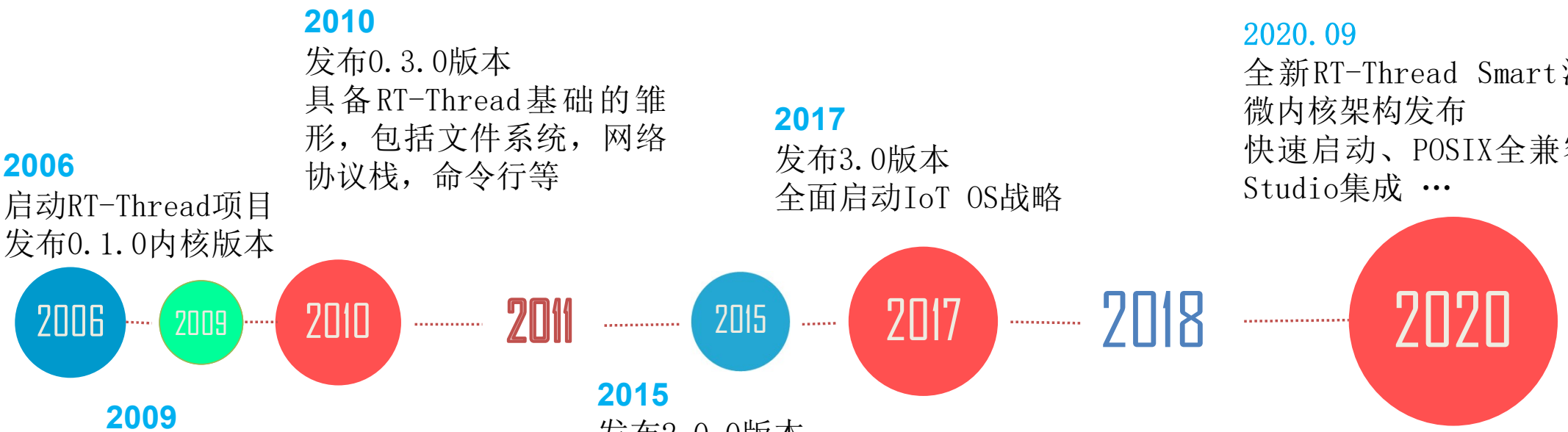
RT-Thread介绍

——By BalanceTWK

RT-Thread发展历程



RT-Thread

操作系统多线程编程思想

1.1 线程的概念

- 在日常生活中，当我们遇到一个大任务，一般会将它分解成多个简单、容易解决的小问题，小问题逐个被解决，大问题也就随之解决了。
- 比如：做一碗这样的盖饭，就可以分解为：蒸米饭和炒菜两个小任务。当我们分别完成这两个小任务，大的任务也就完成了。
- 线程——操作系统能够进行运算调度的最小单位。

