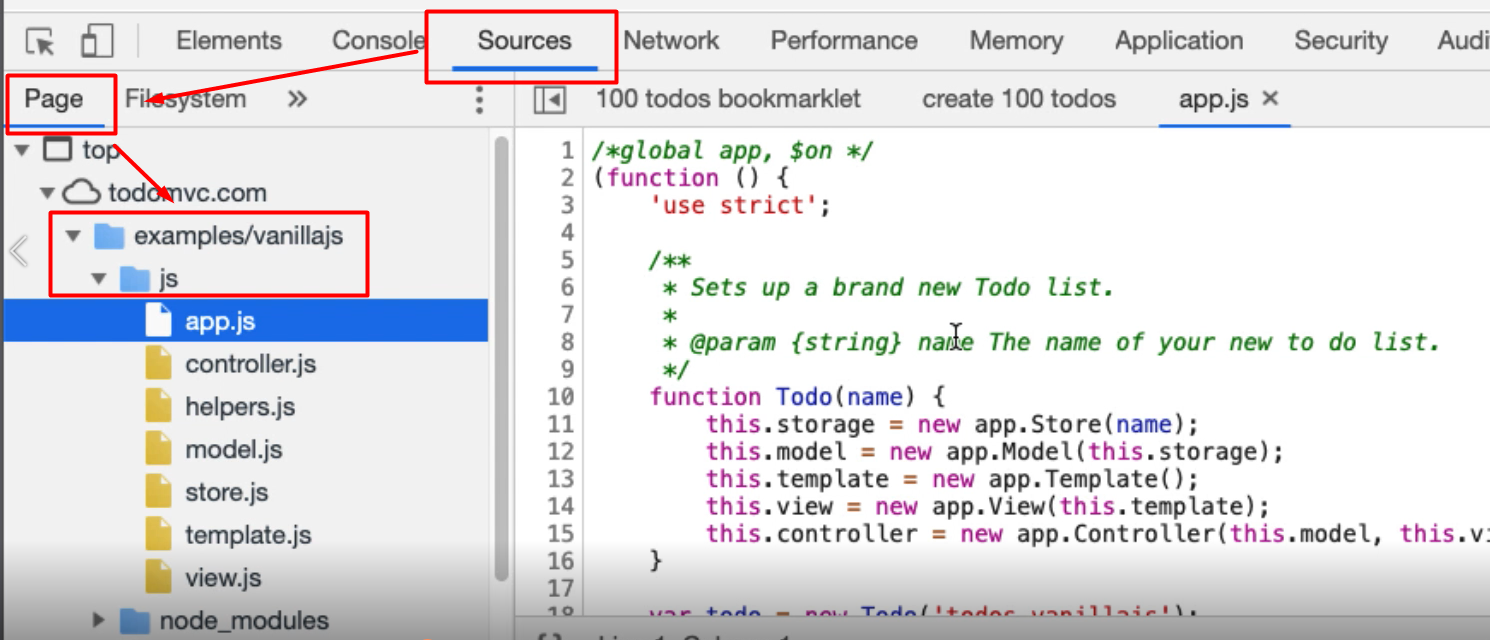
Let's start having a quick look at some advanced functionality, because what we've seen so far is essentially just the basics.

What we have been doing when we have been automating is… we've been triggering events. Everything we've been doing is external through the application. We've been simulating what the user does. Simulating what happens when the user clicks on something.

Very often when I'm automating JavaScript applications, I'll actually use the JavaScript code against itself because the JavaScript application knows how to create to-do's. If I could use that code, I wouldn't have to write so many events, and it would be a little bit easier. I would be testing “under the covers.”

When we normally consider automated execution, we very often talk about don't automate the GUI because it's brittle, automate at the API. Do more unit testing. Now, here we don't have an API, but we can simulate that if we had access to the innards.

