链滴

ES6 class 类的用法

作者：zhujie

原文链接：https://ld246.com/article/1588858039157

来源网站：链滴

es6 class基础用法
以前的JavaScript没有类的概念，它是基于原型的面相对象的语言。原型对象的特点就是将自身属性享给新对象。我们先看一下下面的代码实现。
```javascript
 //常规写法
function Person(name,age) {
    this.name = name;
    this.age = age;
}
Person.prototype.sayInfo = function () {
    console.log(`${this.name}是${this.age}岁`)
}
const liLei = new Person('LiLei',20)
liLei.sayInfo()
 //LiLei是20岁
```

这种常规约定是以大写字母开头来表示一个构造器(大写开头非官方)，可以直接定义函数，也可以通过rototype来扩展函数。这种实现跟java比，实现类的方案太特别了，下面我们看一下es6的类的实现式：

```
class Person{ //定义了一个名字为Person的类
    constructor(name,age){ //constructor是一个构造方法，用来接收参数
    this.name = name;  //this代表的是实例对象
    this.age = age;
}
sayInfo(){
    console.log(`${this.name}是${this.age}岁`)
    }
}
const liLei = new Person('LiLei',21)
liLei.sayInfo()
```

由下面代码可以看出类实质上就是一个函数。类自身指向的就是构造函数。所以可以认为ES6中的类实就是构造函数的另外一种写法！下面的代码可以证明这一点

```
console.log(typeof Person);//function
console.log(Person===Person.prototype.constructor);//true
```

类的继承

JavaScript中的类同样可以像java一样，可以继承某个类，其中被继承的类称为父类，而继承父类的称为子类。子类可以有自己的函数和构造器，当子类中存在父类相同的方法时，则该方法不会从父类承，而使用子类的方法。

```
class Student {
    constructor(name){
        this.name = name
    }
    sayName(){
        console.log(this.name)
    }
    testFn(){
        console.log('我是父类的函数！')
```

```
    }
    }
    class Worker extends Student{
    sayWork(){
        console.log(this.name)
    }
    testFn(){
        console.log('我是子类的函数！')
    }
}
const person = new Worker('liLei')
person.sayName()
person.sayWork()
person.testFn()
    //输出:
    //liLei
    //liLei
    //我是子类的函数!
```

可以看到子类Worker 继承了Student类的sayName函数和name这个内部变量。但是同名函数testF没有继承，是调用到了子类的testFn函数。这里也可以理解为子类的testFn函数覆盖了父类的testFn数。

super关键字的使用

super关键字的一个作用是用来访问父类的构造器或者函数用的。子类在使用构造器的时候，必须使用uper关键字，用来扩展构造器。上面提到的，子类同名函数会覆盖父类同名函数，这时候，我们使用sper关键字，同样能调用到父类的同名函数，就是简单理解为super其实是父类的一个实例对象。

```
class Student {
    constructor(name){
        this.name = name
    }
    testFn(){
        console.log('我是父类的函数！')
    }
}
class Worker extends Student{
    constructor(name,age,sex){
        super(name)    //这里必须先调用super，才有下文的this对象，这里扩展了一个变量age
        this.age = age
        this.sex = sex
    }
    testFn(){
        super.testFn();
        console.log("年龄" + this.age)
        console.log("性别" + this.sex)
        console.log('我是子类的函数！')
    }
}
const person = new Worker('liLei','20')
person.testFn()
//输出:
//我是父类的函数!
```

```
//年龄20
//性别undefined
//我是子类的函数!
//我是子类的函数!
```

可以看到上面用super关键字实现了子类的构造器，还扩展了2个变量age,sex。同时使用super调用了父类的方法，所以在子类中即使有父类的同名方法，一样可以实现父类同名方法的调用。super可理解为父类的一个会实例化对象，但不同的是super只能访问父类的方法和，不能访问私有变量。

static关键字

static关键字一般作用于类的方法，用来定义一个工具函数。static方法不能被实例对象调用，只能通类名来调用。同时static方法也可以被继承，而且也能在子类中用super对象来调用父类中的static方。

```
class Person{      //没有constructor的类会默认生成一个constructor构造器
        static sayName(){
            console.log("我是static函数")
        }
    }
class Student extends Person{}
const student = new Student()
Person.sayName()
Student.sayName()
student.sayName()
//输出:
//我是static函数
//我是static函数
//student.sayName is not a function
```

可以看到用实例化的对象来调用static方法时，代码会报错。

es6中类的使用场景

平时我们开发的时候很少使用类的，特别是现在基于vue或者react开发时，一些组件化的东西直接使各自框架封装好的方式引用，就使用的更少了。

但是某些时候，我们使用es6的类可以让我们的代码的可读性更高。比如说一个分页组件，里面会有算总页数，上一页，下一页，跳页等方法。我们可以把这个分页函数写在一个类里面，在引用的地方实例化它，当一个页面有多个分页时，也可以实例化多个，独立调用，互不影响。

总结来说，类可以在封装工具的时候用。最后附上分页工具类的大致代码：

```
class PageUtil{
    constructor(pageNo,pageSize,total){    //构造初始变量
        this.pageNo = pageNo;     //起始页面
        this.pageSize = pageSize  //一页数据条数
        this.total = total        //数据总数
        this.currentPage = 0      //当前选中页数
        this.pageTotal = Math.ceil(this.total/this.pageSize)   //总页数
    }
    nextPage(){     //下一页
        if(this.currentPage < this.pageTotal){
            this.currentPage++
```

```
        }
    }
    beforePage(){    //上一页
        if(this.currentPage > 1){
            this.currentPage--
        }
    }
    jumpPage(page){     //跳页
        this.currentPage = page
    }
    changePageSize(pageSize){    //改变页大小
        this.pageSize = pageSize
        this.pageTotal = Math.ceil(this.total/this.pageSize)   //总页数
    }
    getTotalPage(){    //获取总页数
        return Math.ceil(this.total/this.pageSize)
    }
}


class DialogPage extends PageUtil{    //继承PageUtil类
    constructor(pageNo,pageSize,total,pageTotal){
        super(pageNo,pageSize,total)
        this.pageTotal = pageTotal
    }
    getTotalPage(){
        return this.pageTotal || super.getTotalPage()   //重写getTotalPage方法
    }
}
const contentPage = new PageUtil(1,10,100)   //实例化2个pageUtil对象
contentPage.getTotalPage()
const dialogPage = new DialogPage(1,10,100,10)
dialogPage.getTotalPage()
```

全文完，如果有不对的地方，希望能积极指出来，大家都是学习者。