RT-Thread

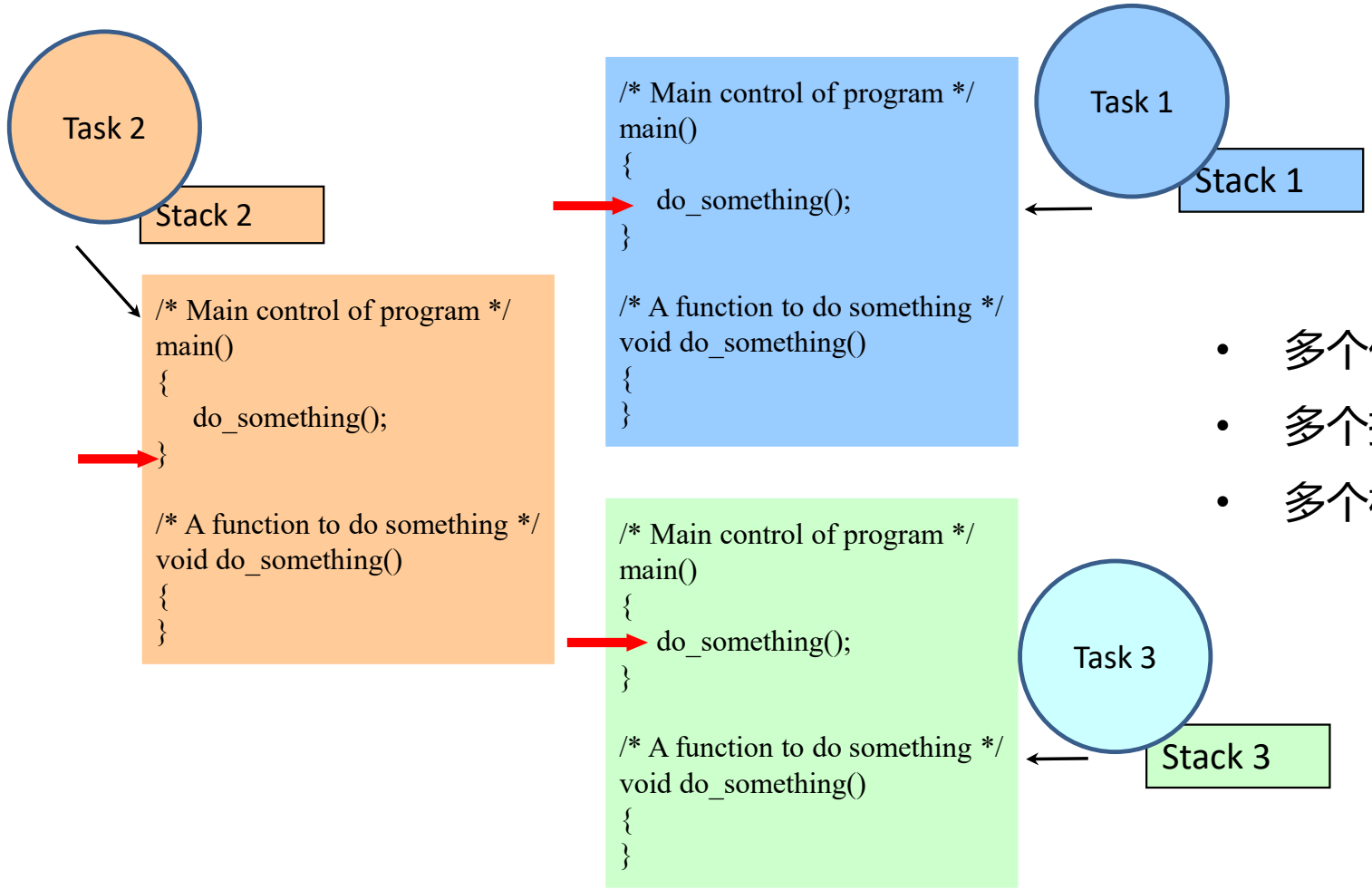
1.2 多线程运行对比

- 裸机

- RTOS

RT-Thread

1.3 多线程的并发调度执行



- 多个任务竞争、并发执行
- 多个执行线索
- 多个相互独立的堆栈空间

RT-Thread

1.4.1 调度器的工作

- 调度器最主要的工作就是：
  - 保证一个最高优先级的任务运行（如果是抢占式调度器，需立刻运行）
    - 从就绪线程列表中查找最高优先级任务。
  - 任务切换
    - 正常状态下：任务到任务的切换；
    - 抢占式调度还需要考虑在中断状态下，唤醒了高优先级任务，“即刻”切换到高优先级的任务运行。