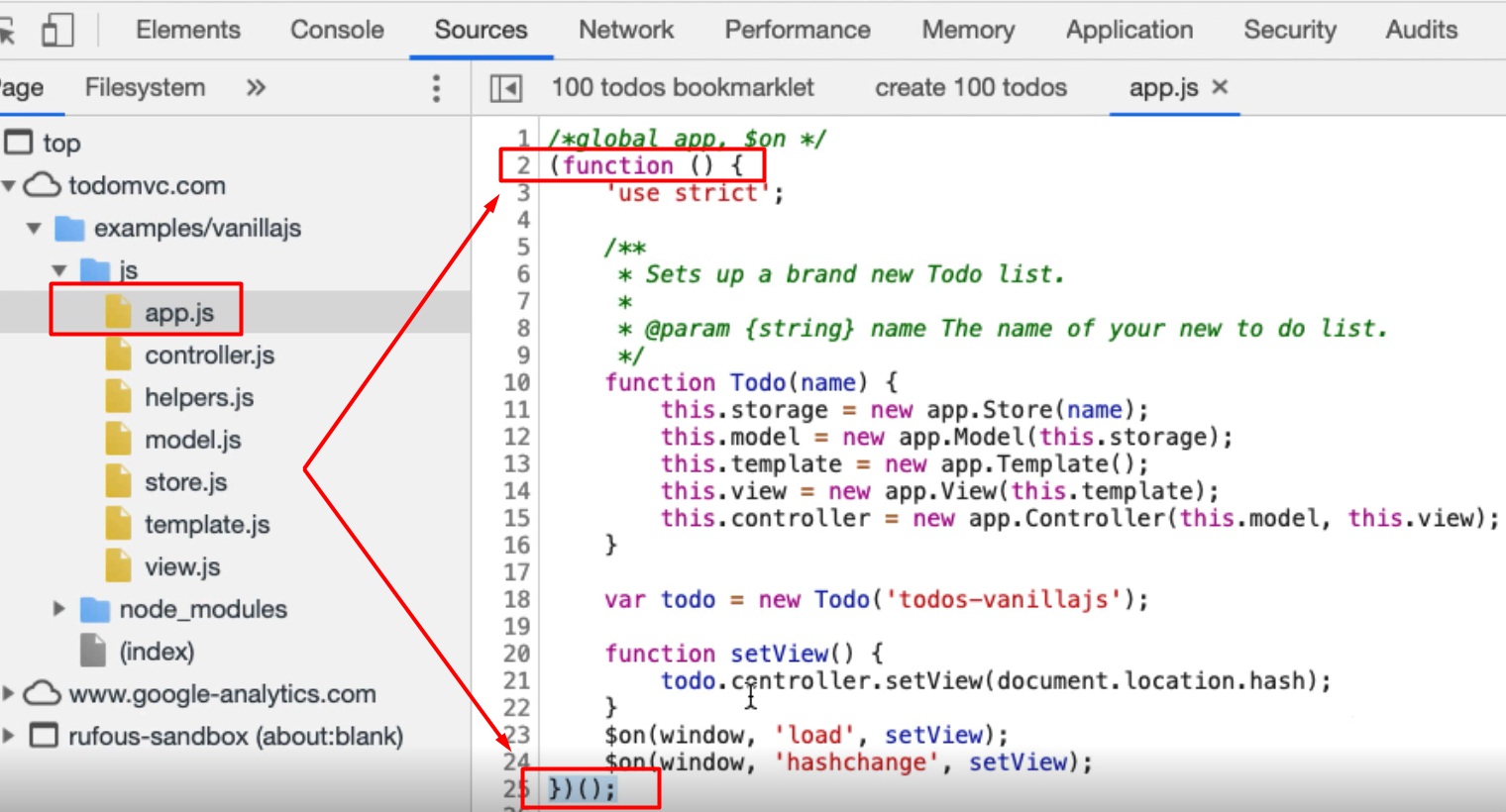
If I have a quick look at the source code for this application, so on *Sources > Page*, I can see all the code for the app.



And there, in the *example/vanillajs > js folder*, is all the code for the app.

Now this app.js has a top-level app object.

Problem is, and you've seen this pattern from the bookmarklets, this is an anonymous function that runs immediately. The only reason it works is because it's setting up hooks onto the page to do the work.

