一.为什么要取代Object.defineProperty

既然要取代Object.defineProperty，那它肯定是有一些明显的缺点，总结起来大概是下面两个：

> - Object.defineProperty无法监控到数组下标的变化，导致直接通过数组的下标给数组设置值，不能实时响应。 为了解决这个问题，经过vue内部处理后可以使用以下几种方法来监听数组

```
push()
pop()
shift()
unshift()
splice()
sort()
reverse()
```

由于只针对了以上八种方法进行了hack处理,所以其他数组的属性也是检测不到的，还是具有一定的局限性。

> - Object.defineProperty只能劫持对象的属性,因此我们需要对每个对象的每个属性进行遍历。Vue 2.x里，是通过 递归 + 遍历 data 对象来实现对数据的监控的，如果属性值也是对象那么需要深度遍历,显然如果能劫持一个完整的对象是才是更好的选择。

而要取代它的Proxy有以下两个优点;

> - 可以劫持整个对象，并返回一个新对象
> - 有13种劫持操作

看到这可能有同学要问了，既然Proxy能解决以上两个问题，而且Proxy作为es6的新属性在vue2.x之前就有了，为什么vue2.x不使用Proxy呢？一个很重要的原因就是：

> - Proxy是es6提供的新特性，兼容性不好，最主要的是这个属性无法用polyfill来兼容

相信尤大大在vue3.0的版本中会有效的提供兼容解决方案。

关于Object.defineProperty来实现观察者机制，可以参照剖析Vue原理&实现双向绑定MVVM这篇文章，下面的内容主要介绍如何基于 Proxy来实现vue观察者机制。

二.什么是Proxy

1.含义：

> - Proxy是 ES6 中新增的一个特性，翻译过来意思是"代理"，用在这里表示由它来“代理”某些操作。
> Proxy 让我们能够以简洁易懂的方式控制外部对对象的访问。其功能非常类似于设计模式中的代理模式。
> - Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。
> - 使用 Proxy 的核心优点是可以交由它来处理一些非核心逻辑（如：读取或设置对象的某些属性前记录日志；设置对象的某些属性值前，需要验证；某些属性的访问控制等）。 从而可以让对象只需关注于核心逻辑，达到关注点分离，降低对象复杂度等目的。

2.基本用法：

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

参数：

> `target` 是用Proxy包装的被代理对象（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）。

> `handler` 是一个对象，其声明了代理target 的一些操作，其属性是当执行一个操作时定义代理的行为的函数。

> `p` 是代理后的对象。当外界每次对 p 进行操作时，就会执行 handler 对象上的一些方法。Proxy共有13种劫持操作，handler代理的一些常用的方法有如下几个：

```
get：读取
set：修改
has：判断对象是否有该属性
construct：构造函数
```

3.示例：

下面就用Proxy来定义一个对象的get和set，作为一个基础demo

```
let obj = {};
let handler = {
  get(target, property) {
   console.log(`${property} 被读取`);
   return property in target ? target[property] : 3;
  },
  set(target, property, value) {
   console.log(`${property} 被设置为 ${value}`);
   target[property] = value;
  }
}

let p = new Proxy(obj, handler);
p.name = 'tom' //name 被设置为 tom
p.age; //age 被读取 3
```

> p 读取属性的值时，实际上执行的是 handler.get()： 在控制台输出信息，并且读取被代理对象 obj 的属性。

> p 设置属性值时，实际上执行的是 handler.set()： 在控制台输出信息，并且设置被代理对象 obj 的属性的值。

以上介绍了Proxy基本用法，实际上这个属性还有许多内容，具体可参考Proxy文档

三.基于Proxy来实现双向绑定

话不多说，接下来我们就来用Proxy来实现一个经典的双向绑定todolist，首先简单的写一点html结构：

```
 <div id="app">
     <input type="text" id="input" />
     <div>您输入的是： <span id="title"></span></div>
     <button type="button" name="button" id="btn">添加到todolist</button>
     <ul id="list"></ul>
</div>
```

