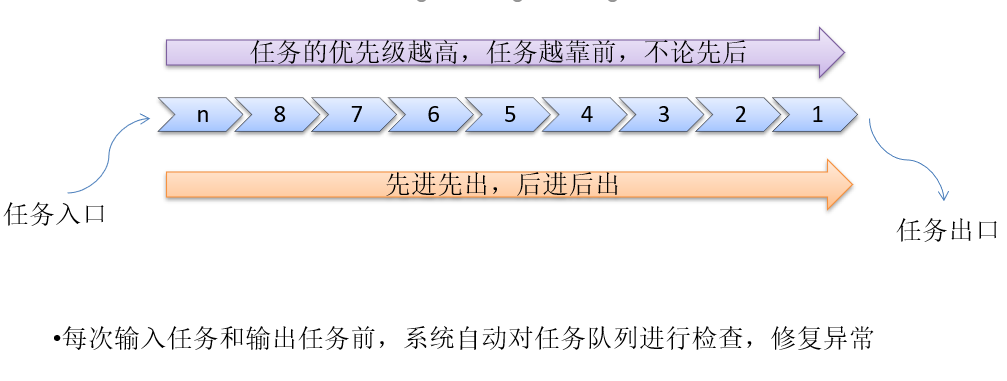
### 3.1.1系统的任务系统



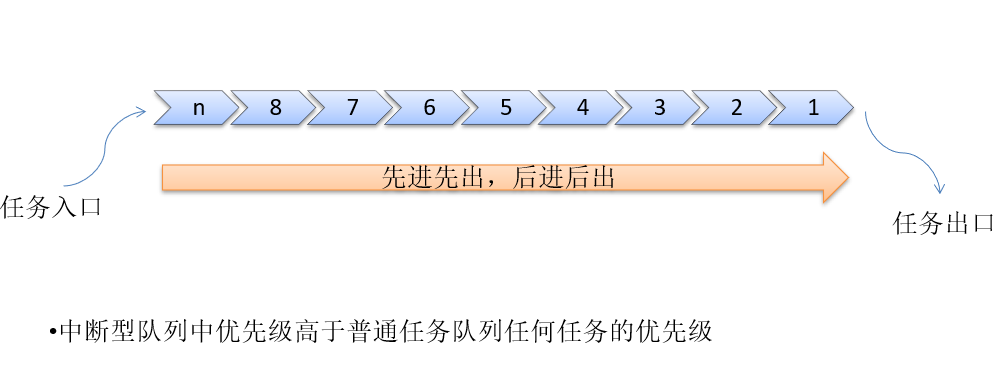
这张图，大概描述了任务系统，将任务运行的方式，分为了三大类，分为定时触发、手动触发和条件触发。

任务系统的重点，主要在队列方面，而任务队列分为普通任务队列和中断型任务队列

普通型任务队列：主要对任务的时序与优先级管理，具体管理方式，如图所示。

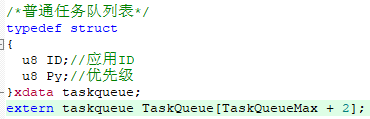


中断型任务队列：用于存放因为某些原因，被强制打断的任务，中断型任务队列的中任务永远比普通型任务队列的优先级要高。中断型任务队列内，没有优先级管理，只有时序管理。比普通型任务队列，运行起来要快的多。



