# How to create beans?

Declaring a bean. To declare a bean, simply **annotate a method with the @Bean annotation**. When JavaConfig encounters such a method, it will execute that method and register the return value as a bean within a BeanFactory

## .1.  @Configuration

Annotating a class with the @Configuration annotation indicates that the class will be used by JavaConfig as a source of bean definitions.

An application may make use of just one @Configuration-annotated class, or many. @Configuration can be considered the equivalent of XML's <beans/> element. Like <beans/>, it provides an opportunity to explicitly set defaults for all enclosed bean definitions.

@Configuration(defaultAutowire = Autowire.BY\_TYPE, defaultLazy = Lazy.FALSE)

public class DataSourceConfiguration {

*// bean definitions follow*

}

Because the semantics of the attributes to the @Configuration annotation are 1:1 with the attributes to the <beans/> element, this documentation defers to the [beans-definition section](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-definition) of Chapter 3, IoC from the Core Spring documentation.

## 4.2.  @Bean

@Bean is a method-level annotation and a direct analog of the XML <bean/> element. The annotation supports most of the attributes offered by <bean/> such as [init-method](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-lifecycle-initializingbean), [destroy-method](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-lifecycle-disposablebean), [autowiring](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-autowire), [lazy-init](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-lazy-init), [dependency-check](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-dependencies), [depends-on](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-dependson) and [scope](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-scopes).

### 4.2.1. Declaring a bean

To declare a bean, simply annotate a method with the @Bean annotation. When JavaConfig encounters such a method, it will execute that method and register the return value as a bean within a BeanFactory. By default, the bean name will be that of the method name (see [bean naming](https://docs.spring.io/spring-javaconfig/docs/1.0.0.m3/reference/html/creating-bean-definitions.html#bean-naming) for details on how to customize this behavior).

@Configuration

public class AppConfig {

@Bean

public TransferService transferService() {

return new TransferServiceImpl();

}

}

The above is exactly equivalent to the following appConfig.xml:

<beans>

<bean name="transferService" class="com.acme.TransferServiceImpl"/>

</beans>

# How to over write the configuration?

To make a configuration in Spring Boot, you need to create a class and annotate it with @Configuration. Usually, in the configuration class, you can define a beans object. But if you want to override built-in configuration, you need to create a new class that extends the built-in class. This includes built-in configurations like security, templating, etc.

Overriding Built-In Configuration

By default, the URL path of the web service is case sensitive. For example:

http://localhost:8080/HelloWorld != http://localhost:8080/helloworld

So, we will make them equal to each other by overriding this configuration. To achieve the goal, we will create a new configuration Class that extends WebMvcConfigurationSupport. Then, we’re going to override configurePathMatch method, as shown below:

package com.hashimati.io.demo.overrideconfig.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.util.AntPathMatcher;

import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;

@Configuration

public class PathConfig extends WebMvcConfigurationSupport {

@Override

protected void configurePathMatch(PathMatchConfigurer configurer) {

configurer.setPathMatcher(new AntPathMatcher(){{

setCaseSensitive(false);

}}

);

}

}

# Advantage of ORM over JDBC ?

Hibernate is a framework that is used to develop persistence logic that is independent of Database software. In JDBC to develop persistence logic, we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic that is independent of database software.

### ORM (Object-Relational Mapping)

ORM, an abbreviation for Object-relational mapping, is a programming approach that connects object code to a relational database via a metadata descriptor. This object code is developed in object-oriented programming (OOP) languages like Java, Python, C++, C#, etc. ORM transforms data between type systems that don’t get along in relational databases or OOP languages. In other words, it is a technique for storing, recovering, updating, as well as deleting from an object-oriented program in a relational (table) database. Now let’s first understand the term object code.

**What is Object Code?**

Object code is defined as low-level code that is comprehensible by computers. It is generated by the compiler using the source code. In other words, it is a file generated by the compiler containing the instructions for the machine in the form of binary digits.

