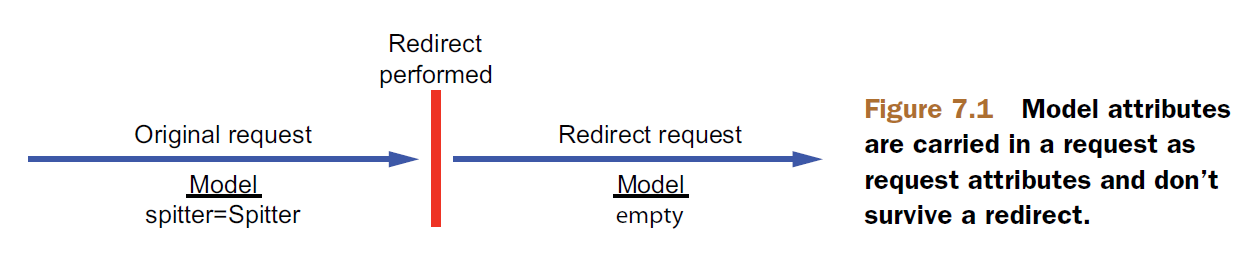
***Carrying data across redirect requests***

* As mentioned in section XREF \_writing\_a\_form\_handling\_controller, it’s generally a good practice to perform a redirect after handling a POST request. Among other things, this prevents the client from reissuing a dangerous POST request if the user clicks the Refresh or back-arrow button in their browser.
* The redirect: prefix in the view names returned from controller methods. When a controller method returns a String whose value starts with redirect:, that String isn’t used to look up a view, but is instead used as a path to redirect the browser to. Looking back at listing XREF ex\_SpitterController\_processRegistration\_validation, you’ll see that the last line of the processRegistration() method returns a redirect:String like this:



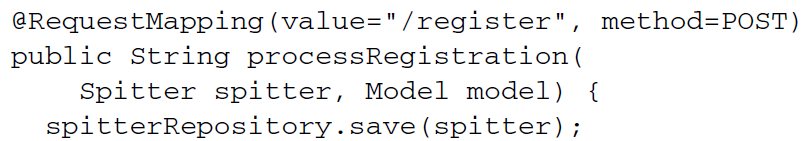
* The redirect: prefix makes working with redirects plain and simple. You’d think there’s nothing more that Spring could do to make working with redirects any simpler. Spring has a bit more to offer to help with redirects. Specifically, how can a redirecting method send data to the method that handles the redirect? Typically, when a handler method completes, any model data specified in the method is copied into the request as request attributes, and the request is forwarded to the view for rendering. Because it’s the same request that’s handled by both the controller method and the view, the request attributes survive the forward.

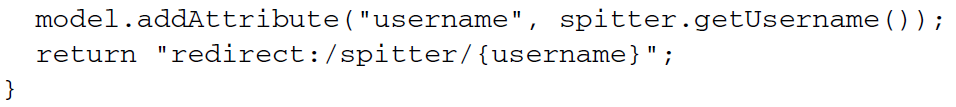


* As illustrated, when a controller method results in a redirect, the original request ends and a new HTTP GET request begins. Any model data carried in the original request dies with the request. The new request is devoid of any model data in its attributes and has to figure it out on its own.
* Clearly, the model isn’t going to help you carry data across a redirect. But there are a couple of options to get the data from the redirecting method to the redirect handling method:
* Passing data as path variables and/or query parameters using URL templates
* Sending data in flash attributes
* Passing data in path variables and query parameters seems simple enough. In listing XREF ex\_SpitterController\_processRegistration\_validation, for example, the newly created Spitter’s username is passed as a path variable. But as it’s currently written, the username value is concatenated to the redirect String. That works, but it’s far from bulletproof. String concatenation is dangerous business when constructing things like URLs and SQL queries.
* Instead of concatenating your way to a redirect URL, Spring offers the option of using templates to define redirect URLs. For example, the last line of processRegistration() in listing XREF ex\_SpitterController\_processRegistration\_validation could be written like this:

