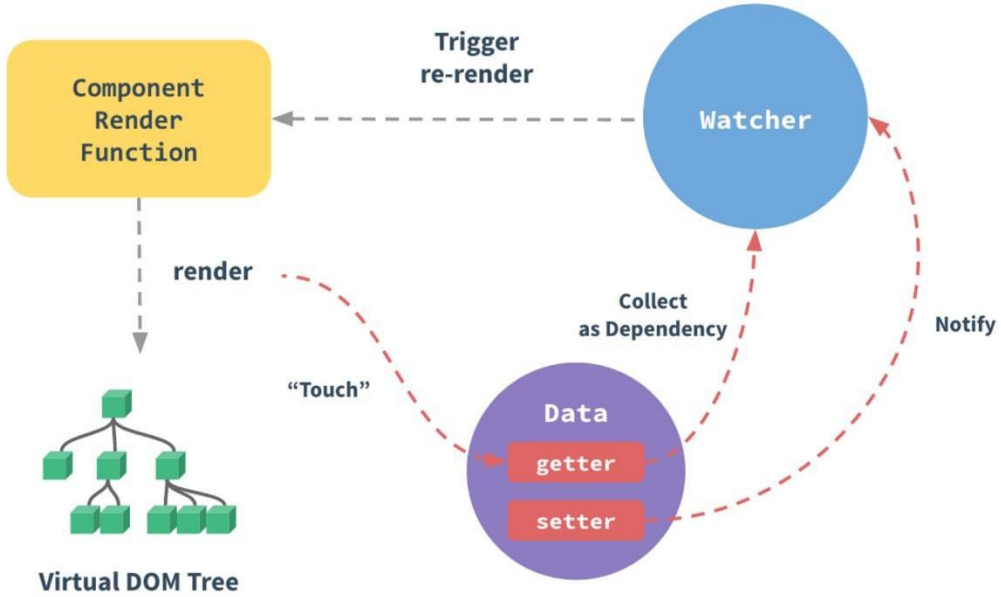
Vue源码探秘之

数据响应式原理

课程简介

彻底弄懂Vue2的数据更新原理，手写相关实现代码，让相关知识不再处于“忽悠阶段”



模板

```
<p>我{{age}}岁了</p>
```

数据变化

```
this.age++;
```

数据变化，视图会自动变化

Vue数据变化

```
this.a ++;
```

非侵入式

React数据变化

```
this.setState({
    a: this.state.a + 1
});
```

小程序数据变化

```
this.setData({
    a: this.data.a + 1
});
```

侵入式

Object.defineProperty()

数据劫持/ 数据代理

利用JavaScript引擎赋予的功能，检测对象属性变化

仅有“上帝的钥匙”不够，还需要设计一套精密的系统

Object.defineProperty()方法

Object.defineProperty()方法

Object.defineProperty()方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性，并返回此对象。

```
var obj = {};

Object.defineProperty(obj, 'a', {
    value: 3
});

Object.defineProperty(obj, 'b', {
    value: 5
});

console.log(obj);
console.log(obj.a, obj.b);
```

Object.defineProperty()方法可以设置一些额外隐藏的属性。

```
Object.defineProperty(obj, 'a', {
    value: 3,
    // 是否可写
    writable: false
});
```

```
Object.defineProperty(obj, 'b', {
    value: 5,
    // 是否可以被枚举
    enumerable: false
});
```

get

属性的 getter 函数，如果没有 getter，则为 `undefined`。当访问该属性时，会调用此函数。执行时不传入任何参数，但是会传入 `this` 对象（由于继承关系，这里的`this`并不一定是定义该属性的对象）。该函数的返回值会被用作属性的值。

默认为 `undefined`。

set

属性的 setter 函数，如果没有 setter，则为 `undefined`。当属性值被修改时，会调用此函数。该方法接受一个参数（也就是被赋予的新值），会传入赋值时的 `this` 对象。