RT-Thread

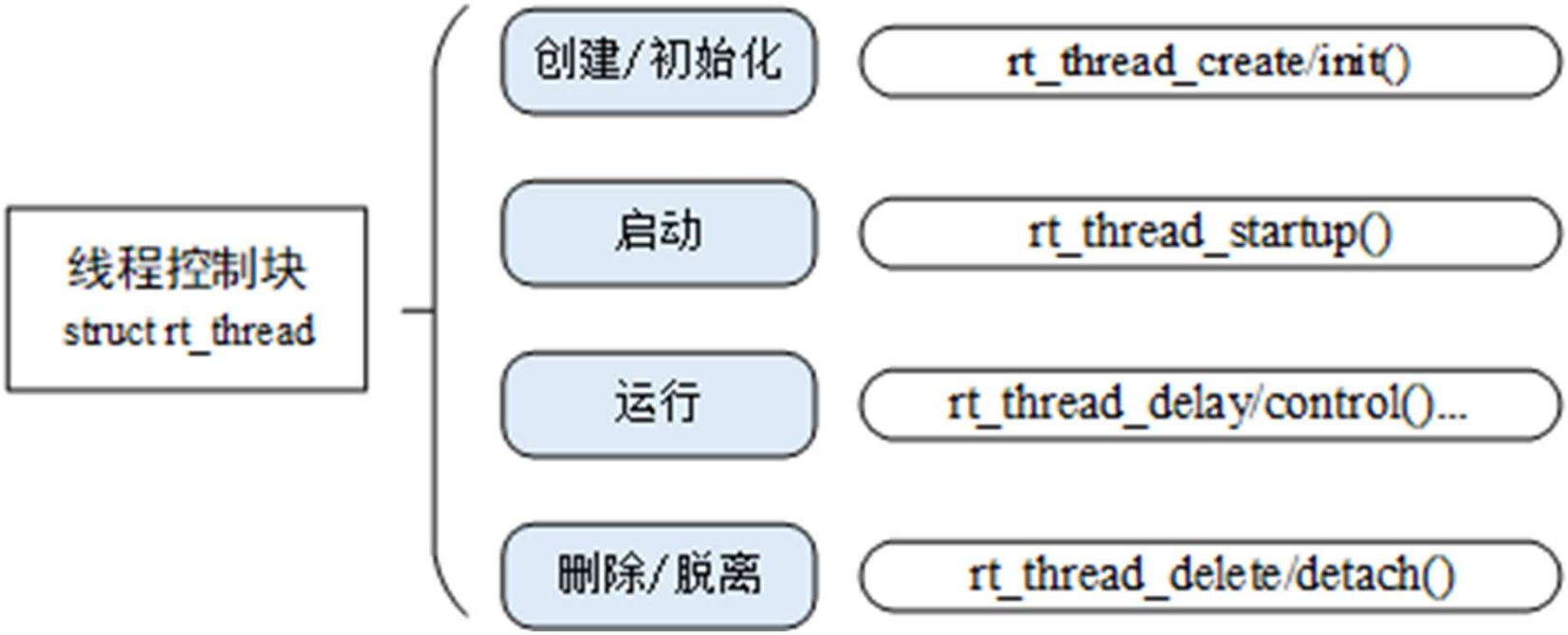
1.4.2 调度器的工作

- 任务切换是调度器的基本功能，但每种处理器架构是不相同的，也是移植时需要用汇编完成的部分移植。

- 当一段代码运行时，它会认为它独占处理机进行运行：
  - 处理机核心中的各种寄存器资源：PC，SP，R0，… LR等。
  - 当任务切换时，这些当前场景将被保存起来，当再切换到这个任务时，这些场景信息将被恢复。
  - 保存场景的地方就是线程运行的栈空间。

RT-Thread

1.5.1 自己写一个线程



- 线程相关API。

RT-Thread

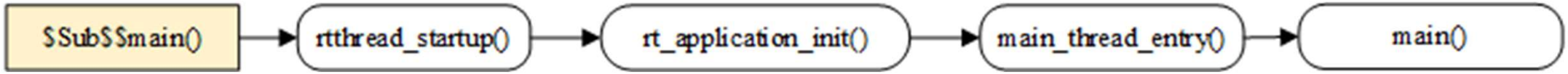
1.5.2 main函数也是一个线程？

```
msh >ps
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ---------- ------  ---------- ---
ledshine  16  suspend 0x00000074 0x00000100     45%  0x00000014 000
tshell    20  running 0x00000084 0x00001000     11%  0x00000002 000
tidle0    31  ready   0x00000044 0x00000100     32%  0x00000007 000
msh >
```

```
msh >ps
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ---------- ------  ---------- ---
tshell    20  running 0x00000084 0x00001000     11%  0x00000003 000
tidle0    31  ready   0x00000044 0x00000100     32%  0x00000008 000
main      10  suspend 0x00000074 0x00000800     11%  0x00000013 000
msh >
```

