如果说 goroutine 是 Go语言程序的并发体的话，那么 channels 就是它们之间的通信机制。

Go语言中的通道（channel）是一种特殊的类型。在任何时候，同时只能有一个 goroutine 访问通道进行发送和获取数据。goroutine 间通过通道就可以通信。  
通道像一个传送带或者队列，总是遵循先入先出（First In First Out）的规则，保证收发数据的顺序。

**声明通道类型**

通道本身需要一个类型进行修饰，就像切片类型需要标识元素类型。通道的元素类型就是在其内部传输的数据类型，声明如下：

var 通道变量 chan 通道类型

* 通道类型：通道内的数据类型。
* 通道变量：保存通道的变量。  
  chan 类型的空值是 nil，声明后需要配合 make 后才能使用。

**创建通道**

通道是引用类型，需要使用 make 进行创建，格式如下：

通道实例 := make(chan 数据类型)

* 数据类型：通道内传输的元素类型。
* 通道实例：通过make创建的通道句柄。

**Go语言中无缓冲的通道（unbuffered channel）是指在接收前没有能力保存任何值的通道。这种类型的通道要求发送 goroutine 和接收 goroutine 同时准备好，才能完成发送和接收操作。**

**Go语言中有缓冲的通道（buffered channel）是一种在被接收前能存储一个或者多个值的通道。这种类型的通道并不强制要求 goroutine 之间必须同时完成发送和接收。通道会阻塞发送和接收动作的条件也会不同。只有在通道中没有要接收的值时，接收动作才会阻塞。只有在通道没有可用缓冲区容纳被发送的值时，发送动作才会阻塞。**

通道实例 := make(chan 通道类型, 缓冲大小)

* 通道类型：和无缓冲通道用法一致，影响通道发送和接收的数据类型。
* 缓冲大小：决定通道最多可以保存的元素数量。
* 通道实例：被创建出的通道实例。

select {  
    case <-chan1:  
    // 如果chan1成功读到数据，则进行该case处理语句  
    case chan2 <- 1:  
    // 如果成功向chan2写入数据，则进行该case处理语句  
    default:  
    // 如果上面都没有成功，则进入default处理流程  
}













