21.4.10学习记录

一、回顾

--扩展运算符

--Array.from()：

　　--把类数组对象转成数组

--Array.of():把一组值转为数组

字符串、arguments、对象等都可以转成数组（具备length就靠谱）

--arr.find()：找出第一个符合条件的数组成员，未找到返回undefined

--arr.findIndex():找第一个符合条件的索引，未找到返回-1

--arr.fill(填充的东西，开始位置，结束位置)

--在ES2016新增的arr.includes()，包含就返回true

```
let arrLi = [...aLi];
let arrLi = [].slice.call(aLi);
let arrLi = Array.from(aLi);
```

二、对象（JSON）

--对象简介语法（非常重要）

　　--外面定义了变量，json中不用再次赋值（name, === name:name），写函数可以省略function，但是千

万不要用箭头函数

--Object.is():比较两个值是否相等

```
console.log(NaN == NaN);  //false
console.log(Object.is(NaN, NaN)); //true
console.log(+0 == -0);  //true
console.log(Object.is(+0, -0)); //false
```

--Object.assign(目标对象，source1，):

--复制一个对象；合并参数

--Object.keys()

--Object.values()

--Object.entries() 键值对

--在对象上加扩展运算符

三、Promise

--解决异步回调的问题

--传统方式，大部分用回调函数，事件驱动等

```
new Promise(function (resolve, reject) {
    //resolve 成功时调用
    //reject 失败时调用

})
```

```
// promise.then(success,fail);
promise.then(res => {
    console.log(res);
}, err => {
    console.log(err);
})
```

```
// promise.then(success,fail);
promise.then(res => {
    console.log(res);
}).catch(err => { //发生错误别名
    console.log(err);
});
```

Promise.resolve()：将现有的东西转成一个promise对象，而且是resolve状态即成功状态

Promise.reject()：将现有的东西转成一个promise对象，而且是reject状态即失败状态

Promise.all([p1,p2,p3]):把promise打包，扔到数组中，打包完还是一个promise对象。但是必须确保所有格promise对象都是成功状态才可以

Promise.race([p1,p2,p3]):同上，但是只要有一个成功状态就行

四、模块化

--js一开始不支持模块化，所以在ES6之前，社区指定一套模块规范：

--Commonjs 主要服务端 nodejs require（‘http’）

--AMD require，curljs

--CMD seajs

--ES6出来统一服务端和客户端模块规范

需要放到服务器环境

①如何定义模块

export 东西

②如何使用

```
<script type="module"></script>
```

import 东西

```
export default 12;
```

使用default，导入时不用加大括号

③特点：

--路径既可以相对又可以绝对

--只会被导入一次

--import './js/mod.js':相当于引入文件

--模块可以依赖

--import会自动提升到顶部

--模块内若有定时器改变，外面也会随之改变

④import（）

--返回promise对象

--可以动态引入，默认import语法不能写到if之类里面

--按需加载，动态路径，也可以写在if中

其他：ES2017加入async、await

```
async function main() {
    const mod1 = await import('./js/mod.js');
    const mod2 = await import('./js/mod2.js');

    const [m1, m2] = await Promise.all([
        import('./js/mod.js'),
        import('./js/mod2.js')
    ]);
    console.log(m1, m2);
}
```

‘use strict’　以后可能默认就是严格模式

五、程序中类

类有属性和方法

--之前时函数模拟function

```
function Person(name, age) {
    //构造函数
    this.name = name;
    this.age = age;
}
Person.prototype.showName = function () {
    return `名字是：${this.name}`;
}
// Object.assign(Person.prototype, {
//     showName() {
//         return `名字是${this.name}`;
//     }
// })
```

现在是class

```
class Person {
    constructor(name, age) {
        //构造函数调用new即自动执行
        this.name = name;
        this.age = age;
    }
    showName() {
        return `名字为:${this.name}`;
    }
    showAge() {
        return `年龄为${this.age}`;
    }
}
```

const Person = class{}

这里允许类中的方法为变量，用【】号，而且JSON中也允许

类中没有提升功能，但是函数模拟有提升功能

类中this简单多了

矫正this：

--fn. call(this指向谁，args1，args2，....)

--fn. apply(this指向谁，[args1，args2，....])

--fn. bind()

```
constructor() {
    this.name = 'lrj';
    this.showName = this.showName.bind(this);
}
```

--取值函数getter　　--存值函数setter

静态方法（类身上方法）

```
static aaa() {
    return `这是静态方法`;
}
```

继承：之前:

```
function Student(name, skill) {
    Person.call(this, name);
    this.skill = skill;
}
Student.prototype = new Person();
```

现在:

```
class Student extends Person {

}
```

六、数据类型

number、string、boolean

Symbol不能new

返回是一个唯一值，是一个单独数据类型，就叫symbol，即基本类型；但是for循环遍历不显示

七、generator函数

--生成器，解决异步和深度嵌套问题

```
function* show() {
    yield 'welcome';
    yield 'to';
    yield 'lrj';
}
```

调用show().next();

可以用for..of调用，而且注意return的东西它并不会遍历

可以解构赋值

```
let [a, ...b] = show();
```

可以扩展运算符

```
console.log(...show())
```

还可以数组

```
console.log(Array.from(show()));
```

还可以结合axios请求

关于异步解决方案：

--回调函数

--事件监听

--发布、订阅

--Promise对象

八、async

读取文件：

--promise

--generator

--async function fn(){ //表示异步，这个函数里有异步任务

let res = await xxx; //表示后面结果需要等待

}

async特点：

--await只能放到async函数中

--相比generator语义性更强

--await后面可以是一个promise对象，也可是基本数据类型

--async函数返回是一个promise对象

--只要await语句后面Promise状态变成reject，整个async函数会终止执行，所以为了解决其抛出错误：

```
try {
    await Promise.reject('出错了');
} catch (e) {

}
```

还可以用promise本身catch

```
await Promise.reject('出错了').catch(err => {
    console.log(err);
});
```

如果遇到请求的文件互不相干，可以用promise.all()合并一起，再解构赋值