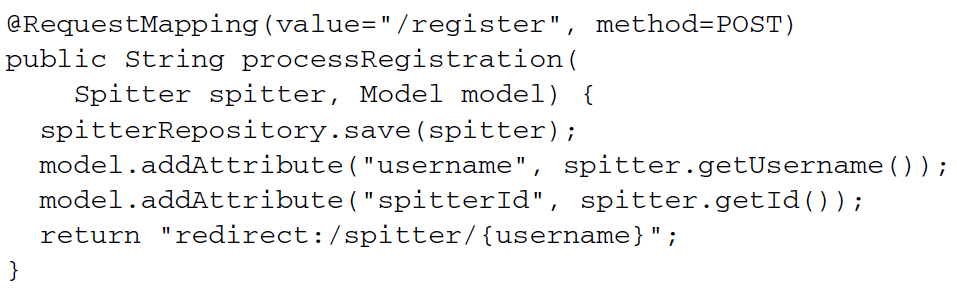


* All you need to do is set the value in the model. To do that, the processRegistration() needs to be written to accept a Model as a parameter and populate it with the username. Here’s how it can set the username value in the model so that it can fill in the placeholder in the redirect path:





* Because it’s filled into the placeholder in the URL template instead of concatenated into the redirect String, any unsafe characters in the username property are escaped. This is safer than allowing the user to type in whatever they want for the username and then appending it to the path.
* What’s more, any other primitive values in the model are also added to the redirect URL as query parameters. Suppose, for the sake of example, that in addition to the username, the model also contained the newly created Spitter object’s id property. The processRegistration() method could be written like this:

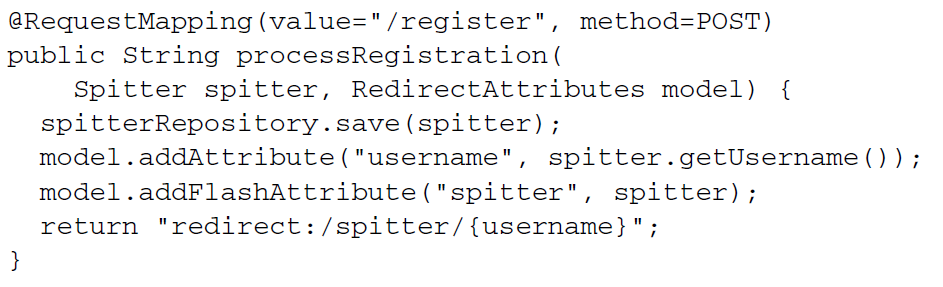


* Not much has changed with regard to the redirect String being returned. But because the spitterId attribute from the model doesn’t map to any URL placeholders in the redirect, it’s tacked on to the redirect automatically as a query parameter. If the username attribute is habuma and the spitterId attribute is 42, then the resulting redirect path will be /spitter/habuma?spitterId=42.
* Sending data across a redirect via path variables and query parameters is easy and straightforward, but it’s also somewhat limiting. It’s only good for sending simple values, such as String and numeric values. There’s no good way to send anything more complex in a URL. But that’s where flash attributes come in to help.

***Working with flash attributes***

Let’s say that instead of sending a username or ID in the redirect, you want to send the actual Spitter object. If you send just the ID, then the method that handles the redirect has to turn around and look up the Spitter from the database. But before the redirect, you already have the Spitter object in hand. Why not send it to the redirect handling method to display?

