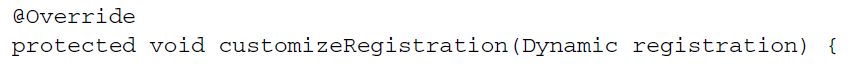
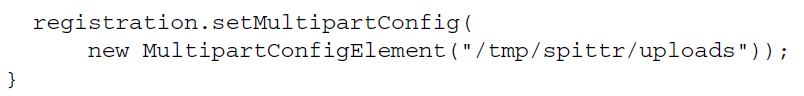
***Advanced Spring MVC***

We’ll look at how to write controllers that accept file uploads, how to handle exceptions thrown from controllers, and how to pass data around in the model such that it survives a redirect.

***Customizing DispatcherServlet configuration:***

* There are more methods that can be overridden to apply additional configuration. One such method is customizeRegistration(). After AbstractAnnotationConfigDispatcherServletInitializer registers DispatcherServlet with the servlet container, it calls the customizeRegistration() method, passing in the ServletRegistration.Dynamic that resulted from the servlet registration. By overriding customizeRegistration(), you can apply additional configuration to DispatcherServlet.
* For Instance, to handle multipart requests and file uploads with Spring MVC. If you plan to use Servlet 3.0 support for multipart configuration, you need to enable DispatcherServlet’s registration to enable multipart requests. You can override the customizeRegistration()method to set a MultipartConfigElement like this:





* With the ServletRegistration.Dynamic that’s given to customizeRegistration(),

you can do several things, including set the load-on-startup priority by calling set-

LoadOnStartup(), set an initialization parameter by calling setInitParameter(), and

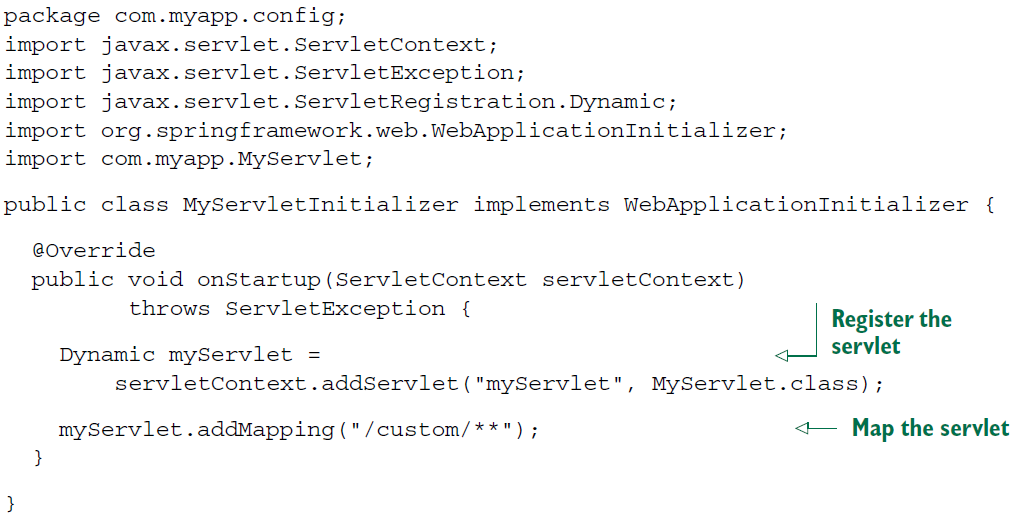
call setMultipartConfig()to configure Servlet 3.0 multipart support.

* In the preceding example, you’re setting up multipart support to temporarily store uploaded files

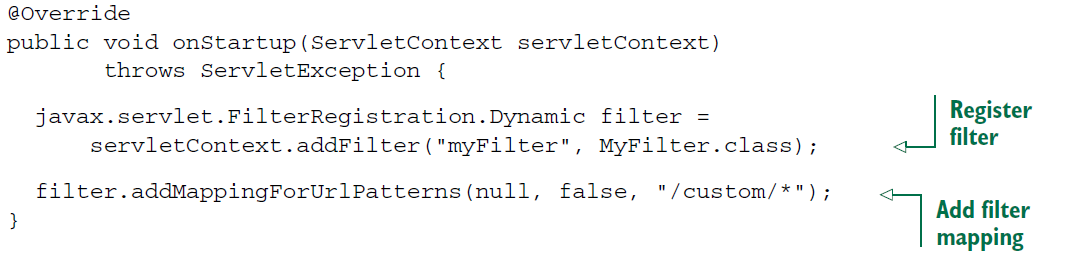
at /tmp/spittr/uploads.

***Adding additional servlets and filters***

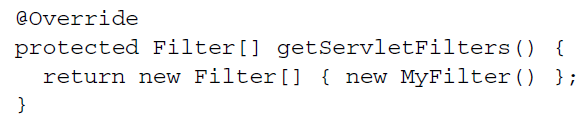
* Given the way that AbstractAnnotationConfigDispatcherServletInitializer is defined, it will create a DispatcherServlet and a ContextLoaderListener. But what if you want to register additional servlets, filters, or listeners?
* One of the nice things about working with a Java-based initializer is that (unlike with web.xml) you can define as many initializer classes as you want. Therefore, if you need to register any additional components into the web container, you need only create a new initializer class. The easiest way to do this is by implementing Spring’s WebApplicationInitializer interface.
* For example, the following listing shows how to create an implementation of WebApplicationInitializer that registers a servlet.



* + It is a rather basic servlet-registering initializer class. It registers a servlet and maps it to a single path. You could use this approach to register DispatcherServlet manually. (But there’s no need, because AbstractAnnotationConfigDispatcherServletInitializer does a fine job without as much code.)
* Similarly, you can register listeners and filters by creating a new implementation of WebApplicationInitializer. For example, the next listing shows how to register a filter.



* WebApplicationInitializer is a fine general-purpose way of registering servlets, filters, and listeners in Java when deploying to a Servlet 3.0 container. But if you’re registering a filter and only need to map that filter to DispatcherServlet, then there’s a shortcut in AbstractAnnotationConfigDispatcherServletInitializer.
* To register one or more filters and map them to DispatcherServlet, all you need to do is override the getServletFilters() method of AbstractAnnotationConfigDispatcherServletInitializer. For example, the following getServletFilters() method overrides the one from AbstractAnnotationConfigDispatcherServletInitializer to register a filter:



* + As you can see, this method returns an array of javax.servlet.Filter. Here it only

returns a single filter, but it could return as many filters as you need. There’s no need

to declare the mapping for the filters; any filter returned from getServletFilters()

will automatically be mapped to DispatcherServlet.

* When deploying to a Servlet 3.0 container, Spring offers several ways of registering servlets (including DispatcherServlet), filters, and listeners *without* creating a web.xml file. But you don’t have to use any of those if you don’t want to. If you aren’t deploying your application to a Servlet 3.0 container (or if you just like working with web.xml), then there’s no reason you can’t configure Spring MVC in a legacy manner with web.xml.