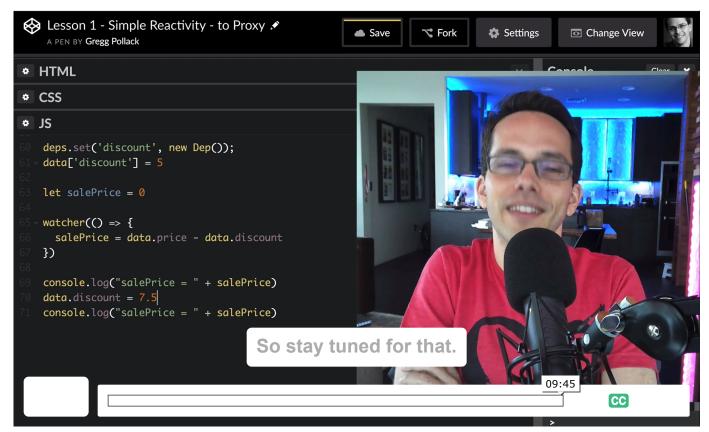


Don't stop your Vue learning. Get **30% Off** an annual subscription this weekend only by using code **WEEKEND30**.





Advanced Components



Lessons

3. Evan You on Proxies © 9:48	✓
2. Build a Reactivity System ① 16:50	✓
1. Introduction ① 1:43	∀

4. Reactivity in Vue.js © 13:44	
5. Evan You on Vue Core © 10:18	
6. Template Compilation © 14:15	
7. Evan You on the Virtual DOM © 5:05	
8. Functional Components © 5:47	
9. Evan You on Functional Components © 2:33	
10. The Mounting Process © 10:31	
11. Evan You on the Mounting Process © 8:32	
12. Scoped Slots & Render Props © 9:22	

Evan You on Proxies

In the previous video we built out a reactivity system that mimics the Vue.js implementation of reactivity. The use of <code>Object.defineProperty()</code> to convert the properties into <code>getters/setters</code> allowed us to track them as dependencies when accessed and then rerun code (notify) when modified.

If you have been following the Vue roadmap, the 2.x-next version's reactivity system will be rewritten with Proxies, which is different than what I showed.

Core

2.6

Various improvements regarding error handling, functional components, SSR

2.x-next

- Will be targeting evergreen browsers only in order to leverage native ES2015 features
- Reactivity system will be rewritten with Proxies with various improvements
- No major breaking changes; will be maintained in parallel to 2.x with feature parity

I wanted to ask Evan what exactly this might look like and the advantages we get from it.

What are the Advantages?

The proxy API allows us to create a virtual representation of an object and provides us with handlers like set(), get() and deleteProperty() etc that we can use to intercept when properties are accessed or modified on the original object. This relieves us from the following limitations:

- Usage of Vue.\$set() to add new reactive properties and Vue.\$delete() to delete existing properties.
- Array change detection.

Our Previous Code

Previously we used Object.defineProperty() to listen for when our properties are get and set. Here is a codepen which shows where we ended up on the last lesson:

```
let data = { price: 5, quantity: 2 };
let target = null;

// Our simple Dep class
class Dep {
```

```
constructor() {
    this.subscribers = [];
  }
  depend() {
    if (target && !this.subscribers.includes(target)) {
      // Only if there is a target & it's not already subscribed
      this.subscribers.push(target);
    }
  }
 notify() {
    this.subscribers.forEach(sub => sub());
  }
}
// Go through each of our data properties
Object.keys(data).forEach(key => {
  let internalValue = data[kev];
  // Each property gets a dependency instance
  const dep = new Dep();
  Object.defineProperty(data, key, {
    get() {
      dep.depend(); // <-- Remember the target we're running</pre>
      return internalValue;
    },
    set(newVal) {
      internalValue = newVal;
      dep.notify(); // <-- Re-run stored functions</pre>
 });
});
// The code to watch to listen for reactive properties
function watcher(myFunc) {
  target = myFunc;
 target();
 target = null;
}
watcher(() => {
  data.total = data.price * data.quantity;
});
console.log("total = " + data.total)
data.price = 20
console.log("total = " + data.total)
data.quantity = 10
```

Solution: Using Proxy to overcome the limitations

Instead of looping through each property to add getters/setters we can set up a proxy on our data object using:

```
//data is our source object being observed
const observedData = new Proxy(data, {
   get() {
      //invoked when property from source data object is accessed
   },
   set() {
      //invoked when property from source data object is modified
   },
   deleteProperty() {
      //invoked when property from source data object is deleted
   }
});
```

The second argument passed to Proxy constructor function is called the handler. Handler is nothing but an object that contains functions known as traps. These traps allow us to intercept operations happening on the source data object.

