代理与反射

代理模式

跟踪属性访问

通过get，set，has等操作符，可以知道对象属性什么时候被访问，被查询。
可以实现监控数据

```
const user = {
 name: 'Jake'
};
const proxy = new Proxy(user, {
 get(target, property, receiver) {
 console.log(`Getting ${property}`);
 return Reflect.get(...arguments);
 },
 set(target, property, value, receiver) {
 console.log(`Setting ${property}=${value}`);
 return Reflect.set(...arguments);
 }
});
proxy.name; // Getting name
proxy.age = 27; // Setting age=27
```

隐藏属性

代理的内部实现对外部代码不可见，可以隐藏目标对象的属性

```
const hiddenProperties = ['foo', 'bar'];
const targetObject = {
    foo: 1,
    bar: 2,
    baz: 3
};
const proxy = new Proxy(targetObject, {
    get(target, property) {
        if (hiddenProperties.includes(property)) {
            return undefined;
        } else {
            return Reflect.get(...arguments);
        }
    },
    has(target, property) {
        if (hiddenProperties.includes(property)) {
            return false;
        } else {
            return Reflect.has(...arguments);
        }
```

```
    }
});
// get()
console.log(proxy.foo); // undefined
console.log(proxy.bar); // undefined
console.log(proxy.baz); // 3
// has()
console.log('foo' in proxy); // false
console.log('bar' in proxy); // false
console.log('baz' in proxy); // true
```

属性验证

所有赋值操作触发set(),可以根据所赋的值决定允许还是拒绝

```
const target = {
    onlyNumbersGoHere: 0
};
const proxy = new Proxy(target, {
    set(target, property, value) {
        if (typeof value !== 'number') {
            return false;
        } else {
            return Reflect.set(...arguments);
        }
    }
});
proxy.onlyNumbersGoHere = 1;
console.log(proxy.onlyNumbersGoHere); // 1
proxy.onlyNumbersGoHere = '2';
console.log(proxy.onlyNumbersGoHere); // 1
```

函数与构造函数参数验证

可以对函数与构造函数的参数进行验证 与 属性验证类似

```
function median(...nums) {
    return nums.sort()[Math.floor(nums.length / 2)];
}
const proxy = new Proxy(median, {
    apply(target, thisArg, argumentsList) {
        for (const arg of argumentsList) {
            if (typeof arg !== 'number') {
                throw 'Non-number argument provided';
            }
        }
        return Reflect.apply(...arguments);
    }
});
console.log(proxy(4, 7, 1)); // 4
console.log(proxy(4, '7', 1));
// Error: Non-number argument provided
```

构造函数参数验证

```
class User {
    constructor(id) {
        this.id_ = id;
    }
}
const proxy = new Proxy(User, {
    construct(target, argumentsList, newTarget) {
        if (argumentsList[0] === undefined) {
            throw 'User cannot be instantiated without id';
        } else {
            return Reflect.construct(...arguments);
        }
    }
});
new proxy(1);
new proxy();
   // Error: User cannot be instantiated without id
```

数据绑定与可观察对象

以将被代理的类绑定到一个全局实例集合，让所有创建的实例都被添加到这个集合中

```
const userList = [];
class User {
    constructor(name) {
        this.name_ = name;
    }
}
const proxy = new Proxy(User, {
    construct() {
        const newUser = Reflect.construct(...arguments);
        userList.push(newUser);
        return newUser;
    }
});
new proxy('John');
new proxy('Jacob');
new proxy('Jingleheimerschmidt');
console.log(userList); // [User {}, User {}, User{}]
```

把集合绑定到一个事件分派程序，每次插入新实例时都会发送消息

```
const userList = [];
function emit(newValue) {
    console.log(newValue);
}
const proxy = new Proxy(userList, {
    set(target, property, value, receiver) {
        const result = Reflect.set(...arguments);
        if (result) {
            emit(Reflect.get(target, property, receiver));
        }
        return result;
    }
});
```

```
proxy.push('John');
// John
proxy.push('Jacob');
// Jacob
```

总结

捕获器
捕获器不变式
代理模式