**Advantages of ORM:**

* Resolves object code and relational mismatch
* Using ORM, the development process is quite simplified as it automates object to table and table to object conversion which results in lower development and maintenance cost
* The code is less as compared to embedded SQL
* Gives an optimized solution that results in faster application and easier maintenance.

### JDBC (Java Database Connectivity)

JDBC is an abbreviation for Java database connectivity. It is a JavaSoft (was once a division of Sun Microsystems, responsible for developing Java and for licensing the use of JVM) specification of standard API that allows Java programs to access database management systems. The JDBC API is a set of Java interfaces and classes that allow you to connect to databases and send queries. This API of JDBC makes use of drivers of JDBC in order to get connected along with the database. There are four types of JDBC drivers:

* JDBC-ODBC bridge plus ODBC drive (Type 1 driver)
* Native -API, partly Java driver (Type 2 driver)
* JDBC-Net, pure Java driver (Type 3 driver)
* Native-protocol, pure Java driver (Type 4 driver)

**Advantages of JDBC:**

* It assists in quickly establishing a connection with a data source.
* It allows you to submit queries and update statements.
* It also allows you to collect data from the database and process the results in a very simple method.

We can also perform numerous activities using JDBC API required to handle the database:

* Connection to the database
* Execution of queries and updation of statements to the database
* Retrieving results that are fetched from the database

### Difference Between ORM and JDBC

ORM when compared to JDBC is easier to work with as it does all the work by itself. It maps Java classes to the database variables via XML. While working with domain-driven applications and in the case of complex object relationships, ORM is mostly preferred but when the application is simple enough then it is better to use JDBC.

| Object Relational Mapping | Java Database Connectivity |
| --- | --- |
| Little slower than JDBC | It is faster compared to ORM |
| SQL queries requirement is comparatively quite less however this doesn’t mean that you have to do less work using ORM | SQL queries are required here |
| Hibernate framework (working on ORM technology) makes it easy to store objects/data to database automatically without writing manual code | We have to write code manually to store objects/ data in the database |
| The flow from Object/data to hibernate i.e. the frontend part is based on the ORM technique | Whereas when the data is stored in the database finally i.e., the backend part is still based on JDBCin |
| There are not many restrictions while dealing with data. Even a single database cell can be retrieved, changed, and saved. | JDBC comes with a lot of restrictions on extracting the result-set, processing it, and then committing it back to the database. |

# How package scanning is done ? where the annotation is used?

<https://www.baeldung.com/spring-component-scanning>

With Spring, **we use the @ComponentScan annotation along with the @Configuration annotation to specify the packages that we want to be scanned**. @ComponentScan without arguments tells Spring to scan the current package and all of its sub-packages.

## @ComponentScan**With Arguments**

@ComponentScan(basePackages = "com.baeldung.componentscan.springapp.animals") @Configuration

The trick with Spring Boot is that many things happen implicitly. We use the @SpringBootApplication annotation, but it's a combination of three annotations:

@Configuration

@EnableAutoConfiguration

@ComponentScan

# How to load the xml file?

# <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

# <https://www.baeldung.com/spring-component-annotation>

<https://www.baeldung.com/spring-application-context>

# @Component @Repository

# @Service

# @Controller @Rest Controller – diiff

<https://www.geeksforgeeks.org/difference-between-controller-and-restcontroller-annotation-in-spring/#:~:text=%40Controller%20annotation%20indicates%20that%20the,assume%20%40ResponseBody%20semantics%20by%20default>.

| @Controller | @RestController |
| --- | --- |
| @Controller is used to mark classes as Spring MVC Controller. | @RestController annotation is a special controller used in RESTful Web services, and it’s the combination of @Controller and @ResponseBody annotation. |
| It is a specialized version of @Component annotation. | It is a specialized version of @Controller annotation. |
| In @Controller, we can return a view in Spring Web MVC. | In @RestController, we can not return a view. |
| @Controller annotation indicates that the class is a “controller” like a web controller. | @RestController annotation indicates that class is a controller where @RequestMapping methods assume @ResponseBody semantics by default. |
| In @Controller, we need to use @ResponseBody on every handler method. | In @RestController, we don’t need to use @ResponseBody on every handler method. |
| It was added to Spring 2.5 version. | It was added to Spring 4.0 version. |

# Security in spring boot

If a Spring Boot Security dependency is added on the classpath, Spring Boot application automatically requires the Basic Authentication for all HTTP Endpoints. The Endpoint “/” and “/home” does not require any authentication. All other Endpoints require authentication.

