Proxy

在 ES6 标准中新增的一个非常强大的功能是 Proxy，它可以自定义一些常用行为如查找、赋值、枚举、函数调用等。通过 Proxy 这个名称也可以来它包含了“代理”的含义，只要有“代理”的诉求都可以考虑使用 Proxy 来实现。

Basic Syntax

语法

```
let p = new Proxy(target, handler)
```

解释

参数	含义	必选
target	用 Proxy 包装的目标对象（可以是任何类型的对象，包括原生数组，函数，甚至另一个代理）	Y
handler	一个对象，其属性是当执行一个操作时定义代理的行为的函数	Y

MDN 给出的解释偏官方，通俗的讲第一个参数 target 就是用来代理的“对象”，被代理之后它是不能直接被访问的，而 handler 就是实现代理的过

场景

我们经常读取一个对象的 key-value：

```
let o = {
  name: 'xiaoming',
  age: 20
}

console.log(o.name) // xiaoming
console.log(o.age) // 20
console.log(o.from) // undefined
```

当我们读取 from 的时候返回的是 undefined，因为 o 这个对象中没有这个 key-value。想想看我们在读取数据的时候，这个数据经常是聚合的，家没有按照规范来的时候或者数据缺失的情况下，经常会出现这种“乌龙”现象。

如果我们不想在调用 key 的时候返回 undefined，之前的做法是这样的：

```
console.log(o.from || '')
```

如果我们对所有代码都是这种写法，那阅读性和观赏性就不得而知了。值得庆幸的是，ES6 的 Proxy 可以让我们轻松的解决这一问题：

```
let o = {
  name: 'xiaoming',
  age: 20
}

let handler = {
  get(obj, key) {
    return Reflect.has(obj, key) ? obj[key] : ''
  }
}

let p = new Proxy(o, handler)

console.log(p.from)
```

这个代码是想表达如果 o 对象有这个 key-value 则直接返回，如果没有一律返回 ''，当然这里是自定义，大家可以根据自己的需要来写适合自己务的规则。