## Se viene usato un file di configurazione Xml 21.15 Code-based Servlet container initialization

Servlet 3.0 config with annotations:

In a Servlet 3.0+ environment, you have the option of configuring the Servlet container programmatically as an alternative or in combination with a web.xml file.   
Attraverso ServletContainerInitializer

Gerarchia classi Spring MVC:

1) ServletContainerInitializer (Servlet-api-3.0+)

2) SpringServletContainerInitializer (Spring MVC)

3) WebApplicationInitializer (Spring MVC)

Below is an example of registering a DispatcherServlet:

**import** org.springframework.web.WebApplicationInitializer;

**public** **class** MyWebApplicationInitializer **implements** WebApplicationInitializer {

*@Override*

**public** **void** onStartup(ServletContext container) {

XmlWebApplicationContext appContext = **new** XmlWebApplicationContext();

appContext.setConfigLocation("/WEB-INF/spring/dispatcher-config.xml");

ServletRegistration.Dynamic registration = container.addServlet("dispatcher", **new** DispatcherServlet(appContext));

registration.setLoadOnStartup(1);

registration.addMapping("/");

}

}

WebApplicationInitializer is an interface provided by Spring MVC that ensures your implementation is detected and automatically used to initialize any Servlet 3 container.

PIU SPECIALIZZATE:

A seconda che la configurazione sia gestita con una classe di SMVC Context con annotations o con un file di configurazione xml si hanno ulteriori specializzazioni:

1. **File di configurazione xml:**

An abstract base class of WebApplicationInitializer    
named AbstractDispatcherServletInitializer makes it even easier to register the DispatcherServlet by simply overriding methods to specify the servlet mapping and the location of the DispatcherServlet.

**public** **class** MyWebAppInitializer **extends** AbstractDispatcherServletInitializer {

*@Override*

**protected** WebApplicationContext createRootApplicationContext() {

**return** null;

}

*@Override*

**protected** WebApplicationContext createServletApplicationContext() {

XmlWebApplicationContext cxt = **new** XmlWebApplicationContext();

**cxt.setConfigLocation("/WEB-INF/spring/dispatcher-config.xml");**

**return** cxt;

}

*@Override*

**protected** String[] getServletMappings() {

**return** **new** String[] { "/" };

}

}

AbstractDispatcherServletInitializer also provides a convenient way to add Filter instances and have them automatically mapped to theDispatcherServlet:

**public** **class** MyWebAppInitializer **extends** AbstractDispatcherServletInitializer {

*// ...*

*@Override*

**protected** Filter[] getServletFilters() {

**return** **new** Filter[] { **new** HiddenHttpMethodFilter(), **new** CharacterEncodingFilter() };

}

}

Each filter is added with a default name based on its concrete type and automatically mapped to the DispatcherServlet.

1. **Initializer con collegamento a context di @Configuration**

**public** **class** MyWebAppInitializer **extends** AbstractAnnotationConfigDispatcherServletInitializer {

*@Override*

**protected** Class<?>[] getRootConfigClasses() {

**return** null;

}

*@Override*

**protected** Class<?>[] getServletConfigClasses() {

**return** **new** Class[] { MyWebConfig.**class** };

}

*@Override*

**protected** String[] getServletMappings() {

**return** **new** String[] { "/" };

}

}

The above example is for an application that uses Java-based Spring configuration

The isAsyncSupported protected method of AbstractDispatcherServletInitializer provides a single place to enable async support on theDispatcherServlet and all filters mapped to it. By default this flag is set to true.

Finally, if you need to further customize the DispatcherServlet itself, you can override the createDispatcherServlet method.

Codice da Pro Spring MVC:

***Listing 5-23.*** *An Example init-binder Method*

package com.apress.prospringmvc.bookstore.web.controller;

//Imports omitted

@Controller

@RequestMapping(value = "/customer")

public class RegistrationController {

// Other methods omitted

@InitBinder

public void initBinder(WebDataBinder binder) {

binder.initDirectFieldAccess();

binder.setDisallowedFields("id");

binder.setRequiredFields("username", "password", "emailAddress");

}

}