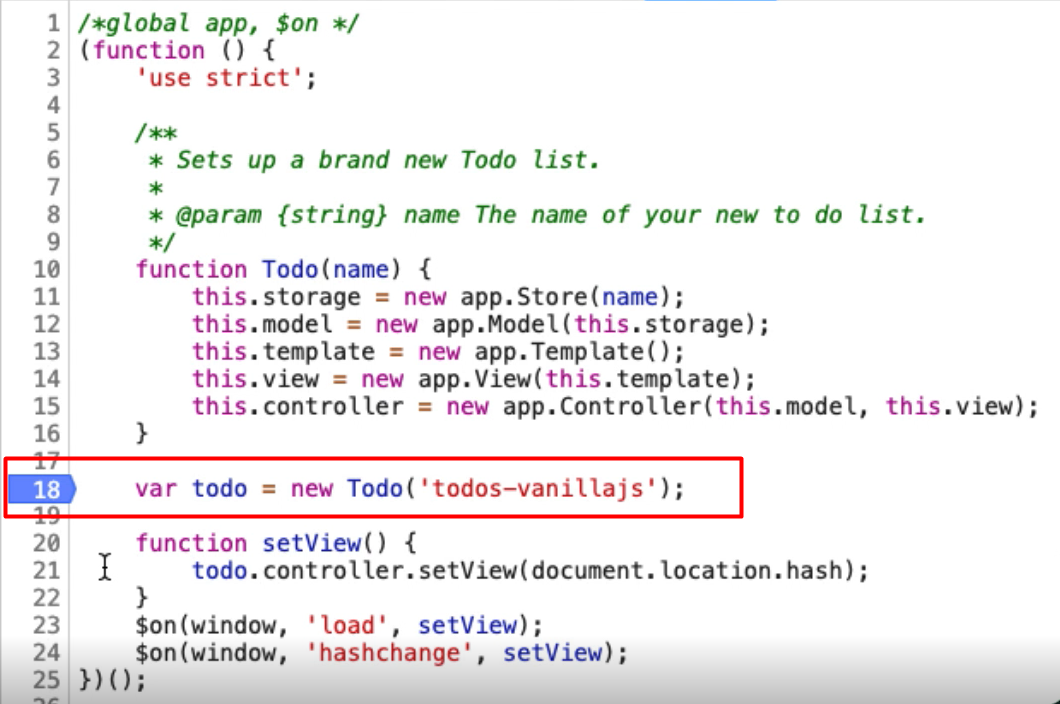


It would be really useful if we had access to this Todo object because then we'd have access to the controller, and the model and the view and we will be able to do things in the actual application, but we don't.

If I go into the *Console* and say Todo, I don't have access to that object. This is a very common pattern that people, when they're coding JavaScript, use; it creates a little bit more protection so that the user can’t access the innards.

**I'm just going to show you a quick way to bypass this.**

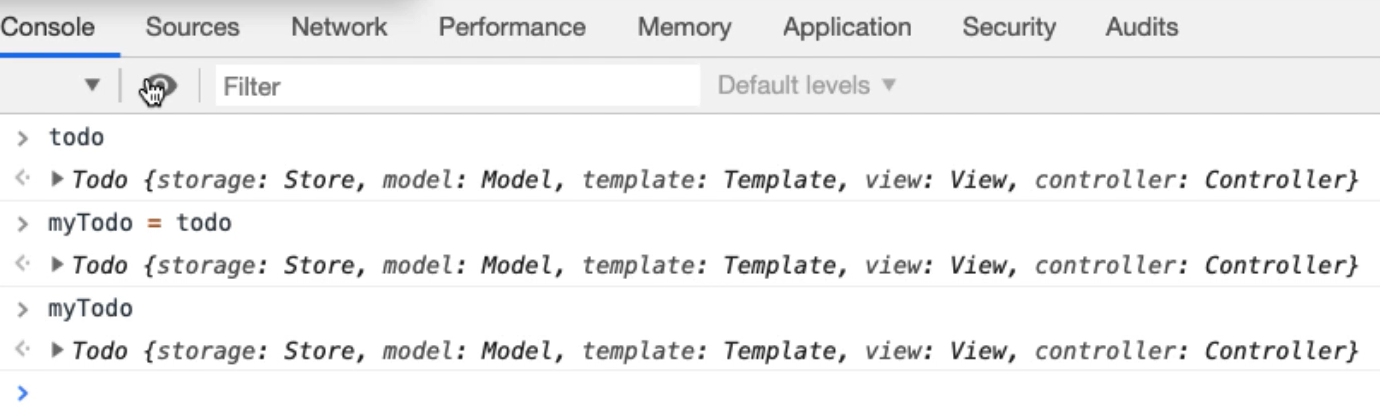
If I've got a breakpoint on this line in the code,



and then refresh the page, what happens is, it's already loaded in the JavaScript, and those as a breakpoint. So now, if I *Step Over* this it will actually create the Todo object.

If I go into the *Console* now, that object that we didn't have access to, we now have access to. Problem is, if I let this code run to completion, it will fall off the end here and I will lose access.

What I want to do is, whilst we still have access to it, I'm going to create a variable, myTodo and just assign todo to it, and let's have a quick look, myTodo.



There we go.

So, now if I run this to completion and go into the console, let me try and access the todo object. I don't have access to it.

Let me access myTodo object and I do have access to it.

**We've essentially hijacked the object in mid-loading which can make things a little bit easier, because all of these objects within the todo object have functionality.**

