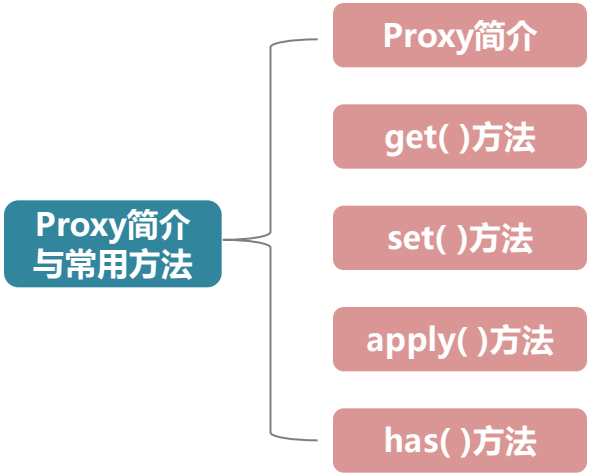
博学谷
www.boxuegu.com

第九章 Proxy和Reflect

一样的在线教育，不一样的教学品质

目录 Contents

Proxy



1.Proxy简介

Proxy相当于在**对象前添加了一个“拦截”层**；外界在对该对象进行访问时，必须先通过这个拦截层。

因此，Proxy提供了一种机制，可以**对外界的访问进行过滤和改写**。

Proxy的意思是“代理”，也就是说由Proxy来“代理”某些操作。所以又称之为**“代理器”**。

Proxy的使用：

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

Target：表示所要拦截的目标对象(原来要访问的对象)；

Handler：一个对象，表示拦截的行为和规则。

2.get( )方法

要想使用Proxy来完成对象的拦截，除了创建对象以外，还需要指定对应的拦截的方法。

get( )方法用于拦截某个属性的读取操作：

```
let student = {
            userName: '张三'
        }
        let proxy = new Proxy(student, {
            get: function(target, property) {
                if (property in target) {
                    return target[property];
                } else {
                    // （引用错误） 对象代表当一个不存在的变量被引用时发生的错误。
                    throw new ReferenceError('访问的属性' + property + "不存在")
                }
            }
        })
        console.log(proxy.userName);
        console.log(proxy.userAge);
```

get( ) 有两个参数：第一个参数表示目标对象，第二个参数表示要访问的属性。

注意：要使Proxy起作用，必须针对Proxy对象进行操作，不是针对目标对象进行操作(上面的是student对象)。

3.set( )方法

set( ) 方法用于拦截某个属性的赋值操作。

示例：若Student对象有一个age属性，表示学生的年龄；要求对年龄进行限制，如果大于60岁，给出错误提示，即可以使用Proxy对象保证age属性的取值是符合要求的。

```
let student = {
            name: 'zs',
            age: 20
        };
        let proxy = new Proxy(student, {
            set: function(obj, prop, value) {
                console.log('obj=', obj);
                console.log('prop=', prop);
                console.log('value=', value);
                if (prop === 'age') {
                    if (!Number.isInteger(value)) {
                        throw new TypeError('年龄不是整数! ')
                    }
                    if (value > 60) {
                        throw new RangeError('年龄太大了')
                    }
                }
            }
        })
        proxy.age = '80';
        console.log(proxy.age);
```

set( )方法：

第1个参数：表示拦截的对象；

第2个参数：表示操作的属性名称；

第3个参数：表示属性的值。

4.apply( )方法

apply( ) 方法拦截函数的调用，call和apply的操作；

apply( ) 函数的语法：

```
let handler={
    apply(target,ctx,args){
    }
}
```

4.apply( )方法

apply( )函数可以有3个参数：

第1个参数：表示目标对象，也就是要拦截的函数；

第2个参数：表示目标对象的上下文对象(this)；

第3个参数：表示目标对象的参数数组。

```
let target = function(msg) {
            return '你好' + msg;
        }
        let handler = {
            apply: function(target, ctx, args) {
                console.log('target=', target);
                console.log('ctx=', ctx === obj);
                console.log('args=', args);
                return 'hello'
            }
        }
        let proxy = new Proxy(target, handler);
        let obj = {
            proxy,
        };
```

