ES6（2015）

Reflect

Reflect => 反射，什么是反射机制?

Java的反射机制是在编译阶段不知道是哪个类被加载，而是在运行的时候才加载、执行

用法：

当不确定哪个方法执行这个方法时

```
console.log(Reflect.apply(Math.floor,null,[3.75]));  // 第一个参数：调用这个方法的对象/函数，后面为这个函数的其他参数
```

es5:

```
let price = 101.5;
if(price>100){
    price = Math.floor.apply(null,[price])
}else{
    price = Math.ceil.apply(null,[price])
}
console.log(price);
```

es6:

```
Reflect.apply(price>100?Math.floor:Math.ceil,null,[3.75]) // 动态调用方法

let d = Reflect.construct(Date,[]); // 动态实例化一个类
Reflect.has(obj,'x') // 判断对象上是否有这个属性
Reflect.get(obj,'name') // 读取对象数据
Reflect.get(array,0) // 读取数组数据
Reflect.set(obj,'z',3)  // 写数据
Reflect.set(arr,2,dfd) // 写数据

Object.freeze(obj) // 冻结对象
```

```
Reflect.preventExtensions(obj) // 冻结对象

Reflect.isExtensible(obj) // 判断是否是可扩展的（是否冻结）
Reflect.ownKeys(obj) // 返回它的key
Reflect.ownKeys([1,2])// 返回它的key



let student = {}
Object.defineProperty(student,'name',{value:'key'}); // 动态生成对象的属性，返回这个对象
Reflect.defineProperty(student,'name',{value:'key'}); // 区别在于返回值不同，true
Reflect.deleteProperty(student,'name',{value:'key'}); // 区别在于返回值不同，true
let res = Reflect.getOwnPropertyDescriptor(obj,"x") //  判断对象的某个键值的属性
let res1 = Object.getOwnPropertyDescriptor(obj,"x")
let res2 = Reflect.getPrototypeOf(obj,"x") // 获取对象的原型
let res3 = Object.getPrototypeOf(obj,"x")
let res2 = Reflect.setPrototypeOf(obj,string.prototype) // 修改对象的原型
let res3 = Object.setPrototypeOf(obj,string.prototype)
```