How to Call a Vue Method on Page Load

As soon as the page loads, you want a certain function to be called.

Maybe you're fetching data, or you're initializing something.

How do we do this in Vue?

You'll want to use the `mounted` lifecycle hook so that you can run code as soon as your component is mounted to the DOM. From this lifecycle hook you can fetch data, manipulate the DOM, or do anything else you might need in order to initialize your component.

This chapter will explain a few things:

- What lifecycle hooks are
- How to use lifecycle hooks
- Why you should prefer using the `mounted` hook over the `created` hook

In order to call a function as soon as our Vue component has loaded, we'll first need to get familiar with Vue's lifecycle hooks.

Lifecycle Hooks

All Vue components have a series of stages — or **lifecycles** — that they go through.

As your app is run, your component will be:

- Created
- Mounted
- Updated
- Destroyed

Let's take a quick look at each of these 4 stages.

Lifecycles at a glance

First, the component is **created**.

Here, everything — including reactive `data`, computed props, and watchers — are setup and intialized.

Second, the component is **mounted** to the DOM.

This involves creating the actual DOM nodes and inserting your component into the page.

Third, your component is **updated** as reactive data changes. Vue will re-render your component many times, in order to keep everything on the page up-to-date.

Lastly, when the component is no longer needed, it is **destroyed**.

Any event listeners are cleaned up, DOM nodes are removed from the page, and any memory it was using is now released.

That's not all.

Vue let's us hook into these lifecycles. This lets us run code when the component is **created**, **mounted**, **updated**, or **destroyed**!

> **There is so much more to talk about when it comes to lifecycle methods. I would suggest you check out the docs as well as this awesome article about them to learn even more.**

Hooking into a lifecycle

So how do we use a lifecycle hook?

All we have to do is create a method on our component that uses the name of that lifecycle.

If we wanted to call a method right when the component is **created**, this is how we would do that:

```
export default {
  created() {
    console.log('Component has been created!');
  }
};
```

Similarly, hooking into the **destroyed** lifecycle works like this:

```
export default {
  destroyed() {
    console.log('Component has been destroyed!');
  }
};
```

But we're here to figure out **how to call a function as soon as our page loads.**

Both `created` and `mounted` hooks seem like they would work.

Which is the best one to use?

Using the Mounted Hook

In nearly all cases, the `mounted` hook is the right one for this job.

In fact, I always use this one. I only switch to a different hook if for some reason the `mounted` hook doesn't work for what I am trying to do.

The reason is this.

In the `mounted` hook, everything about the component has been initialized properly.

Props have been initialized, reactive `data` is going, computed props have been setup, and so have your watchers.

Most importantly though, Vue has setup everything in the DOM.

This means that you can safely do whatever you need to do in this lifecycle hook.

Using the Created Hook

There can often be some confusion around the differences between the `created` and `mounted` hooks.

As we saw earlier, the `created` hook is run before the `mounted` hook is run.

And in the `created` hook, nearly everything in the component has been setup. The only thing missing is the DOM.

So what is the `created` hook good for?

Well, since we don't have access to our DOM element, we should use `created` for anything that doesn't need the DOM element.

Oftentimes it is used to fetch data:

```
export default {
  data() {
    cars: {},
  },
  created() {
    fetch('/cars').then(cars => {
      this.cars = cars;
    });
  }
};
```

The advantage of using `created` instead of `mounted` is that `created` will be called a little sooner. This means you'll get your data just a tiny bit faster.

Mounting the Component

Earlier I said that the `created` hook doesn't have access to the DOM.

Let me explain why that is.

Vue first **creates** the component, and then it **mounts** the component to the DOM. We can only access the DOM **after** the component has been mounted. This is done using the `mounted` hook.

You can prove this to yourself by putting the following into your component:

```
created() {
  console.log(this.$el);
},
mounted() {
  console.log(this.$el);
}
```

What was logged out?

You should have gotten `undefined`, followed by a DOM element.

The DOM element for our component is set to `this.$el`, but it doesn't exist yet in the `created` hook.

However, it **does** exist in our `mounted` hook, because Vue has already done the work of adding it to the DOM by the time it is called.