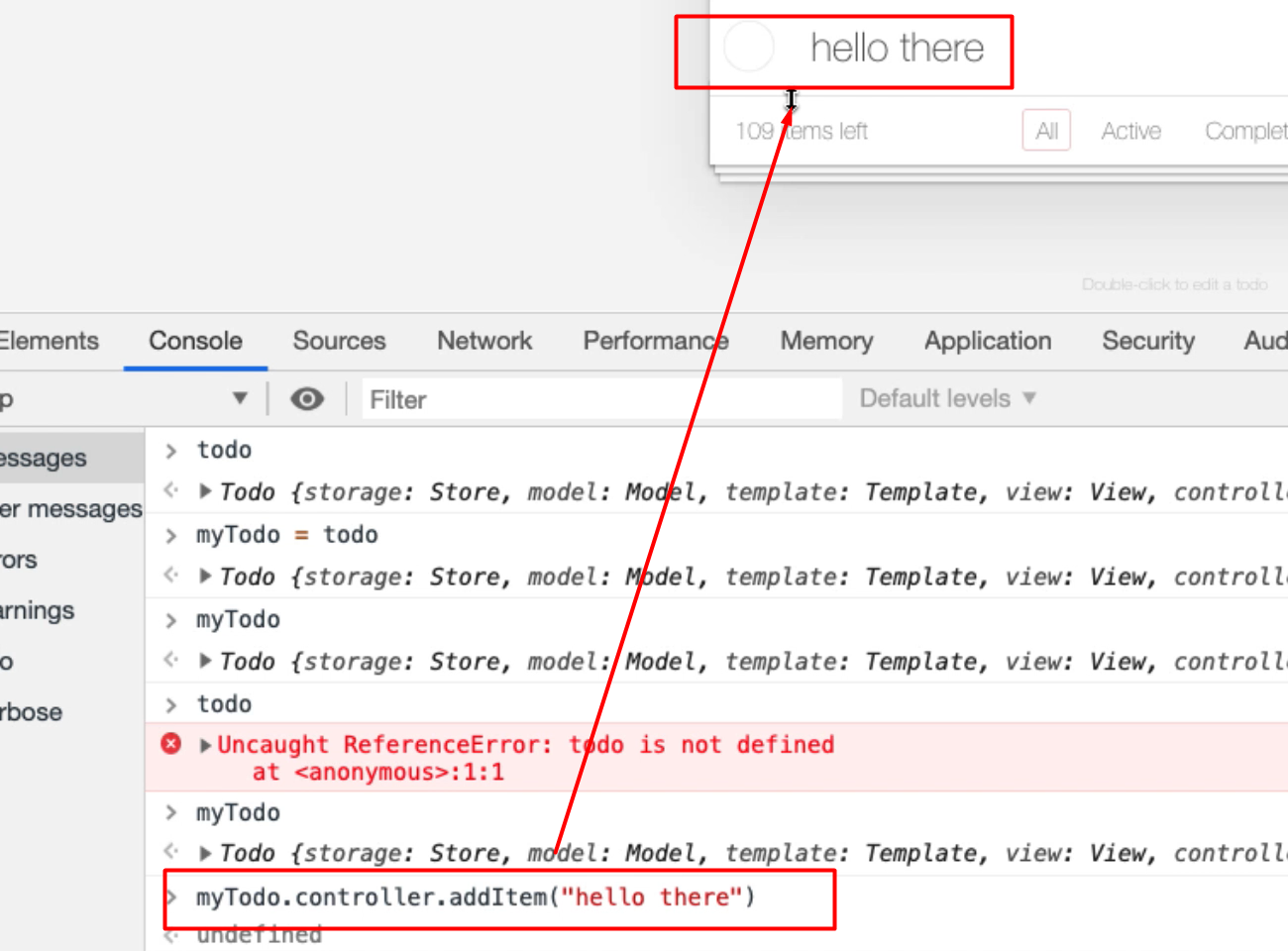
Let me have a quick look at the controller.

The controller has the ability to add an item. So, let me try that.

If I say

mytodo.controller.addItem("hello there")

There we go.



And this is perfectly valid, it's in the application.

We've triggered all of the events, and that's a lot simpler than writing all the code to trigger the events. But because of the way the application is written, we don't have access to that by default because it's in a private object.

This is the only way I know of hijacking this. Unfortunately, every time I refresh the page, I'll lose access to this, but it's a very simple enough step to manually interact with, and if I wanted to do things quickly in a very tactical fashion, this might be a very simple way of doing it.

**Now, we have to learn how to understand the actual JavaScript at this point because we're no longer triggering the external events.**

When we were doing it previously, all we had to understand was: how does a browser work? How do events work?

Now we have to start understanding: how does the actual application work? How does this code work?

So, that would take us a little bit longer to learn and we'll have to study JavaScript on our own to do that. But this is a very useful technique, and if the application itself does expose the objects publicly, then you could write bookmarklets and use this in your code.

**You just have to understand that there's a risk because we're no longer triggering the GUI events, we're now working at the application itself.**

So, it may be useful for setting up lots of data, but it might not be the same as the user interacting with it. But this is a technique that I use a lot when I'm hacking JavaScript games and writing bots to execute those.

And it also demonstrates that nothing in the front end is safe or immune to manipulation by a user.

People very often use this pattern because they think, "Well, no one has access to it," but clearly, we can get access to it if we are determined.

**Quiz**

The quiz for this chapter can be found in section 12.3