默认为 `undefined`。

这里有一个小坑：用闭包存储get和set的值

```
Object.defineProperty(obj, 'a', {
    // getter
    get() {
        console.log('你试图访问obj的a属性');
    },
    // setter
    set() {
        console.log('你试图改变obj的a属性');
    }
});
console.log(obj.a);
obj.a = 10;
```

defineReactive函数

getter/setter需要变量周转才能工作

```
var temp;

Object.defineProperty(obj, 'a', {
    // getter
    get() {
        console.log('你试图访问obj的a属性');
        return temp;
    },
    // setter
    set(newValue) {
        console.log('你试图改变obj的a属性', newValue);
        temp = newValue;
    }
});
```

使用defineReactive函数不需要设置临时变量了，而是用闭包

```
function defineReactive(data, key, val) {
    Object.defineProperty(data, key, {
        // 可枚举
        enumerable: true,
        // 可以被配置，比如可以被delete
        configurable: true,
        // getter
        get() {
            console.log('你试图访问obj的' + key + '属性');
            return val;
        },
        // setter
        set(newValue) {
            console.log('你试图改变obj的' + key + '属性', newValue);
            if (val === newValue) {
                return;
            }
            val = newValue;
        }
    });
}
```

递归侦测对象全部属性

Observer

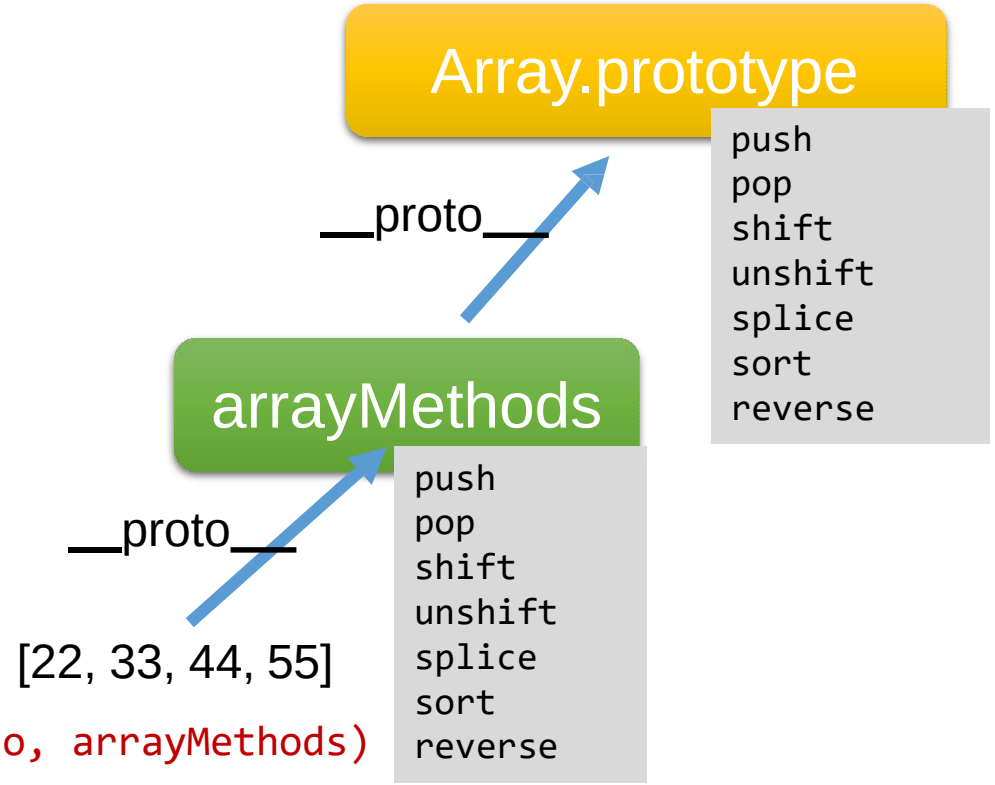
将一个正常的object转换为每个层级的属性都是响应式（可以被侦测的）的object

```
var obj = {
    a: {
        m: {
            n: 5
        }
    },
    b: 4
};
```