这两个队列的具体的定义在“System\_task.h”文件中

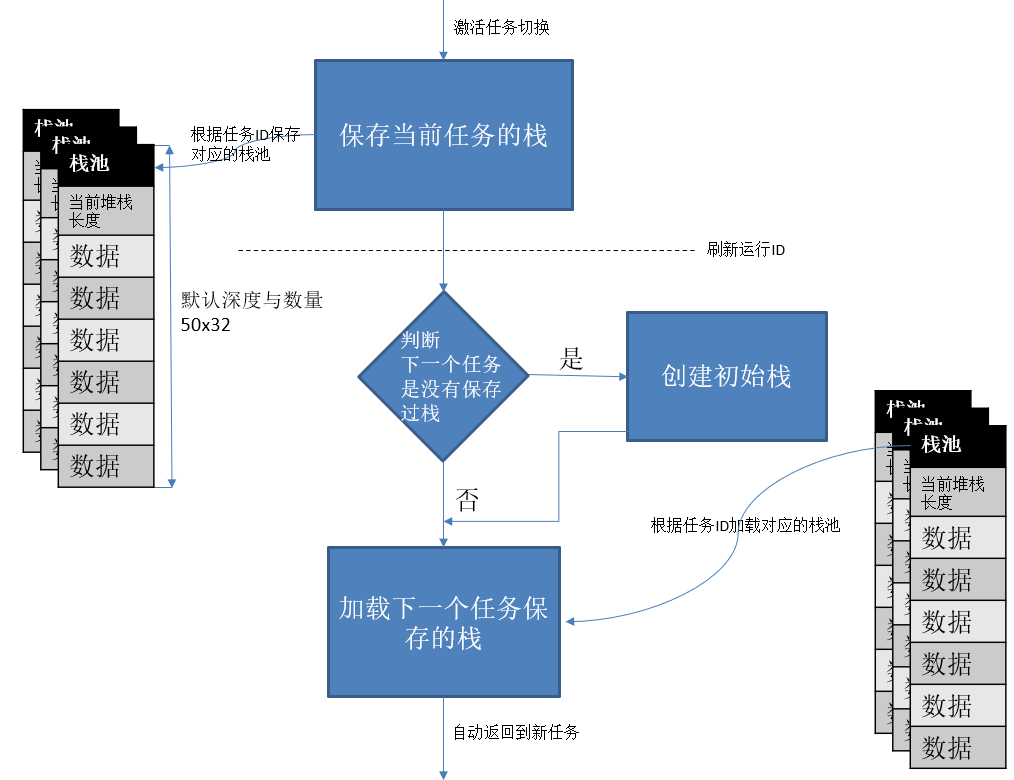




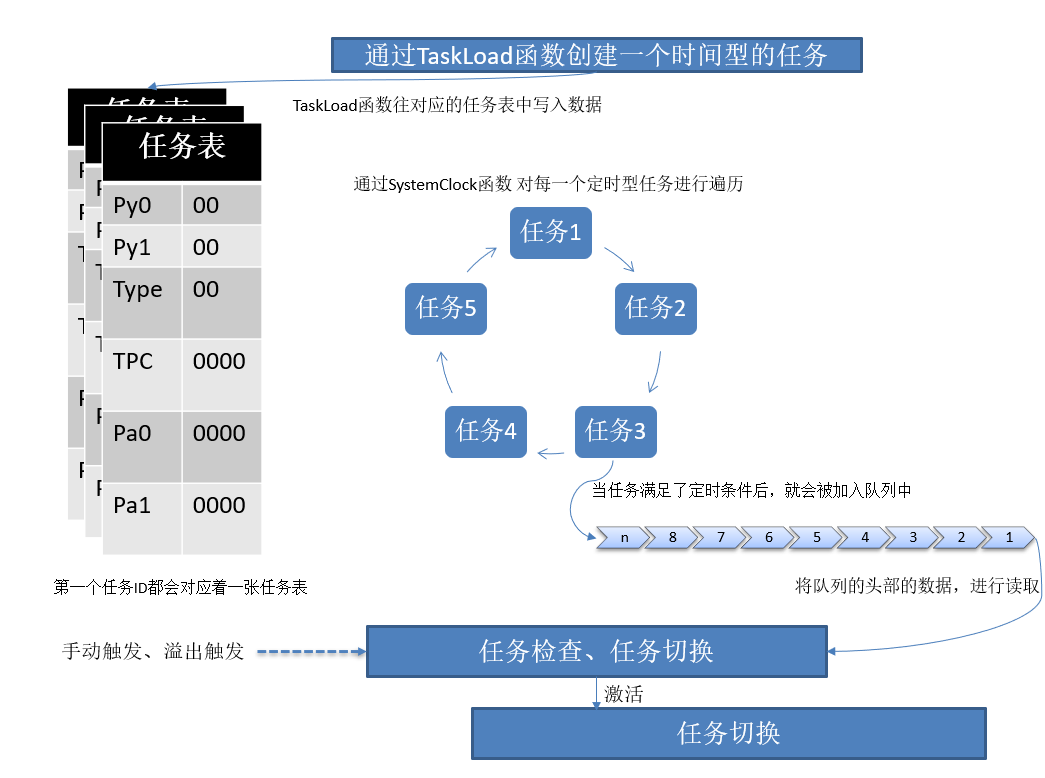
从定义中，我们可以看出中断型任务队列与普通型任务队列的长度是相同的，都由TaskQueueMax进行定义，那我们为什么对最大长度，还要加二呢？这么做，是为了保证队列的尾部有空隙，以免意外发生的溢出。

