以前这样封装，会造成回调地狱问题，难道只能这样套下去？？

```
$.ajax({
      type: "method",
      url: "url",
      data: "data",
      dataType: "dataType",
      success: function (response) {
        console.log(response);
        $.ajax({
          type: "method",
          url: "url",
          data: response.authorId,
          dataType: "dataType",
          success: function (res) {
            $.ajax({
              type: "method",
              url: "url",
              data: res.address,
              dataType: "dataType",
              success: function (response) {

              }
            });
          }
        });
      }
    });
```

promise为了解决回调地狱

poromise 实例中需要传递函数，函数中又两个参数（resolve，reject)

resolve用来处理成功函数 ， reject用来处理失败的原因 成功或者失败只能存在一种

promise 实例中保存的是状态

fulfilled（成功状态）

rejected(失败状态)

pending （初始状态）

Promise.all() 传入的是一个数组，会等所有异步调用完成一起执行会把结果放入一个数组里面

```
Promise.all([p1, p2]).then(res => {
    //   console.log(res);
    // })
```

Promise.race() 不会等说有结果，谁先出结果展示谁

```
 Promise.race([p1, p2]).then(res => {
      console.log(res);
    })
```

async

被async关键字修饰得函数，会挂载在promise上

💣💣💣await 等待 等待一个async（承诺）这个promise不知道什么时候来 是异步的，必须和async连用 不能单使用

被aeait修饰的函数会像同步一样执行

```
async function glote(){
           let res= await getauthor(2)
           console.log(res);

        }
         glote()

```

用try 和 cath 捕获错误

```
async function glote() {
            try {
                // console.log(adhaldh);
```

```
            let res = await getauthor(2)
            let sd = await datale()
            console.log(res, sd);

        }
        catch (error) {
            console.log(error);

        }

    }
    glote()
```