数组的响应式处理

改写七个方法

- push
- pop
- shift
- unshift
- splice
- sort
- reverse



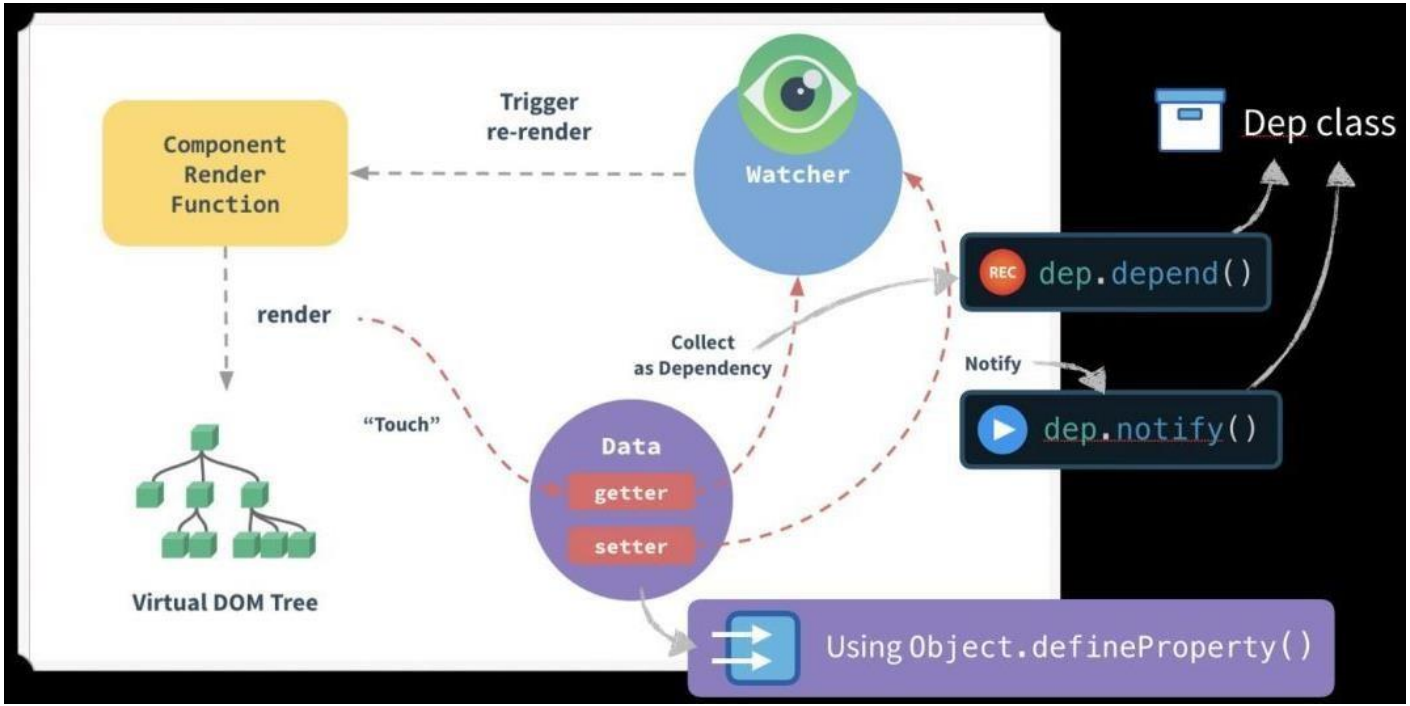

```
Object.setPrototypeOf(o, arrayMethods)
o.__proto__= arrayMethods
```

