**What’s new in Spring 5?**

1. **Baseline upgrades:** To build and run Spring 5 application, you will need minimum [JDK 8](https://howtodoinjava.com/java-8-tutorial/) and Java EE 7. Previous JDK and Java EE versions are not supported anymore. To elaborate, Java EE 7 includes –

* Servlet 3.1
* [JMS 2.0](https://howtodoinjava.com/spring/spring-boot/spring-boot-jmstemplate-activemq/)
* [JPA 2.1](https://howtodoinjava.com/jpa-tutorials-and-examples/)
* [JAX-RS 2.0](https://restfulapi.net/create-rest-apis-with-jax-rs-2-0/)
* [Bean Validation 1.1](https://howtodoinjava.com/spring/spring-mvc/spring-bean-validation-example-with-jsr-303-annotations/)

Similar to Java baseline, there are changes in baselines of many other frameworks as well. e.g.

* Hibernate 5
* Jackson 2.6
* EhCache 2.10
* [JUnit 5](https://howtodoinjava.com/junit-5-tutorial/)
* Tiles 3

Also, note down the minimum supported versions of various servers.

* Tomcat 8.5+
* Jetty 9.4+
* WildFly 10+
* Netty 4.1+
* Undertow 1.4+

## JDK 9 runtime compatibility

## Reactive programming support : [Reactive programming](https://howtodoinjava.com/rxjava/rxjava-2-0-tutorial/) is one of the most important features of Spring Framework 5.0. Reactive programming provides an alternate style of programming focused on building applications that react to events.

## A functional web framework : Spring 5 also provides a functional web framework. It provides features to define endpoints using functional programming style. This framework introduces two fundamental components: HandlerFunction and RouterFunction.

The **HandlerFunction** represents a function that handles incoming requests and generates responses. **RouterFunction** serves as an alternative to the @RequestMapping annotation. It’s used for routing incoming requests to handler functions. e.g.

|  |
| --- |
| RouterFunction<String> route =      route(GET("/hello-world"),      request -> Response.ok().body(fromObject("Hello World"))); |

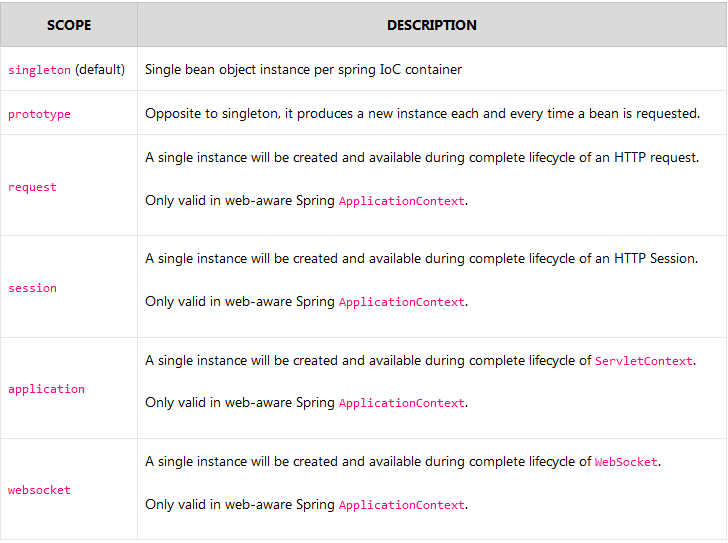
## Kotlin support : [Kotlin](https://kotlinlang.org/) is a statically typed JVM language that enables code that is expressive, short, and readable. Spring framework 5.0 has good support for Kotlin.

1. **Dropped features:** Along with the increase in baseline versions for Java, Java EE and a few other frameworks, Spring Framework 5 removed support for a few frameworks. e.g:

* Portlet
* Velocity
* JasperReports
* XMLBeans
* JDO
* Guava

**Spring Bean Scope :**  In Spring framework, we can create beans in 6 inbuilt **spring bean scopes** and you can also define your **custom bean scope** as well. Out of these six scopes, four are available only if you use a web-aware ApplicationContext. singleton and prototype scopes are available in any type of IOC containers.

* In Spring, scope can be defined using spring bean **@Scope** annotation.



* As a rule, you should prefer to use the prototype scope for all stateful beans and the singleton scope for stateless beans

#### prototype scope

prototype scope results in the creation of a new bean instance every time a request for the bean is made by application code.

You should know that destruction [bean lifecycle methods](https://howtodoinjava.com/spring/spring-core/spring-bean-life-cycle/) are not called prototype scoped beans, only initialization callback methods are called. So as developer, you are responsible for clean up prototype-scoped bean instances and any resource there hold.

**Note :** To use beans in the request, session, application and websocket scopes, you need to register the [RequestContextListener](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/context/request/RequestContextListener.html) or [RequestContextFilter](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/RequestContextFilter.html).

