NodeJS：回调地狱的解决方案

Promise, Async, Await

异步与回调

- 似乎异步与回调密不可分，不过事实并非如此。
- 只是我们多数见到的形式是这样设计的。
- 并且仍然有底层异步模式，代码层同步编写的设计，比如 Go 语言。

现实问题

- 现实的情况是， Node 异步回调的地狱已经形成。
- 事实上，在其他一些语言中，支持异步回调的模式也存在这样的问题。
- 指望重新设计解释器是不可能的。于是不得不寻找其他的解决方案。

以人类的认知去理解

- 如果一个异步的任务分为几个步骤，每个步骤先后顺序不能确定，但是却存在先后依赖。
- 作为一个观察者，你可以看到每个步骤的过程，并知道最终执行结果。
- 于是你每过一会儿看看任务是否完成并把结果传递到下一个 ...

Promise

- Promise 出现了，它承诺一个异步的调用可以有确定的先后关系。
- 它派出了两个使者： resolve 和 reject 。
- resolve 一旦执行表示 Promise 状态为 fulfilled， reject 则表示 Promise 状态为 rejected。

Promise

- 这个时候，可以使用 then 方法接收 resolve 或 reject 携带的值。
- 并且， Promise 说，你在 then 的回调函数中返回的值，我能看到，并且会传递到 then 调用链的下一层。

Promise 的问题

- Promise 能解决很多回调层级过多的问题，但是它不能完全消除回调。
- 另一方面，链式调用代码比较冗余，并且难以进行任务分解。

- 多数文章都在重复的讨论 Generator， yield 关键字，以及如何从生成器转换到 async 和 await 。
- 但是似乎这样讲反而适得其反，并且其底层具体的实现方式可能只有解释器的设计者知道。
- 于是这里我们尝试一些其他轻量级方案。

async function

- async function 定义一个返回 AsyncFunction 对象的函数。
- 调用 async function 声明的函数会返回 Promise 对象，并可以使用 .then 方法接收函数的返回值。

await

- await 只能用于 asyn function 声明的函数中。
- await 后面可以是 async function 声明的函数，可以使 Promise，如果 await 后面不是 Promise，则返回该值本身。

await

- await 会暂停当前 async function 的执行，等待 Promise 处理完成。若 Promise 正常处理 (fulfilled)，其回调的 resolve 函数参数作为 await 表达式的值。
- 若 Promise 异常 (rejected)， await 会把 Promise 的异常抛出。

async、 await 简单示例

```
function fa() {
    return 1;
}

function fb() {
    return 2;
}

async function add() {
    //如果await后面跟的不是Promise，则直接返回值
    var x = await fa();
    var y = await fb();
    return x+y;
}

add().then(r => {
    console.log(r);
});
```

async、 await 示例

```
function delay(seconds, str) {
    return new Promise((rv, rj) => {
        setTimeout(function() {
            rv(str);
        }, seconds * 1000);
    });
}

async function ax() {
    var x = await delay(2, 'success');
    var y = await delay(3, 'ok');
    console.log(x, y);
}

ax();
```

延时 5 秒后
才会输出结果

其他示例

- ppt 无法展示长代码，请参考配套源代码文件。