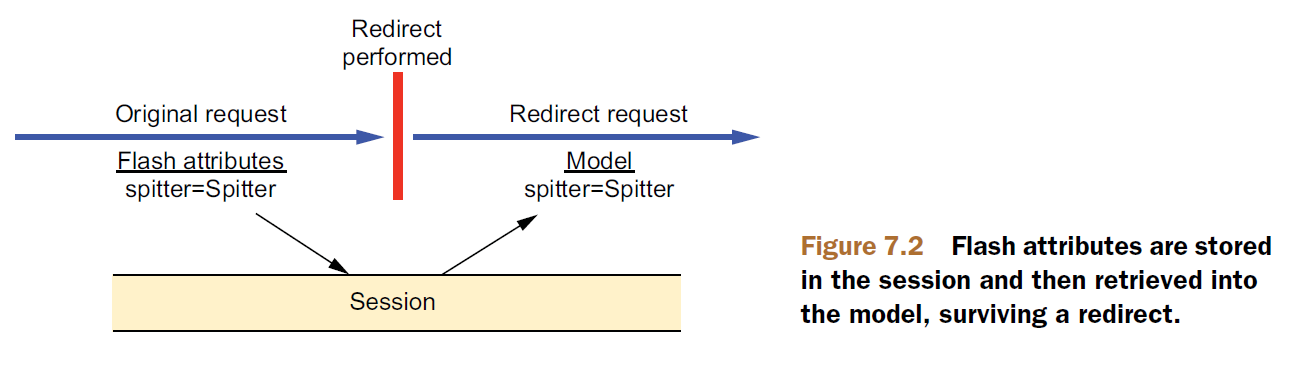
* A Spitter object is a bit more complex than a String or an int. Therefore, it can’t easily be sent as a path variable or a query parameter. It can, however, be set as an attribute in the model.
* But as we’ve already discussed, model attributes are ultimately copied into the request as request attributes and are lost when the redirect takes place. Therefore, you need to put the Spitter object somewhere that will survive the redirect.
* One option is to put the Spitter into the session. A session is long-lasting, spanning multiple requests. So you could put the Spitter into the session before the redirect and then retrieve it from the session after the redirect. Of course, you’re also responsible for cleaning it up from the session after the redirect.
* As it turns out, Spring agrees that putting data into the session is a great way to pass information that survives a redirect. But Spring doesn’t think you should be responsible for managing that data. Instead, Spring offers the capability of sending the data as *flash attributes*. Flash attributes, by definition, carry data until the next request; then they go away.
* Spring offers a way to set flash attributes via RedirectAttributes, a sub-interface of Model added in Spring 3.1. RedirectAttributes offers everything that Model offers, plus a few methods for setting flash attributes.
* Specifically, RedirectAttributes provides a couple of addFlashAttribute() methods for adding a flash attribute. Revisiting the processRegistration() method once more, you can use addFlashAttribute() to add the Spitter object to the model:



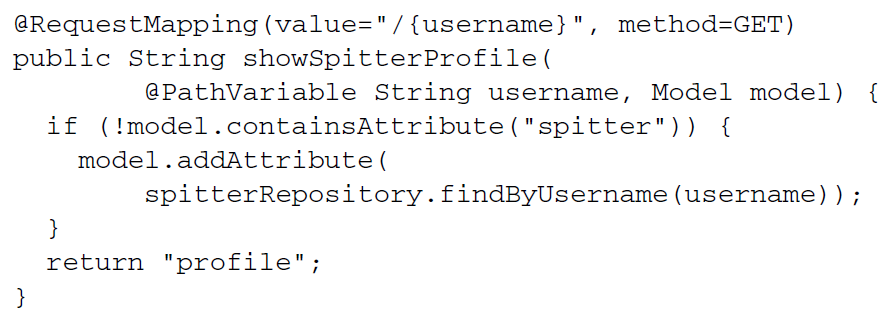
* Here, you’re calling addFlashAttribute(), giving it spitter as the key and the Spitter object as a value. Optionally, you can leave the key parameter out and let the key be inferred from the value type:



* Because you’re passing a Spitter object to addFlashAttribute(), the key is inferred to be spitter.
* Before the redirect takes place, all flash attributes are copied into the session. After the redirect, the flash attributes stored in the session are moved out of the session and into the model. The method that handles the redirect request can then access the Spitter from the model, just like any other model object. Figure illustrates how this works.



* To complete the flash attribute story, here’s a slightly updated version of showSpitterProfile() that checks for a Spitter in the model before going to the trouble of looking it up from the database:



* As you can see, the first thing showSpitterProfile() does is check to see if there’s a model attribute whose key is spitter. If the model contains a spitter attribute, then there’s nothing to do. The Spitter object contained therein will be carried forward to the view for rendering. But if the model doesn’t contain a spitter attribute, then showSpitterProfile() will look up the Spitter from the repository and store it in the model.