依赖收集

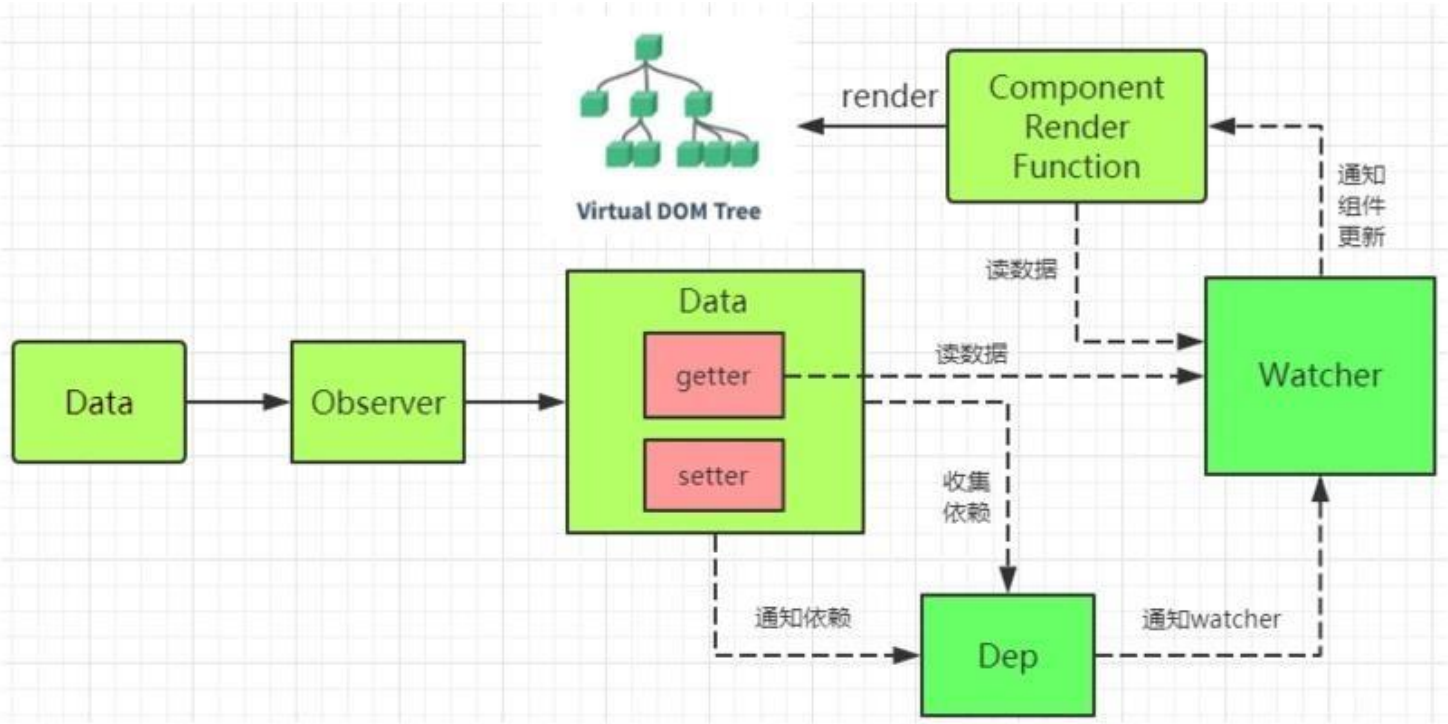
- 需要用到数据的地方，称为依赖
- Vue1.x，细粒度依赖，用到数据的DOM都是依赖；
- Vue2.x，中等粒度依赖，用到数据的组件是依赖；
- 在getter中收集依赖，在setter中触发依赖

Dep类和Watcher类

- 把依赖收集的代码封装成一个Dep类，它专门用来管理依赖，**每个Observer的实例，成员中都有一个Dep的实例；**
- Watcher是一个中介，数据发生变化时通过Watcher中转，通知组件



- 依赖就是Watcher。只有Watcher触发的getter才会收集依赖，哪个Watcher触发了getter，就把哪个Watcher收集到Dep中。
- Dep使用发布订阅模式，当数据发生变化时，会循环依赖列表，把所有的Watcher都通知一遍。
- 代码实现的巧妙之处：Watcher把自己设置到全局的一个指定位置，然后读取数据，因为读取了数据，所以会触发这个数据的getter。在getter中就能得到当前正在读取数据的Watcher，并把这个Watcher收集到Dep中。



图片来自：https://blog.csdn.net/Mikon_0703/article/details/111367773