```
<script>
export default {
  data() {
    return {
      name: "John Doe"
    };
  },
  render(h) {
    return h("div", this.name);
  }
};
</script>
```

```
let originalValue = this.data.name;

Object.defineProperty(this.data, 'name', {
    get() {
        // do something
        return originalValue;
    },

    set(newValue) {
        // do something
        originalValue = newValue;
    }
});
```

```
this.data.name
```

```
this.data.name = 'Richard Roe'
```

```
this.name = 'Richard Roe'
```

```
{
    tag: 'div',
    children: [‘Richard Roe’]
}
```

Richard Roe

vue updates DOM

```
<script>
export default {
  data() {
    return {
      name: "John Doe"
    };
  },
  render(h) {
    return h("div", this.name);
  }
};
</script>
```

```
let originalValue = this.data.name;

Object.defineProperty(this.data, 'name', {
    get() {
        // do something
        return originalValue;
    },

    set(newValue) {
        // do something
        originalValue = newValue;
    }
});
```

```
/**
 * Define a reactive property on an Object.
 */
export function defineReactive (
  obj: Object,
  key: string,
  val: any,
  customSetter?: ?Function,
  shallow?: boolean
) {
  const dep = new Dep()

  const property = Object.getOwnPropertyDescriptor(obj, key)
  if (property && property.configurable === false) {
    return
  }

  // cater for pre-defined getter/setters
  const getter = property && property.get
  const setter = property && property.set
  if ((!getter || setter) && arguments.length === 2) {
    val = obj[key]
  }

  let childOb = !shallow && observe(val)
  Object.defineProperty(obj, key, {
    enumerable: true,
    configurable: true,
    get: function reactiveGetter () {
      const value = getter ? getter.call(obj) : val
      if (Dep.target) {
        dep.depend()
```