RT-Thread

1.5.3 main函数也是一个线程？




```
193 void rt_application_init(void)
194 {
195     rt_thread_t tid;
196
197 #ifdef RT_USING_HEAP
198     tid = rt_thread_create("main", main_thread_entry, RT_NULL,
199                            RT_MAIN_THREAD_STACK_SIZE, RT_MAIN_THREAD_PRIORITY, 20);
200     RT_ASSERT(tid != RT_NULL);
201 #else
202     rt_err_t result;
203
204     tid = &main_thread;
205     result = rt_thread_init(tid, "main", main_thread_entry, RT_NULL,
206                             main_stack, sizeof(main_stack), RT_MAIN_THREAD_PRIORITY, 20);
207     RT_ASSERT(result == RT_EOK);
208
209     /* if not define RT_USING_HEAP, using to eliminate the warning */
210     (void)result;
211 #endif
212
213     rt_thread_startup(tid);
214 }
```

1.5.4 多线程编程要点

- 在实时系统中，线程通常是被动式的：
  - 这个是由实时系统的特性所决定的；
  - 实时系统通常总是等待外界事件的发生，而后进行相应的服务：

```
void thread_entry(void* paramenter)
{
    while (1)
    {
        /* 等待事件的发生 */

        /* 对事件进行服务、进行处理 */
    }
}
```

RT-Thread

1.5.5 多线程编程要点

- 另一种情况：一次性服务线程：
  - 线程启动后将持续运行，直到它消亡：

```
static void thread_entry(void* paraemter)
{
    /* 处理事物#1 */
    …
    /* 处理事物#2 */
    …
    /* 处理事物#3 */
}
```

- 在RT-Thread中例如初始化线程，即采用如上方式。

RT-Thread

1.5.6 多线程编程要点（总结）

- 设计一个线程应该考虑的几个因素：
  - 名称信息
    - RT-Thread线程支持（对象）名称，为了调试方便，系统中应为每个线程安排不同的线程名称。
  - 线程栈大小
    - 对于资源相对较大的MCU，可以适当设计较大的线程栈。
    - 也可以初始时设置较大的栈，然后在finsh shell中用list_thread()查看最大的栈使用量，而后加上适当的余量形成最终的线程栈大小。
  - 线程时间片
    - 在RT-Thread中可对每个线程的时间大小进行设置，通常时间片大小应该大于3。
    - 时间片通常仅对同优先级就绪线程有效，可以根据情况进行适当配置。
    - 通常时间片对于系统的实时行为是无法确定性的。

RT-Thread

1.5.7 多线程编程要点（总结）

- 线程优先级
  - 在实时系统中，系统的好坏与线程优先级的安排有着密切的关系。
  - 优先级的设计请遵循以下原则：
    - 与实时性需求相关的线程优先级应设置得比较高。
    - 每次运行时间短（或运行时间/空闲时间值较低）的线程优先级可以进行提高；
    - 管理类线程优先级应提高；
    - 运行时间长的线程应该降低；

RT-Thread

2.1.0 线程间同步的概念

- 同步是指按预定的先后次序进行运行，线程同步是指多个线程通过特定的机制（如互斥量，事件对象，临界区）来控制线程之间的执行顺序，也可以说是在线程之间通过同步建立起执行顺序的关系，如果没有同步，那线程之间将是无序的。
- 信号量（semaphore）、互斥量（mutex）、和事件集（event）。

RT-Thread

2.1.1 信号量（semaphore）

信号量

以生活中的停车场为例来理解信号量的概念：

①当停车场空的时候，停车场的管理员发现有很多空车位，此时会让外面的车陆续进入停车场获得停车位；

②当停车场的车位满的时候，管理员发现已经没有空车位，将禁止外面的车进入停车场，车辆在外排队等候；

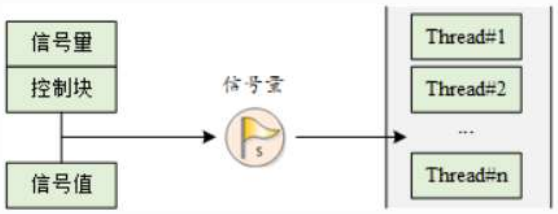
③当停车场内有车离开时，管理员发现有空的车位让出，允许外面的车进入停车场；待空车位填满后，又禁止外部车辆进入。

在此例子中，管理员就相当于信号量，管理员手中空车位的个数就是信号量的值（非负数，动态变化）；停车位相当于公共资源（临界区），车辆相当于线程。车辆通过获得管理员的允许取得停车位，就类似于线程通过获得信号量访问公共资源。