下面我们即将进入，系统的“大动脉“了，就是任务的上下文切换。

任务上下文切换，主要交给TaskSwitch函数来完成。其大概具体流程如上图所示。其函数存放在”CPU.c”文件中。

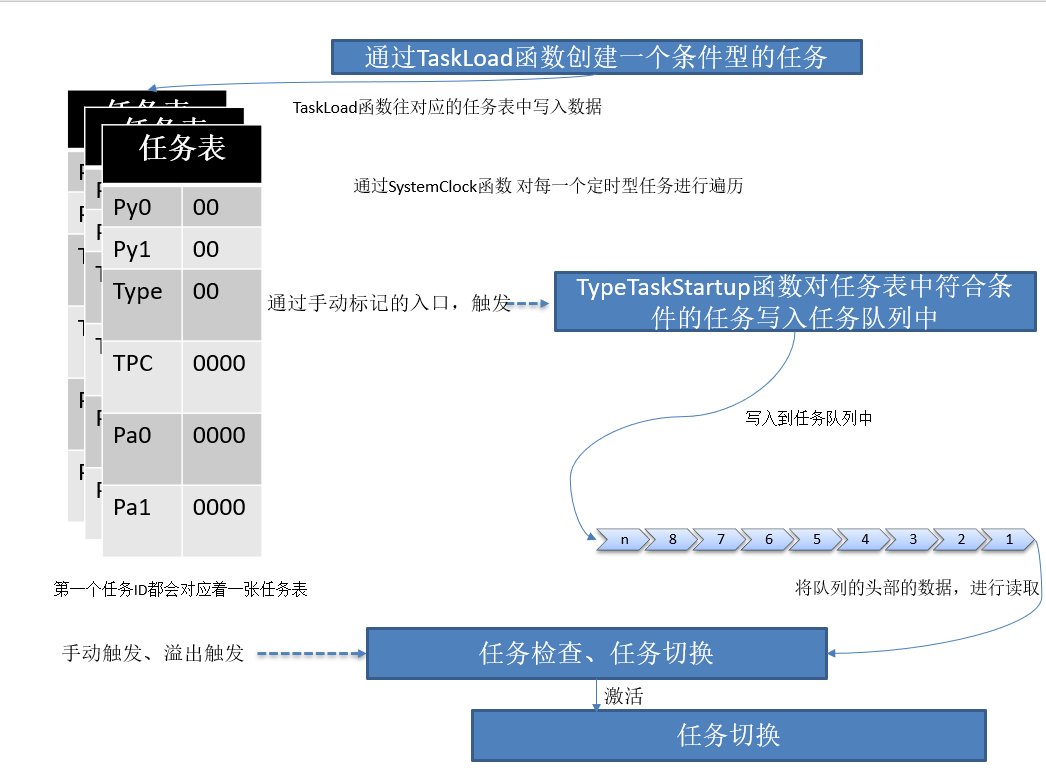


下面分别以时间型任务与条件型任务为例，讲述下，大概运行机制。



上图为时间型任务的运行大概的原理，其中的时钟系统为核心，具体内容请参考下一节的“系统的时钟系统”中内容。

下面，让我们看一下条件型条件的运行大概原理

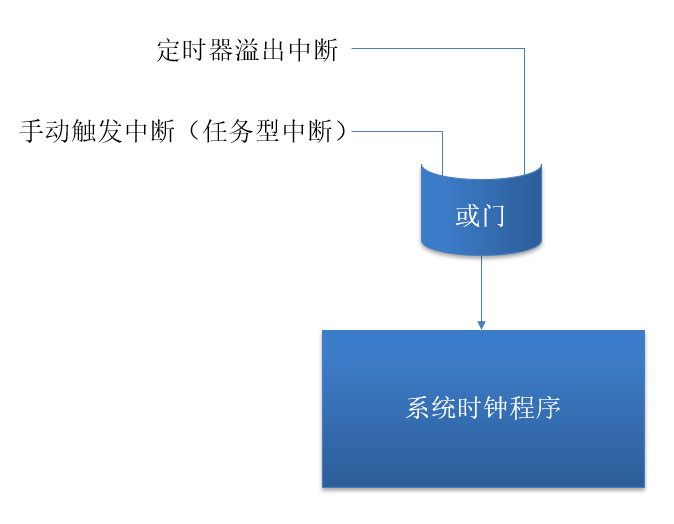


与时钟型任务的运行机制差不多，依赖于手动触发，在原有手动触发上加上任务对照检查。

注意：任务表默认开启弹性检查（弹性检查：根据占用最大数量，进行实时的计算，降低在进行遍历与查询操作的时间开销），如需要可在System\_Contlor.h文件中将对其的宏定义注释即可。