5.has( )方法

has( ) 可以隐藏某些属性，不被in操作符发现。

```
let user = {
            _name: 'zhangsan',
            age: 20

        }
        let handler = {
            has(target, key) {
                console.log('target=', target);
                console.log('key=', key);
            }
        }
        let proxy = new Proxy(user, handler);
        '_name' in proxy; // 自动调用 has 方法
```

说明：has方法有两个参数：

第一个参数：表示**目的对象**；

第二个参数：表示**操作的属性**。

6.Proxy所支持的拦截操作

- **`get(target, propKey, receiver)`：**拦截对象属性的读取，比如proxy.foo和proxy['foo']。
- **`set(target, propKey, value, receiver)`：**拦截对象属性的设置，比如proxy.foo = v或proxy[‘foo’] = v，返回一个布尔值。
- **`has(target, propKey)`：**拦截propKey in proxy的操作，返回一个布尔值。
- **`deleteProperty(target, propKey)`：**拦截delete proxy[propKey]的操作，返回一个布尔值。
- **`ownKeys(target)`：**拦截Object.getOwnPropertyNames(proxy)、Object.getOwnPropertySymbols(proxy)、Object.keys(proxy)、for...in循环，返回一个数组。该方法返回目标对象所有自身的属性的属性名，而Object.keys()的返回结果仅包括目标对象自身的可遍历属性。
- **`getOwnPropertyDescriptor(target, propKey)`：**拦截Object.getOwnPropertyDescriptor(proxy, propKey)，返回属性的描述对象。

6.Proxy所支持的拦截操作

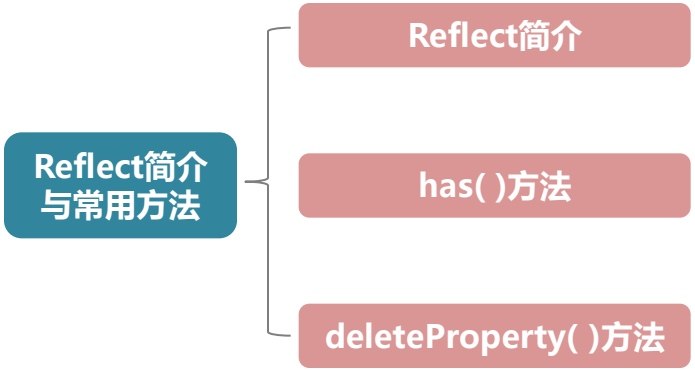
- **`defineProperty(target, propKey, propDesc)`：**拦截Object.defineProperty(proxy, propKey, propDesc）、Object.defineProperties(proxy, propDescs)，返回一个布尔值。
- **`preventExtensions(target)`：**拦截Object.preventExtensions(proxy)，返回一个布尔值。
- **`getPrototypeOf(target)`：**拦截Object.getPrototypeOf(proxy)，返回一个对象。
- **`isExtensible(target)`：**拦截Object.isExtensible(proxy)，返回一个布尔值。
- **`setPrototypeOf(target, proto)`：**拦截Object.setPrototypeOf(proxy, proto)，返回一个布尔值。如果目标对象是函数，那么还有两种额外操作可以拦截。
- **`apply(target, object, args)`：**拦截 Proxy 实例作为函数调用的操作，比如proxy(...args)、proxy.call(object, ...args)、proxy.apply(...)。
- **`construct(target, args)`：**拦截 Proxy 实例作为构造函数调用的操作，比如new proxy(...args)。

博学谷
www.boxuegu.com

目录 Contents

- Proxy简介与常用方法
- Reflect简介与常用方法
- 综合应用场景

Reflect



1.Reflect简介

Reflect：为操作对象提供新的API。

Reflect对象的设计目的：

1）修改某些Object方法的返回结果，让其变得更合理。

例如：Object.defineProperty(obj,name,desc)在无法定义属性时，会抛出一个错误；

而Reflect.defineProperty(obj,name,desc)则会返回false。

```
//老写法
try{
    Object.defineProperty(target,property,attributes);
    //success
}catch(e){
    //failure
}

//新写法
if(Reflect.defineProperty(target,property,attributes)){
    //success
}else{
    //failure
}
```