信号量工作机制

信号量是一种轻型的用于解决线程间同步问题的内核对象，线程可以获取或释放它，从而达到同步或互斥的目的。

信号量工作示意图如下图所示，每个信号量对象都有一个信号量值和一个线程等待队列，信号量的值对应了信号量对象的实例数目、资源数目，假如信号量值为 5，则表示共有 5 个信号量实例（资源）可以被使用，当信号量实例数目为零时，再申请该信号量的线程就会被挂起在该信号量的等待队列上，等待可用的信号量实例（资源）。



RT-Thread

2.1.2 互斥量（mutex）

互斥量

互斥量又叫相互排斥的信号量，是一种特殊的二值信号量。互斥量类似于只有一个车位的停车场：当有一辆车进入的时候，将停车场大门锁住，其他车辆在外面等候。当里面的车出来时，将停车场大门打开，下一辆车才可以进入。

互斥量工作机制

互斥量和信号量不同的是：拥有互斥量的线程拥有互斥量的所有权，互斥量支持递归访问且能防止线程优先级翻转；并且互斥量只能由持有线程释放，而信号量则可以由任何线程释放。

互斥量的状态只有两种，开锁或闭锁（两种状态值）。当有线程持有它时，互斥量处于闭锁状态，由这个线程获得它的所有权。相反，当这个线程释放它时，将对互斥量进行开锁，失去它的所有权。当一个线程持有互斥量时，其他线程将不能够对它进行开锁或持有它，持有该互斥量的线程也能够再次获得这个锁而不被挂起，如下图时所示。这个特性与一般的二值信号量有很大的不同：在信号量中，因为已经不存在实例，线程递归持有会发生主动挂起（最终形成死锁）。

RT-Thread

2.1.3 事件集（event）

事件集

事件集也是线程间同步的机制之一，一个事件集可以包含多个事件，利用事件集可以完成一对多，多对多的线程间同步。下面以坐公交为例说明事件，在公交站等公交时可能有以下几种情况：

①P1 坐公交去某地，只有一种公交可以到达目的地，等到此公交即可出发。

②P1 坐公交去某地，有 3 种公交都可以到达目的地，等到其中任意一辆即可出发。

③P1 约另一人 P2 一起去某地，则 P1 必须要等到 “同伴 P2 到达公交站” 与“公交到达公交站”两个条件都满足后，才能出发。

这里，可以将 P1 去某地视为线程，将 “公交到达公交站”、“同伴 P2 到达公交站” 视为事件的发生，情况①是特定事件唤醒线程；情况②是任意单个事件唤醒线程；情况③是多个事件同时发生才唤醒线程。

事件集工作机制

事件集主要用于线程间的同步，与信号量不同，它的特点是可以实现一对多，多对多的同步。即一个线程与多个事件的关系可设置为：其中任意一个事件唤醒线程，或几个事件都到达后才唤醒线程进行后续的处理；同样，事件也可以是多个线程同步多个事件。这种多个事件的集合可以用一个 32 位无符号整型变量来表示，变量的每一位代表一个事件，线程通过 “逻辑与” 或“逻辑或”将一个或多个事件关联起来，形成事件组合。事件的 “逻辑或” 也称为是独立型同步，指的是线程与任何事件之一发生同步；事件 “逻辑与” 也称为是关联型同步，指的是线程与若干事件都发生同步。

RT-Thread

2.2.0 线程间通信的概念

- 在裸机编程中，经常会使用全局变量进行功能间的通信，如某些功能可能由于一些操作而改变全局变量的值，另一个功能对此全局变量进行读取，根据读取到的全局变量值执行相应的动作，达到通信协作的目的。RT-Thread 中则提供了更多的机制帮助在不同的线程中间传递信息。
- 邮箱（mailbox）、消息队列（messagequeue）、信号（signal）用于线程间的通信。

RT-Thread

2.2.1 邮箱（mailbox）

邮箱

邮箱服务是实时操作系统中一种典型的线程间通信方法。举一个简单的例子，有两个线程，线程 1 检测按键状态并发送，线程 2 读取按键状态并根据按键的状态相应地改变 LED 的亮灭。这里就可以使用邮箱的方式进行通信，线程 1 将按键的状态作为邮件发送到邮箱，线程 2 在邮箱中读取邮件获得按键状态并对 LED 执行亮灭操作。