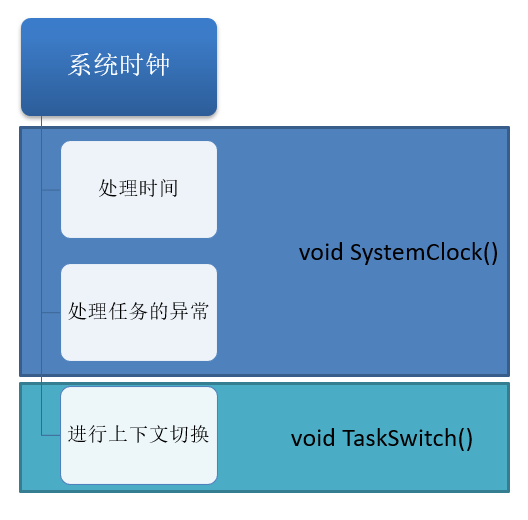
## 3.2系统的时钟系统

本系统的时钟，分为两种触发中断方式，通过” TimerIRQTouchType”参数用于判断类型，只有参数为零时，才可判断为定时器溢出触发的中断。

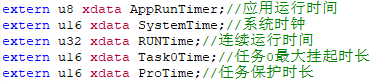


系统时钟程序，不仅仅处理时间，还要处理任务系统的异常，进行任务的上下文切换，可谓是整个系统的核心。

系统时钟的程序，由”SystemClock”和”TaskSwitch”两个函数构成，其作用如下图所示。



除此以外，还有几个系统时钟的参数，需要留意。



其中 SystemTime、RUNTime都可以直接使用。

## 3.3系统的移植与剪裁

在进行移植之前，我要了解系统基本情况。

硬件方面：必须提供一个系统时钟。

Timer\_x : 任意一个定时器，用于提供系统时钟（必选）。

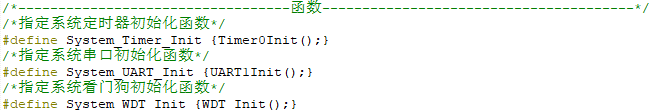
UART\_x : 任意一个串口，提供串口通信（可选）。

WDT : 看门狗一个，用于宕机复位（可选）。

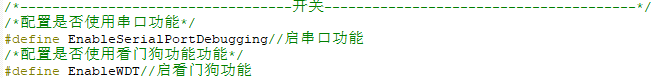
Power : 提供硬件系统电源管理，用于系统对MCU复位，进入烧录模式等（可选）。