先来一个Proxy，实现输入框的双向绑定显示：

```
    const obj = {};
    const input = document.getElementById("input");
    const title = document.getElementById("title");

    const newObj = new Proxy(obj, {
      get: function(target, key, receiver) {
        console.log(`getting ${key}!`);
        return Reflect.get(target, key, receiver);
      },
      set: function(target, key, value, receiver) {
        console.log(target, key, value, receiver);
        if (key === "text") {
          input.value = value;
          title.innerHTML = value;
        }
        return Reflect.set(target, key, value, receiver);
      }
    });

    input.addEventListener("keyup", function(e) {
      newObj.text = e.target.value;
    });
```

这里代码涉及到 `Reflect` 属性，这也是一个es6的新特性，还不太了解的同学可以参考Reflect文档.
接下来就是添加todolist列表，先把数组渲染到页面上去：

```
// 渲染todolist列表
    const Render = {
      // 初始化
      init: function(arr) {
        const fragment = document.createDocumentFragment();
        for (let i = 0; i < arr.length; i++) {
          const li = document.createElement("li");
          li.textContent = arr[i];
          fragment.appendChild(li);
        }
        list.appendChild(fragment);
      },
      addList: function(val) {
        const li = document.createElement("li");
        li.textContent = val;
        list.appendChild(li);
      }
    };
```

再来一个Proxy，实现Todolist的添加：

```
const arr = [];
    // 监听数组
    const newArr = new Proxy(arr, {
      get: function(target, key, receiver) {
        return Reflect.get(target, key, receiver);
      },
      set: function(target, key, value, receiver) {
        console.log(target, key, value, receiver);
        if (key !== "length") {
```

```
          Render.addList(value);
        }
        return Reflect.set(target, key, value, receiver);
      }
    });

    // 初始化
    window.onload = function() {
      Render.init(arr);
    };

    btn.addEventListener("click", function() {
      newArr.push(parseInt(newObj.text));
    });
```

这样就用 Proxy实现了一个简单的双向绑定Todolist,具体代码可参考proxy.html

四.基于Proxy来实现vue的观察者机制

1.Proxy实现observe

```
    observe(data) {
        const that = this;
        let handler = {
         get(target, property) {
            return target[property];
          },
          set(target, key, value) {
            let res = Reflect.set(target, key, value);
            that.subscribe[key].map(item => {
              item.update();
            });
            return res;
          }
        }
        this.$data = new Proxy(data, handler);
      }
```

这段代码里把代理器返回的对象代理到`this.$data`，即`this.$data`是代理后的对象，外部每次对`this.$data`进行操作时，实际上执行的是这段代码里handler对象上的方法。

2.compile和watcher

> 比较熟悉vue的同学都很清楚，vue2.x在 new Vue() 之后。 Vue 会调用 _init 函数进行初始化，它会初始化生命周期、事件、 props、 methods、 data、 computed 与 watch 等。其中最重要的是通过 Object.defineProperty 设置 setter 与 getter 函数，用来实现「响应式」以及「依赖收集」。类似于下面这个内部流程图：

image

> 而我们上面已经用Proxy取代了Object.defineProperty这部分观察者机制，而要实现整个基本mvvm双向绑定流程，除了observe还需要compile和watche等一系列机制，我们这里像模板编译的工作就不展开描述了，为了实现基于Proxy的vue添加Totolist,这里只写了compile和watcher来支持observe的工作，具体代码参考proxyVue,这个代码相当于一个基于Proxy的一个简化版vue，主要是实现双向绑定这个功能，为了方便这里把js放到了html页面中，大家本地运行后可以发现，现在的效果和第三章的效果达到一致了，等到明年vue3发布，它源码

> 里基于 Proxy实现的的观察者机制可能和这里的实现会有很多不同，这篇文章主要是对 Proxy这个特性做了一些介绍以及它的一些应用，而作者本人也通过对Proxy 的观察者机制探索学到了不少东西，所以整合资源，总结出了这篇文章，希望能和大家共勉之，以上，我们下次有缘再见。