# Circuit breaker in spring boot

# CORS

<https://www.tutorialspoint.com/spring_boot/spring_boot_cors_support.htm#:~:text=Cross%2DOrigin%20Resource%20Sharing%20(CORS,the%20requests%20against%20different%20origin>.

Cross-Origin Resource Sharing (CORS) is a security concept that allows restricting the resources implemented in web browsers. It prevents the JavaScript code producing or consuming the requests against different origin.

For example, your web application is running on 8080 port and by using JavaScript you are trying to consuming RESTful web services from 9090 port. Under such situations, you will face the Cross-Origin Resource Sharing security issue on your web browsers.

Two requirements are needed to handle this issue −

* RESTful web services should support the Cross-Origin Resource Sharing.
* RESTful web service application should allow accessing the API(s) from the 8080 port.

In this chapter, we are going to learn in detail about How to Enable Cross-Origin Requests for a RESTful Web Service application.

## **Enable CORS in Controller Method**

We need to set the origins for RESTful web service by using **@CrossOrigin** annotation for the controller method. This @CrossOrigin annotation supports specific REST API, and not for the entire application.

@RequestMapping(value = "/products")

@CrossOrigin(origins = "http://localhost:8080")

public ResponseEntity<Object> getProduct() {

return null;

}

## **Global CORS Configuration**

We need to define the shown @Bean configuration to set the CORS configuration support globally to your Spring Boot application.

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/products").allowedOrigins("http://localhost:9000");

}

};

}

To code to set the CORS configuration globally in main Spring Boot application is given below.

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/products").allowedOrigins("http://localhost:8080");

}

};

}

}

Now, you can create a Spring Boot web application that runs on 8080 port and your RESTful web service application that can run on the 9090 port. For further details about implementation about RESTful Web Service, you can refer to the chapter titled **Consuming RESTful Web Services** of this tutorial.

# Diff ways of creating Object

# How to sort arrayList , HashMap

# Compare 2 diff objects in java

# How will you config db in springboot

# Acutator :

<https://www.callicoder.com/spring-boot-actuator/>

Spring Boot Actuator is **a sub-project of the Spring Boot Framework**. It uses HTTP endpoints to expose operational information about any running application. The main benefit of using this library is that we get health and monitoring metrics from production-ready applications.

Why do we use actuator in spring boot?

Spring Boot's 'Actuator' dependency is used **to monitor and manage the Spring web application**. We can use it to monitor and manage the application with the help of HTTP endpoints or with the JMX

How do I turn on actuators in spring boot?

To enable Spring Boot actuator endpoints to your Spring Boot application, we need to **add the Spring Boot Starter actuator dependency in our build configuration file**.

What are the actuator endpoints?

Actuator endpoints **let you monitor and interact with your application**. Spring Boot includes a number of built-in endpoints and lets you add your own. For example, the health endpoint provides basic application health information. Each individual endpoint can be enabled or disabled.

How do I enable and disable actuators?

By default, all the endpoints that I listed in the previous section are enabled except the shutdown endpoint.

You can enable or disable an actuator endpoint by setting the property management.endpoint.<id>.enabled to true or false (where id is the identifier for the endpoint).

For example, to enable the shutdown endpoint, add the following to your application.properties file -

management.endpoint.shutdown.enabled=true

# Collection Framework

# 8. Array List

# 9. Comparable and comparator

# 10. Throws vs Throw

# Dependency Injection

# 12. @Controller Annotation

# 13. @Autowired Annotation

# 14. @Id Annotation.

# 15. How will you configure DB connection in project?

# JSP: 16. Implicit object

# SQL/MYSQL: 17. What is query

# Subquery

# Constraints

# Where and Having clause

# Joins

# Type of left Joins