#### 1.3. request scope

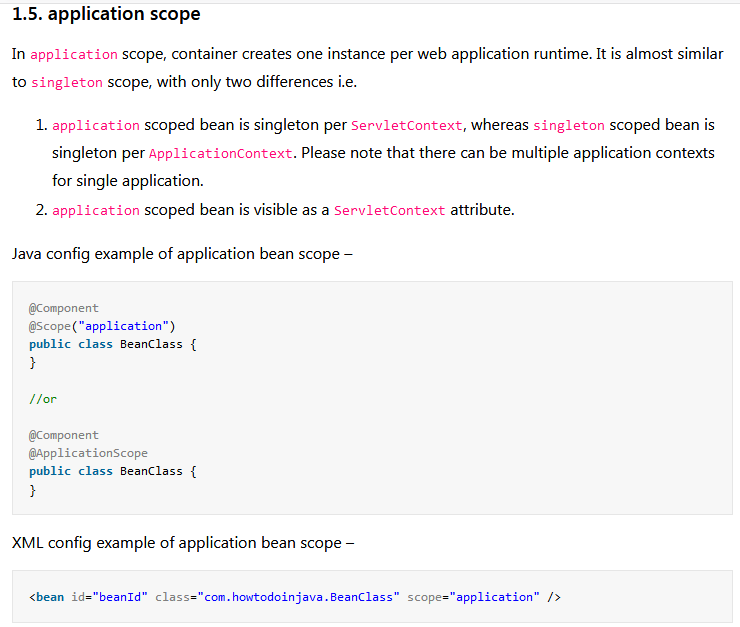
In request scope, container creates a new instance for each and every HTTP request. So, if server is currently handling 50 requests, then container can have at most 50 individual instances of bean class. Any state change to one instance, will not be visible to other instances. These instances are destructed as soon as the request is completed.

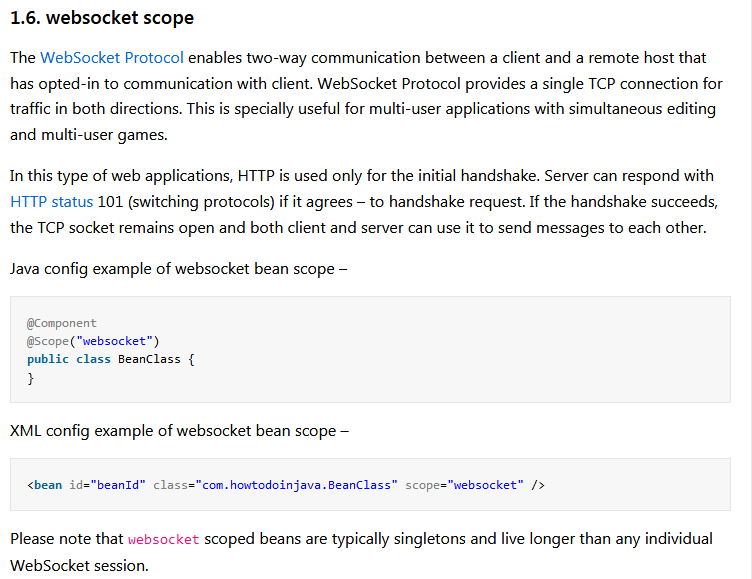


#### 1.4. session scope

In session scope, container creates a new instance for each and every HTTP session. So, if server has 20 active sessions, then container can have at most 20 individual instances of bean class. All HTTP requests within single session lifetime will have access to same single bean instance in that session scope.

Any state change to one instance, will not be visible to other instances. These instances are destructed as soon as the session is destroyed/end on server.





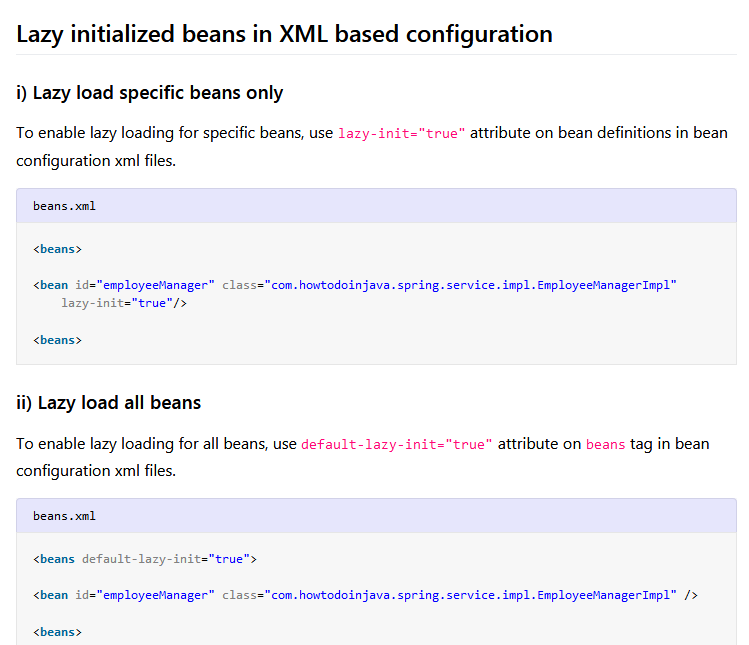
## 2. Custom thread scope



# Spring Bean Eager vs Lazy Initialization Configurations:

By default, Spring “application context” eagerly creates and initializes all ‘[singleton scoped](https://howtodoinjava.com/design-patterns/creational/singleton-design-pattern-in-java/)‘ beans during application startup itself. It helps in detecting the bean configuration issues at early stage, in most of the cases. But sometimes, you may need to mark some or all beans to be lazy initialized due to different project requirements.

Spring provides two easy ways to configure lazy initialization of beans based on which kind of configuration you are employing i.e. [XML based configuration](https://howtodoinjava.com/spring5/core/applicationcontext-xml-config-example/) or [java based configuration](https://howtodoinjava.com/spring5/core/spring-bean-container-java-configuration-example/).



## Lazy initialized beans in Java based configuration:

## 

## 

## 

## 

## @PostMapping vs @RequestMapping:

* **@PostMapping** annotation is one specialized version of @RequestMapping annotation which handle **HTTP POST** requests.

## @PostMapping acts as a shortcut for @RequestMapping(method = RequestMethod.POST).