1.Reflect简介

2）让Object操作都变成函数行为。

例如：某些Object操作是命令式，比如name in obj和delete obj[name]，而Reflect.has(obj,name)和Reflect.deleteProperty(obj,name)让其变成函数行为；目的是**为了让代码更加好维护，更容易向下兼容**。

```
//老写法
// 就是检查一个对象上是否含有特定的属性
'assign' in Object//true
//新写法
Reflect.has(Object,'assign')//true
```

1.Reflect简介

3）Reflect对象的方法与Proxy对象的方法一一对应；

只要是Proxy对象的方法，就能在Reflect对象上找到对应的方法。

Proxy对象可以方便地调用对应的Reflect方法，完成默认行为，作为修改行为的基础 。

```
let student = {
            name: 'zs',
            age: 20
        };
        let proxy = new Proxy(student, {
            set: function(obj, prop, value) {
                if (prop === 'age') {
                    if (!Number.isInteger(value)) {
                        throw new TypeError('年龄不是整数! ')
                    }
                    if (value > 60) {
                        throw new RangeError('年龄太大了')
                    }
                    // 如果输入的年龄符合规则，完成年龄的赋值
                    Reflect.set(obj, prop, value);
                }
            }
        })
        proxy.age = 30;
        console.log(proxy.age);
```

说明：

Proxy对象的set等方法完成了拦截操作这一行为，而Reflect对象中的set方法完成给对象属性赋值的这一行为。

1.Reflect简介

Reflect对象的方法与Proxy对象的方法一一对应，所以，**Reflect方法**如下所示：

```
Reflect.apply(target, thisArg, args)
Reflect.construct(target, args)
Reflect.get(target, name, receiver)
Reflect.set(target, name, value, receiver)
Reflect.defineProperty(target, name, desc)
Reflect.deleteProperty(target, name)
Reflect.has(target, name)
Reflect.ownKeys(target)
Reflect.isExtensible(target)
Reflect.preventExtensions(target)
Reflect.getOwnPropertyDescriptor(target, name)
Reflect.getPrototypeOf(target)
Reflect.setPrototypeOf(target, prototype)
```

2.has( )方法

Has( )方法：判断对象中是否有某个属性；

示例：

```
var myObject = {
        foo: 1,
    };
    // 旧写法
    console.log('foo' in myObject) // true
    // 新写法
    console.log(Reflect.has(myObject, 'foo')) // true
```

3.deleteProperty( )方法

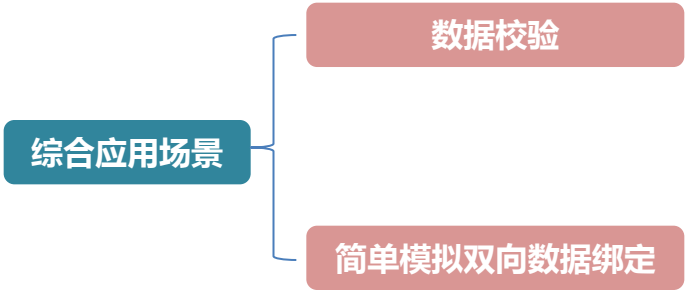
deleteProperty( )方法：**删除对象中某个属性；**

示例：

```
const myObj = {
            foo: 'bar'
        };
        // 旧写法
        delete myObj.foo;
        // 新写法
        Reflect.deleteProperty(myObj, 'foo');
```

目录 Contents

应用场景



1.数据校验

以一个比较完整的数据校验的案例进行实际演练；

要求：将对象的创建，数据验证的规则，以及具体的验证方式都进行分离；

2.简单模拟双向数据绑定

Vue中具有双向绑定功能；**通过proxy模拟双向数据绑定**：

要求：

用户在页面上修改数据会自动同步到对象数据模型中去；

如果对象数据模型中的值发生了变化，也会立即同步到页面上。

一样的在线教育，不一样的教学品质