**异步处理部分**

异步处理能够提高JavaScript单线程执行效率，使得JS线程不会在返回时白白空闲

具体实现是

JS线程遇到异步函数 -> 向对应线程发起任务 -> 对应线程执行异步任务，主线处继续其他同步任务 -> 对应线程完成任务后将返回放入异步队列 ->主线程空闲时检查异步队列，依次执行对应回调函数

使用异步函数的过程中常常会发生，同步部分中希望使用某个值异步更新后的新值，而此时异步函数还没有实际回调的问题，可以使用打包回调函数来解决，但又会产生层层回调嵌套时代码的可读性问题

使用promise和async await 来处理异步函数即可以在解决值是否符合预期的基础上，不会产生代码可读性的问题

Promise(function(resolve, reject){}

Promise的定义一般要包含在其他函数中，或是通过var=new Promise的方式定义，防止意外执行function中的内容。

可以使用Promise.then()来解决异步处理中的问题，then()会在promise的resolve完成,也就是异步任务处理完之后再执行，可以保证拿到的值是预计的

但需要注意的是promise.then().then()这样的连环then形式中只有第一个then能保证数据是更新后的，其余的then会在一个函数执行后就执行, 此时异步任务还未处理，值不是预计的，可以通过前一个then返回一个新的promise来解决，不过代码写起来依旧很长

Promise还可以用于处理执行中设好条件触发的另一分支，执行reject中的回调，也有catch()方法对应resolve中的then(),此外promise整体中若出现异常，此时即时没有满足reject条件，也会进入到catch()中

async function(){}

async会将一个同步函数改变为异步函数，也可以认为是允许了将异步代码写成同步样式，解决了其他方法的可读性和代码量的问题

async默认返回的是一个promise对象，使得它可以直接使用连锁.then()的形式来保证值的预期

await只能使用在async中，await之后大多会放一个promise对象或返回promise 对象的表达式，~~await会使得promise异步任务完成后再之后的代码执行，相当于将这个promise中的内容加入了同步部分中（简书中的一篇文章误导了我好久）~~，await会使得promise在等待返回值时让出该进程，执行async之后的同步部分内容，等待到同步内容全部执行完毕后，回到async中之前等待返回的部分得到返回值，保证在async中，之后使用到的值都是预期的，使得async函数与同步函数的写法一致，但本质上还是异步的

async异步外，若除了同步内容还有异步函数，则第一次await时，跳出async执行同步内容，执行异步函数将其回调函数进入异步队列，但还轮不到执行，第二次await时因同步内容已经执行完了，故会执行异步队列，执行完毕后返回async继续执行，故上一段中强调只能在async的后续中保证值的预计

**cookie和localstorage部分**

二者都可以储存数据便于服务端进行各种判断

存在生命周期与浏览器一致，储存于内存的临时cookie和存在时间由设定决定，储存于硬盘的长效cookie,标准形式是cookie{key,value,存在时间}

Cookie容量小(4kb)，只支持字符串类型,由服务端发放，并在客户端的每一次请求中伴随着上传至客户端，用户没有改变cookie的权限，常用与储存基础简单的用户信息

由于用户每一次发送请求都会使得cookie们被伴随着上传一次，所以过多使用cookie会导致流量，效率方面的问题

localStorage存在周期为永久的，容量大（5mb），保存于客户端，不会将其中数据回传会服务端，仅用于进行本地判断，但服务端会通过识别码来判断localStorage

sessionstorage的容量和和交互方式同localstorage，作用类似于cookie，差别在于cookie保存在浏览器端，session保存在服务器端，在客户端时的生命周期仅为当前标签页。更大的容量使session承担其了体量较大的用户信息的储存工作，如购物车等