这里的线程 1 也可以扩展为多个线程。例如，共有三个线程，线程 1 检测并发送按键状态，线程 2 检测并发送 ADC 采样信息，线程 3 则根据接收的信息类型不同，执行不同的操作。

邮箱的工作机制

RT-Thread 操作系统的邮箱用于线程间通信，特点是开销比较低，效率较高。邮箱中的每一封邮件只能容纳固定的 4 字节内容（针对 32 位处理系统，指针的大小即为 4 个字节，所以一封邮件恰好能够容纳一个指针）。典型的邮箱也称作交换消息，如下图所示，线程或中断服务例程把一封 4 字节长度的邮件发送到邮箱中，而一个或多个线程可以从邮箱中接收这些邮件并进行处理。

RT-Thread

2.2.2 消息队列（messagequeue）

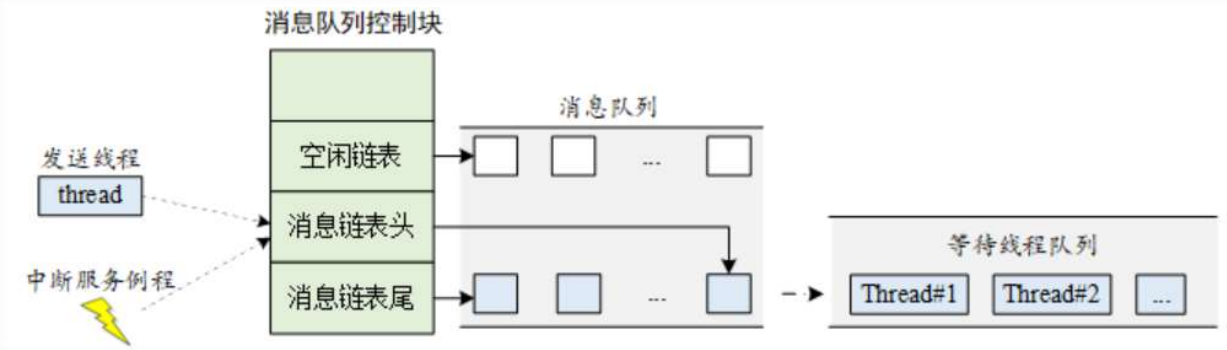
消息队列

消息队列是另一种常用的线程间通讯方式，是邮箱的扩展。可以应用在多种场合：线程间的消息交换、使用串口接收不定长数据等。

消息队列的工作机制

消息队列能够接收来自线程或中断服务例程中不固定长度的消息，并把消息缓存在自己的内存空间中。其他线程也能够从消息队列中读取相应的消息，而当消息队列是空的时候，可以挂起读取线程。当有新的消息到达时，挂起的线程将被唤醒以接收并处理消息。消息队列是一种异步的通信方式。

如下图所示，线程或中断服务例程可以将一条或多条消息放入消息队列中。同样，一个或多个线程也可以从消息队列中获得消息。当有多个消息发送到消息队列时，通常将先进入消息队列的消息先传给线程，也就是说，线程先得到的是最先进入消息队列的消息，即先进先出原则 (FIFO)。



RT-Thread

2.2.3 信号（signal）

信号

信号（又称为软中断信号），在软件层次上是对中断机制的一种模拟，在原理上，一个线程收到一个信号与处理器收到一个中断请求可以说是类似的。

信号的工作机制

信号在 RT-Thread 中用作异步通信，POSIX 标准定义了 sigset_t 类型来定义一个信号集，然而 sigset_t 类型在不同的系统可能有不同的定义方式，在 RT-Thread 中，将 sigset_t 定义成了 unsigned long 型，并命名为 rt_sigset_t，应用程序能够使用的信号为 SIGUSR1（10）和 SIGUSR2（12）。

信号本质是软中断，用来通知线程发生了异步事件，用做线程之间的异常通知、应急处理。一个线程不必通过任何操作来等待信号的到达，事实上，线程也不知道信号到底什么时候到达，线程之间可以互相通过调用 rt_thread_kill() 发送软中断信号。