所有系统相关的软硬件，都会在System\_Init函数中完成初始化操作。

在System\_Control.h文件中，可指定用于硬件初始化函数入口。

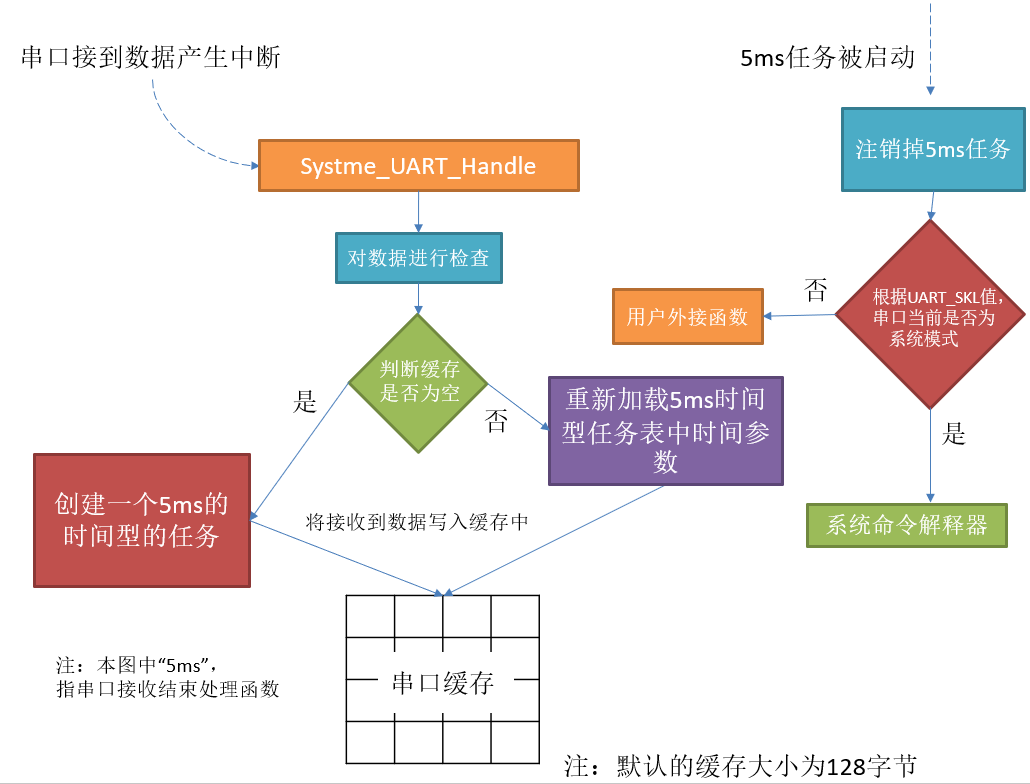


在System\_Control.h文件中，可配置启用的硬件功能。



一般在System\_Control.h文件中，配置完成后，基本不用修改System\_Init函数了。

## 3.4系统的串口系统



上图系统自带的串口系统基本工作原理，串口系统的主要代码存放于System\_Uart.c/.h文件中，如果不需要的话，可以在System\_Control.h文件中，可配置启用的硬件功能。

负责串口系统主要由Systme\_UART\_Handle与Systme\_UART\_Input两大函数负责。

串口缓存的大小与串口响应时间，可以在System\_Uart.h中进行配置。



## 3.5绍系统串口的基本使用

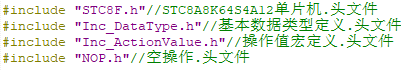
系统串口在初始化以115200bps进行的初始化，使用时需要将串口调试软件的波特率调至115200，才可正常使用。

下表是所有串口系统所有的命令以及其含义。

|  |  |
| --- | --- |
| 命令 | 命令含义 |
| OU | 使用串口输出字符串 |
| RST | 进位单片机复位 |
| ISP | 使单片机进入到ISP模式 |
| OFT | 关闭任务系统 |
| PHI | 使单片机进入高功耗模式 |
| OFW | 关闭看门狗 |
| OPT | 启动某个任务 |
| QTS | 查看任务表中所有的任务 |
| QST | 查看系统时间参数 |
| QRT | 查看系统运行时长 |
| PPD | 使单片机进入掉电模式 |
| INFO | 查看系统信息 |
| OPTU | 开启系统监视器 |
| OFTU | 关闭系统监视器 |
| PIDL | 使单片机进入掉电模式 |
| PLOW | 使单片机进入低功耗模式 |
| HELP | 列出所有串口命令 |

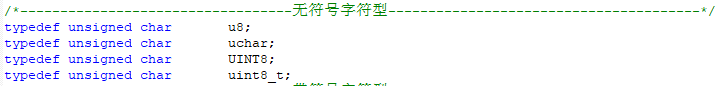
## 3.6其他文件的介绍

### 3.6.1 Inc\_Basic.h文件



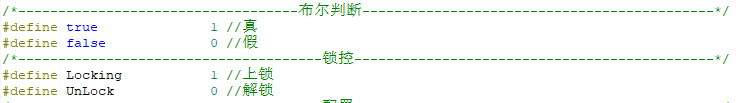
用于存放与加载基本，通用的头文件

### 3.6.2 Inc\_DataType.h文件



用于基本数据类型的定义

### 3.6.3 Inc\_ActionValue.h文件



用于操作值宏定义

### 3.6.4 NOP.h文件



用于空指令函数定义

### 3.6.5 NOP.A51 文件

存放NOP函数的语句体

### 3.6.6 STARTUP.A51 文件

存放8051单片机的启动文件

4.结束

在开发之前，你要知道以下几点

1.我们已经完全适配了STC8A8K64S4A12@22.1184MHz1

2.你的单片机至少要有11KByte的程序空间和至少50Byte的RAM以及2200Byte的SRAM，这样系统才能正常运行2

3.你的单片机最好支持1T或者更快3

4.请使用keil5来开发

5.关于系统的详细内容请参考《NerveRTOS开发指南》

注释：

1.目前只适配了STC8A8K64S4A12@22.1184MHz，不保证STC8A8K64S4A12的库正确性、稳定性、完整性

2.以上的硬件配置只能保证系统正常运行，不预留用户程序的空间

3.6T/12T单片机也可以运行系统，速度和稳定性可能较低

感谢您Nerve RTOS的支持，也希望Nerve RTOS对于您的学习或者开发有所帮助