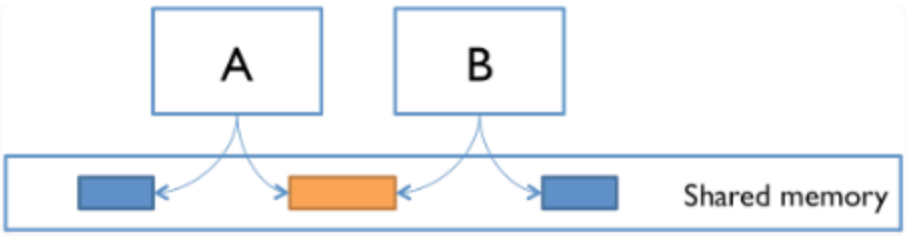
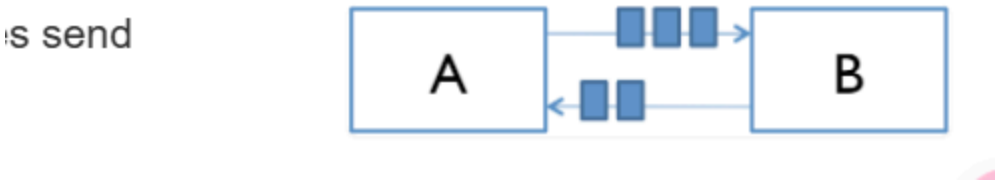
21.并发

两种并发模型





第一种是AB共享一个内存空间
第二种是AB是消息传递型

共享内存

在并发编程的共享内存模型中，并发模块通过读取和写入内存中的共享对象来进行交互。在右侧的图中 （文中未实际显示图片 ），A 和 B 是并发模块，它们可以访问相同的共享内存空间。蓝色对象是 A 或 B 私有的（只有一个模块可以访问），但橙色对象是 A 和 B 共享的（两个模块都有对它的引用）。

共享内存模型的示例：

- A 和 B 可能是同一台计算机中的两个处理器（或处理器核心），它们共享相同的物理内存。
- A 和 B 可能是在同一台计算机上运行的两个程序，它们共享一个可以读写文件的公共文件系统。

- A 和 B 可能是同一个程序中的两个线程（我们将在下面解释什么是线程），它们共享相同的对象。

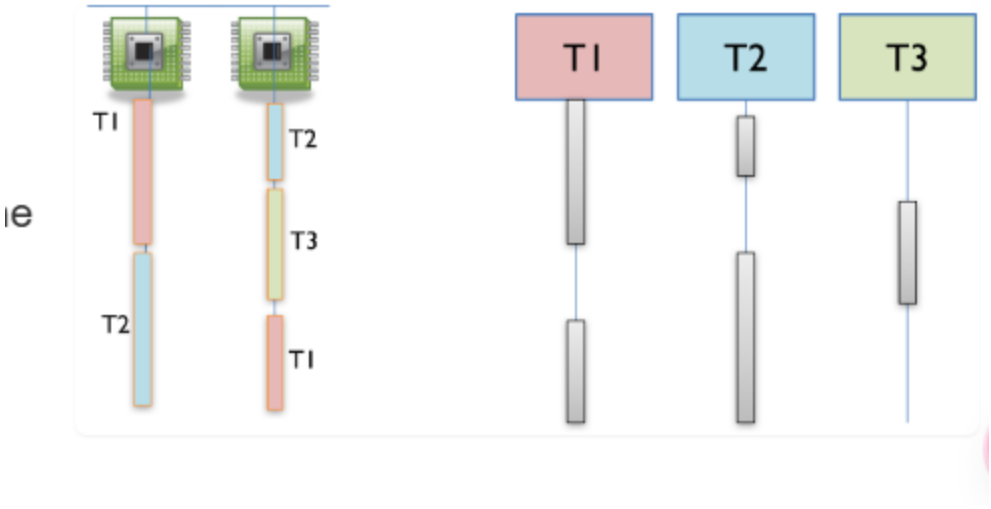
消息传递

在消息传递模型中，并发模块通过通信通道相互发送消息来进行交互。模块发送消息，进入每个模块的消息会被排队等待处理。示例包括：

- A 和 B 可能是网络中的两台计算机，通过网络连接进行通信。
- A 和 B 可能是一个网页浏览器和一个网页服务器 ——A 打开与 B 的连接并请求一个网页，B 将网页数据发送回 A。
- A 和 B 可能是一个即时通讯客户端和服务器。
- A 和 B 可能是在同一台计算机上运行的两个程序，它们的输入和输出通过管道连接，就像在命令提示符中输入 `ls | grep` 一样。

进程，线程，时间切片

进程：就像一台虚拟计算机，进程之间通常不会共享内存，一个进程无法访问另一个进程的内存和对象
线程：就像是一台虚拟计算机中的一个虚拟处理器。创建一个新线程就好比在由进程代表的虚拟计算机内部创建一个新的处理器。这个新的虚拟处理器运行与进程中其他线程相同的程序，并且共享相同的内存。
时间切片：假设一台计算机中只有两个处理器，那为什么能同时运行3个进程呢？因为可以将进程切片：



TypeScript 中的 Worker

TypeScript 没有可供用户直接访问的线程，但它有一个名为 `Worker` 的抽象概念。由于这个 API 最初出现在网页浏览器中，所以有时也被称为 `Web Workers`，并且 Node 也支持这个 API。

`Worker` 是通过指定它应执行的 JavaScript 文件作为起始点来创建的。例如，假设我们已经有一个通过编译以下 TypeScript 代码生成的 `hello-world.js` 文件：

收起

```typescript
// hello-world.ts（编译为hello-world.js）
console.log('Hello from a worker!')
```

然后我们可以像这样为它启动一个 `Worker`：

收起

```typescript
let worker: Worker = new Worker('./hello-world.js');
//...
```

这段代码会立即为 `Worker` 创建一个新的环境，开始执行 `hello-world.js`，并打印出问候消息。同时，启动 `Worker` 的代码也会继续并发执行。

Mozilla 和 Node 关于 `Worker` 的文档都将它们称为线程，但它们的行为更像是独立的进程：

- 与线程类似，`Worker` 可以使用 `SharedArrayBuffer` 来访问共享内存，以共享二进制数据。
- 但与进程类似，每个新的 `Worker` 都在一个全新的全局环境中运行其 JavaScript 文件。当它导入一个模块时，会加载该模块的一个全新私有副本，不会与其他 `Worker` 共享该模块。`Worker` 无法共享变量或常规对象的实例。
- 相反，由于它们的共享内存能力非常有限，`Worker` 通常通过消息传递进行通信。在创建 `Worker` 的代码与在该 `Worker` 内部运行的代码之间会自动设置一个双向消息传递通信通道。

总的来说，我们可以把 `Worker` 看作是一种轻量级的进程。

交叉执行（交错）

A	B
A readBalance() returns 200	
A add 1	
A writeBalance(201)	
	B readBalance() returns 201
	B add 1
	B writeBalance(202)

但是也可能是：

A	B
A readBalance() returns 200	
	B readBalance() returns 200
A add 1	
	B add 1
A writeBalance(201)	
	B writeBalance(201)

这就是“交错”