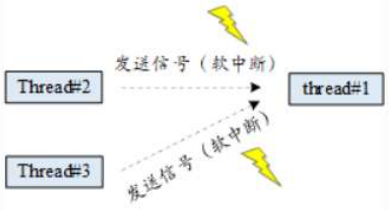
收到信号的线程对各种信号有不同的处理方法，处理方法可以分为三类：

第一种是类似中断的处理程序，对于需要处理的信号，线程可以指定处理函数，由该函数来处理。

第二种方法是，忽略某个信号，对该信号不做任何处理，就像未发生过一样。

第三种方法是，对该信号的处理保留系统的默认值。

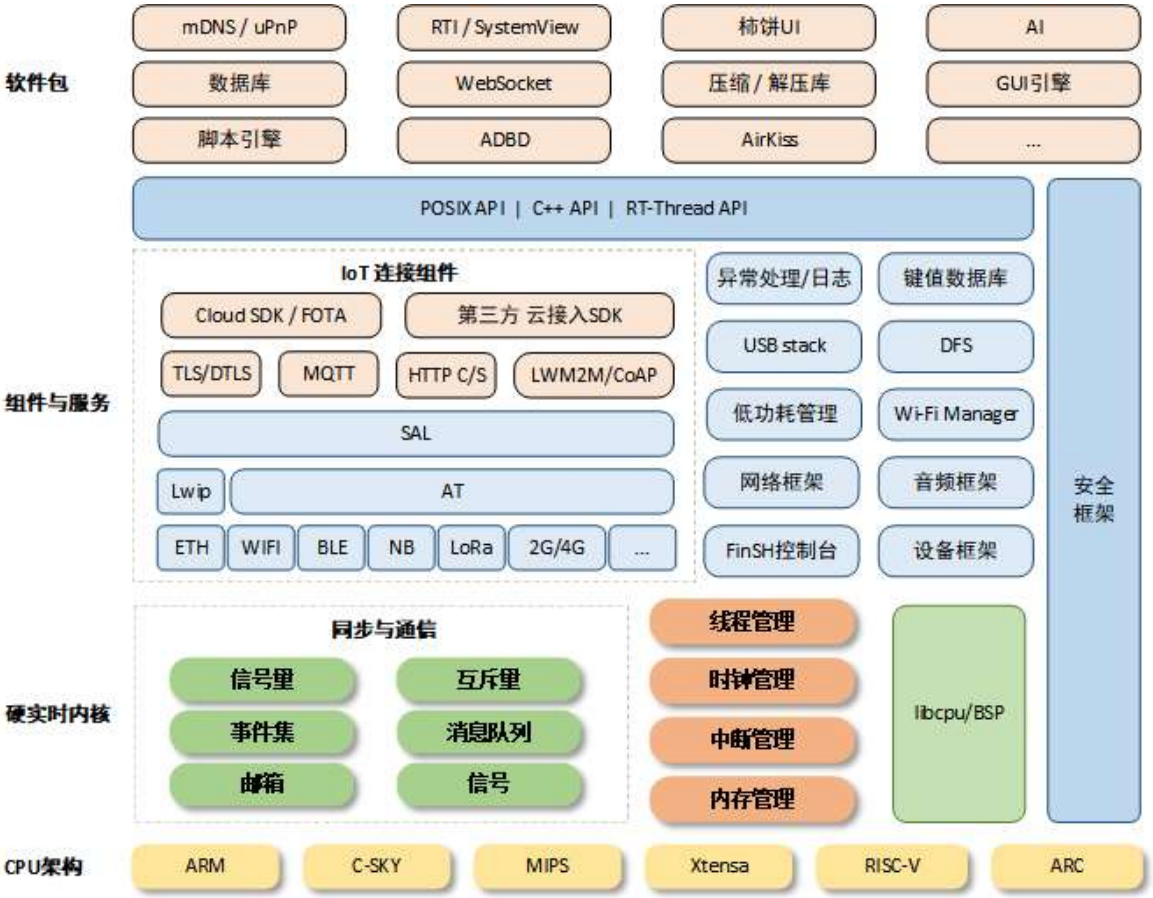
如下图所示，假设线程 1 需要对信号进行处理，首先线程 1 安装一个信号并解除阻塞，并在安装的同时设定了对信号的异常处理方式；然后其他线程可以给线程 1 发送信号，触发线程 1 对该信号的处理。