The <code>get()</code> and <code>set()</code> are two traps that can be used to invoke <code>dep.depend()</code> and <code>dep.notify()</code> respectively. The <code>set()</code> trap will be invoked even for the newly added properties, so it can be it can be used to make new properties reactive. Hence, we no longer need to declare new reactive properties using <code>Vue.\$set()</code>. The same applies for deletion of reactive properties which can be handled in <code>deleteProperty()</code> trap.

Implementing the Reactivity System Using Proxies

Even though the Proxy API is not yet incorporated into Vue's reactivity system, let's try to implement the reactivity system from the previous lesson using Proxy ourselves. The first thing we'll change is our Object.keys(data).forEach loop, which we'll now use to create a new Dep for each reactive property.

```
let deps = new Map(); // Let's store all of our data's deps in a
map
Object.keys(data).forEach(key => {
```

```
// Each property gets a dependency instance
deps.set(key, new Dep());
});
```

Side Note: The Dep class remains the same. Now we'll replace the use of Object.defineProperty with the use of a proxy:

As you can see, we create a variable data_without_proxy that holds the copy of our source data object which will be used when overwriting the data object to have a Proxy object. The get() and set() traps are passed in as the properties to handler object which is the 2nd argument.

get(obj, key) This is the function that gets invoked when a property is accessed. It receives the original object i.e data_without_proxy as obj and the key of the property that is accessed. We call the depend() method of the specific Dep class associated to that particular property. At last the value related to that key is returned using return obj[key].

set(obj, key, newVal) The first two arguments are the same as the above-mentioned get() trap.
The 3rd argument is the new modified value. Then we set the new value to the property that is
modified using obj[key] = newVal and call the notify() method.

Moving Total & Testing

We need to make one more small change to our code. We need to extract total into its own variable as it does not need to be reactive.

```
let total = 0;
watcher(() => {
   total = data.price * data.quantity;
});
console.log("total = " + total);
data.price = 20;
console.log("total = " + total);
data.quantity = 10;
console.log("total = " + total);
```

Now when we re-run the program, we see the following output in the console:

```
total = 10
total = 40
total = 200
```

That's a good sign. The total updates when we update the price and quantity.

Adding Reactive Properties

Now we should be able to add properties into data without declaring them upfront. That was one of the reasons for considering proxies over getters/setters right? Let's try it out.

We can add the following code:

```
deps.set("discount", new Dep()); // Need a new dep for our
property
   data["discount"] = 5; // Add our new property

let salePrice = 0;

watcher(() => { // New code to watch which includes our reactive property
        salePrice = data.price - data.discount;
   });

console.log("salePrice = " + salePrice);
   data.discount = 7.5; // This should be reactive, and rerun the watcher.
   console.log("salePrice = " + salePrice);
```

and of the state of

When the program is run we can see the following output:

```
salePrice = 15
salePrice = 12.5
```

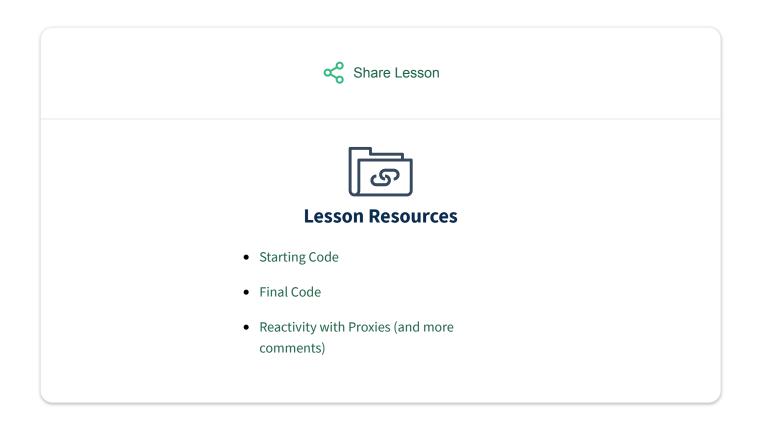
You can see that when the data.discount is modified the salePrice also gets updated. Hurray! The finished code can be seen here.

ReVue

In this lesson Evan You spoke to us about how future versions of Vue (v2.6-next) could implement reactivity using Proxies. We learned more about:

- · The limitations of current reactivity system
- How proxies work
- How to build a reactivity system using proxies

In the next video, we'll dive into the Vue source code and discover where reactivity lies.











The Official Vue.js News

We also help produce the Official Vue.js News. It's a free community resource where we curate the most impactful Vue.js news and tutorials. Offered up in both text and audio versions. Consider subscribing today.

Read Official Vue.js News

Listen to the Podcast

VUE MASTERY

Courses

Conference Videos

Live Training

Pricing

Vue Cheat Sheet

ABOUT US

About

Contact

Privacy Policy

Terms of Service

Nuxt Cheat Sheet



As the ultimate resource for Vue.js developers, Vue Mastery produces weekly lessons so you can learn what you need to succeed as a Vue.js Developer.







