Why and How to use SpringBootServletInitializer?

Amit

February 19, 2023

Spring Boot

Spring Boot

Spring Boot provides SpringBootServletInitializer that runs a Spring Application from a war deployment. It binds *Servlet*, *Filter* and *ServletContextInitializer* beans from the spring application context to the server.

Note that a *WebApplicationInitializer* is only needed if we build a war file and deploy it. We won't need this if we prefer to run an embedded web server.

1. Bootstrapping of a Spring Boot Application

A typical Spring Boot application uses the *main()* method as the application's entry point when it is run in the embedded server. Inside the *main()* method, *SpringApplication.run()* scans and configures necessary beans for running the web application.

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
```

such as Apache Tomcat or JBoss, the entry point is not the *main()* method. Instead, we would need to configure the *ServletContext*, this is where an initializer comes into play.

There are two ways in which we can configure the *ServletContext* in a web application:

- Using *web.xml*
- Using Java Configuration

We will skip the web.xml configurations for the scope of this article and focus on programmatic configuration.

2. Servlet Context Initialization

Let us understand how *SpringBootServletInitializer* comes into the picture during the web application startup phase. For this, we need to understand the following interfaces and classes phase and perform any required programmatic registration of servlets, filters, and listeners in response to it. This interface has an *onStartup(ServletContext)* method, which is invoked at the start of the application represented by the given *ServletContext*.

- (C) ***SpringServletContainerInitializer***: The *org.springframework.web.SpringServletContainerInitializer* is a Spring Framework provided implementation of the *ServletContainerInitializer* and is responsible for instantiating and delegating the *ServletContext* to any user-defined *WebApplicationInitializer* implementations. In general, this class should be viewed as supporting infrastructure only.

```
public class SpringServletContainerInitializer extends Object implements Se

     onStartup(Set<Class<?>> webAppInitializerClasses, ServletContext servl
}
```

- (I) ***WebApplicationInitializer***: The *org.springframework.web.WebApplicationInitializer* interface provides code-based *ServletContext* configuration and does the actual work of initializing the *ServletContext*.

- (C) ***SpringBootServletInitializer***: The *org.springframework.boot.web.servlet.support.SpringBootServletInitializer* is an *abstract* class that implements the *WebApplicationInitializer*. It provides a way to run a Spring Boot application from a traditional WAR deployment.

```
public abstract class SpringBootServletInitializer extends Object implement

    configure(SpringApplicationBuilder builder);
    //...
```

value of *WebApplicationInitializer*. This means that the Servlet container will scan for classes implementing the *WebApplicationInitializer* interface and call the *onStartUp()* method in these classes.

A typical Spring web application initialization flow is as below:

- *SpringServletContainerInitializer* is scanned, loaded and instantiated by the server (servlet container) and its *onStartup()* is invoked by the Servlet container.
- The *onStartup()* will delegate the *ServletContext* to the classes implementing *WebApplicationInitializer* interface in the application classpath.
- *SpringBootServletInitializer* class implements *WebApplicationInitializer* interface. It binds Servlet, Filter and other necessary beans from the application context to the server.

3. How to Use *SpringBootServletInitializer*?

By default, Spring Boot applications use the embedded Tomcat server. To deploy a Spring Boot application using the traditional war deployment, we extend the *SpringBootServletInitializer* class and override its *configure()* method:

```
@SpringBootApplication
public class MyApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder a
        return application.sources(MyApplication.class);
    }
}
```

Spring Boot projects using Spring Initializr.

By default, Spring boot applications are packaged as *jar*. We need to package the application as a *'war'* file to deploy it as a web application in external servers.

```
<packaging>war</packaging>
```

The *WebApplicationInitializer* is needed to bootstrap the *ServletContext* programmatically as opposed to the traditional `web.xml`-based approach. Let's extend the *SpringBootServletInitializer* in our application class and override the configure method.

We can, optionally, retain the *main()* method so the application can be tested in the embedded server during the development phase.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitia

@SpringBootApplication
public class MyApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder a
        return application.sources(MyApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

```
@RestController
public class GreetingController {

    @GetMapping("/greet")
    public String greet() {
        return "Hey there!";
    }
}
```

To create the WAR archive, we need to build the application. Go to the root folder of the application and run the following command:

```
$ mvn clean package
```

This will create the WAR file at the location */target* folder. We can now copy this WAR in the external tomcat's webapps/ folder and start the tomcat server, which would deploy the WAR, and we can test the API.

5. Conclusion

In this article, we looked at the internal workings of *SpringBootServletInitializer* and how we can use it to deploy a Spring Boot application to an external tomcat server.

Happy Learning!!!

DISCOVER MORE

Related Articles And Resources

Why Java Iterator throws ConcurrentModificationException?

Spring Boot SOAP Client – WebServiceTemplate Example

Spring Boot Dev Tools Tutorial

Comments

Subscribe

Be the First to Comment!

B I U S {} [+]

0 COMMENTS

Search …

About Us

HowToDoInJava provides tutorials and how-to guides on Java and related technologies.

It also shares the best practices, algorithms & solutions and frequently asked interview questions.

Tutorial Series

- OOP
- Regex
- Maven
- Logging
- TypeScript
- Python

Meta Links

- About Us
- Advertise
- Contact Us
- Privacy Policy

Our Blogs

REST API Tutorial