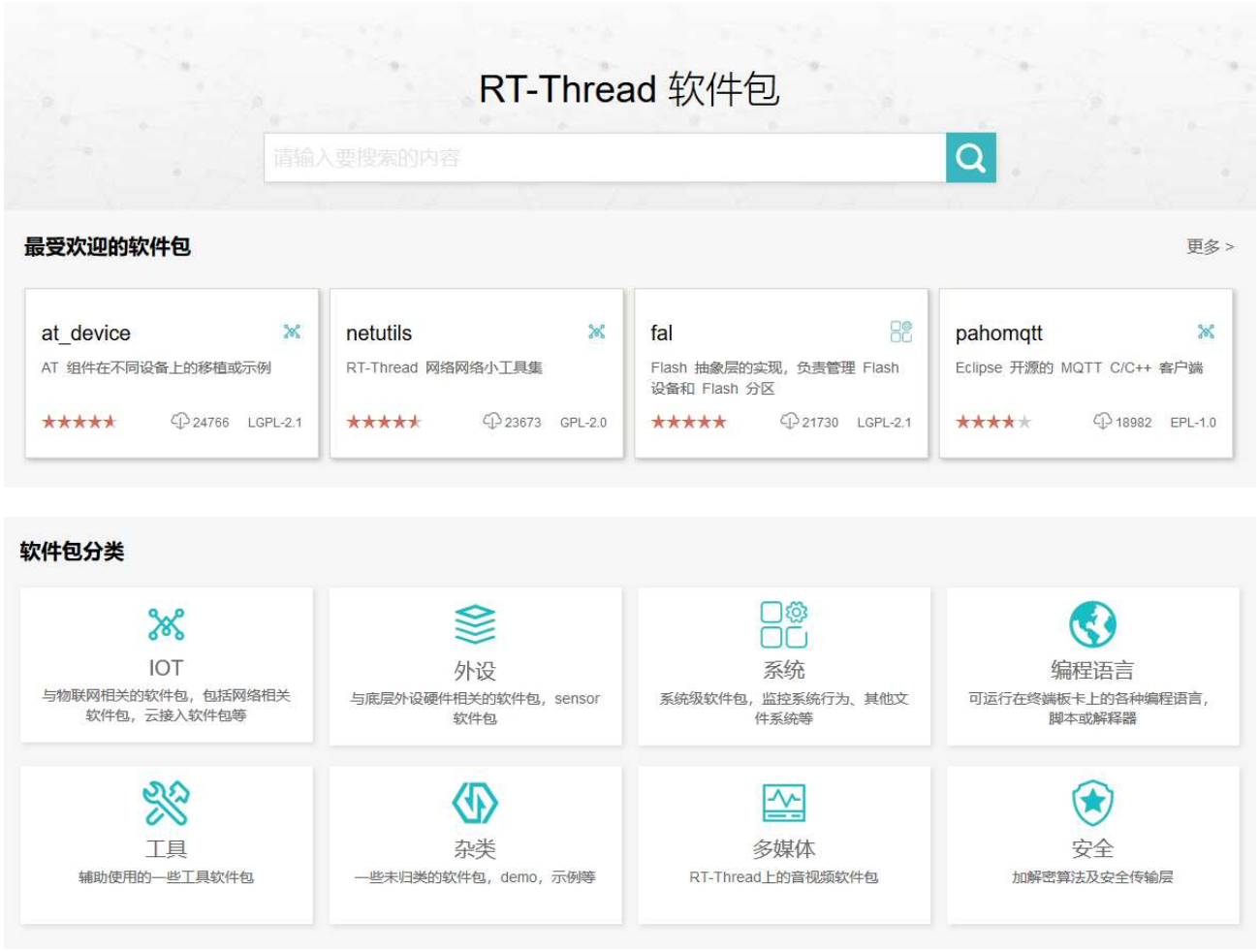
RT-Thread

RT-Thread 组件与软件包介绍

RT-Thread 的丰富功能



RT-Thread 的丰富功能



RT-Thread