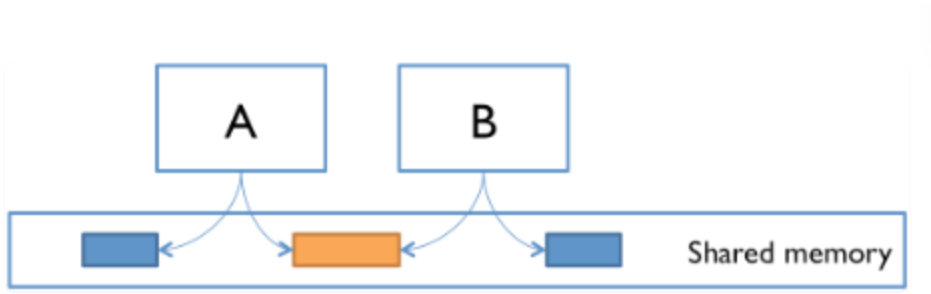
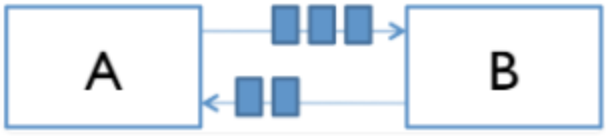
24.Message-Passing

两中并发模型

共享内存模型：



消息传递模型：



Worker

worker有两个消息端口，一个是 创建worker线程时的possMessage，另一个是工作线程的parentPort.postMessage

主线程代码（`main.ts`）

收起 ^

typescript

```typescript
import { Worker } from 'worker_threads';

// 创建一个工作线程
const worker = new Worker('./worker.ts');

// 主线程通过 worker 实例的消息端口向工作线程发送消息
worker.postMessage('Hello from main thread!');

// 主线程通过 worker 实例的消息端口监听来自工作线程的消息
worker.on('message', (message) => {
    console.log('Main thread received:', message);
});
```

工作线程代码（`worker.ts`）

收起 ^

typescript

```typescript
import { parentPort } from 'worker_threads';

// 工作线程通过 parentPort 消息端口监听来自主线程的消息
parentPort.on('message', (message) => {
    console.log('Worker thread received:', message);

    // 工作线程通过 parentPort 消息端口向主线程发送消息
    parentPort.postMessage('Hello from worker thread!');
});
```

停止线程

1. 定义停止请求的消息类型

首先，我们需要定义一种用于表示停止请求的消息类型。在之前关于 `DrinksFridge` 的示例中，使用了可辨识联合类型来定义请求消息，示例代码如下：

收起

typescript

```typescript
type FridgeRequest = DrinkRequest | StopRequest;
type DrinkRequest = { name: 'drink', drinksRequested: number };
type StopRequest = { name:'stop' };
```

这里定义了 `FridgeRequest` 类型，它可以是表示饮料请求的 `DrinkRequest` 类型，也可以是表示停止请求的 `StopRequest` 类型。

2. 发送停止请求消息

在主线程中，当我们想要停止 `DrinksFridge` 线程时，需要向该线程发送停止请求消息。假设我们已经创建了 `DrinksFridge` 的实例 `fridge` ，并且它与主线程之间通过消息端口进行通信，

示例代码如下：

收起

typescript

```
import { Worker } from 'worker_threads';
// 假设这里创建了与 DrinksFridge 相关的 Worker 实例 fridge
const fridge = new Worker('./drinksFridgeWorker.ts');
// 发送停止请求消息
fridge.postMessage({ name:'stop' });
```

上述代码通过 `postMessage` 方法向 `DrinksFridge` 线程发送了一个 `StopRequest` 类型的消息，以此告知线程需要停止运行。

3. 线程接收并处理停止请求

在 `DrinksFridge` 线程中，需要监听并处理接收到的消息。以下是 `DrinksFridge` 类中相关的处理逻辑示例代码：

```
class DrinksFridge {
    private readonly port: MessagePort;
    private messageCallback: ((req: FridgeRequest) => void) | null = null;

    constructor(port: MessagePort) {
        this.port = port;
    }

    public start() {
        this.messageCallback = (req: FridgeRequest) => {
            if (req.name ==='stop') {
                this.stop();
            } else if (req.name === 'drink') {
                const n = req.drinksRequested;
                const reply = this.handleDrinkRequest(n);
                this.port.postMessage(reply);
            }
        };
        this.port.addListener('message', this.messageCallback);
    }
```

竞态条件和死锁

其上主要讨论的是消息传递型的并发
同样存在竞态条件和死锁

比如这种死锁：

python

```python
QUEUE_SIZE = 100
N = 100

requests = Queue(maxsize=QUEUE_SIZE)
replies = Queue(maxsize=QUEUE_SIZE)

fridge = DrinksFridge(requests, replies)
fridge.start()

# 往冰箱里放入足够的饮料以开始
requests.put(-N)
print(replies.get())

# 发送请求
for x in range(N):
    requests.put(1)
    print(f'person #{x} is looking for a drink')
# 收集回复
for x in range(N):
    print(f'person #{x}: {replies.get()}')

print('done')
os._exit(0)  # 突然结束程序，通常这不是个好主意
```