|  |  |
| --- | --- |
| JDBC | JPA |
| JDBC is database-dependent, which means that different scripts must be written for different databases. | JPA is database-agnostic, meaning that the same code can be used in a variety of databases with few (or no) modifications.  For example -> when we a findByUserame -> this will fire a select from query in case of Oracle or Mysql and it wil fire a json query in case of Mongo db , so from Developer perspective, his command remains the same. Where as in JDBC , the developer has to provide the DB specific SQL/NoSQL queries. |
| method invocations are transformed into pure SQL | repositories allow us to use derived methods |
|  |  |

Advantages of Springboot:

* Spring Boot works well with **several servlet containers** - tomcat is by default but can easily swap it for Jetty, Undertow, Resin, and Wildfly.
* If we have to use some other webserver , we can do it by excluding the tomcat dependency from spring-stater-web and then add explicit depenedency for spring-start-jetty or undertow.
* **Bootstrapping saves memory space 🡪** Spring Boot uses Boot Initializer to compile the source language. This bootstrapping technique makes it possible for users to save space on their devices and load applications quickly.

#### Decreased boilerplate code – getter and setter

#### No XML configuration required – annotations used

#### WAR files are not required – JAR has a shorter, simpler structure that makes them useful for developers and users

**IOC and DI: ????????**

In Spring Objects are loosely coupled i.e., each class is independent of each other so that everything can be tested individually. But when using those classes, a class may be dependent on other classes which need to be instantiated first.

So, we tell spring that class A is dependent on class B. So, when creating bean(like class) for class A, it instantiates class B prior to that of class A and injects that in class A using setter or constructor DI methods. I.e., we are telling spring the dependency at run-time. This is DI.

As, we are assigning the responsibility of creating objects(beans), maintaining them and their aggregations to Spring instead of hard-coding it, we call it Inversion Of Control(IOC).

**CIRCULAR DEPENDANCY:**

A circular dependency occurs when a bean A depends on another bean B, and the bean B depends on bean A as well: Spring cannot decide which of the beans should be created first since they depend on one another. In these cases, Spring will raise a *BeanCurrentlyInCreationException* while loading.

* @Lazy annotations in one of the beans
* Setter/field injections ???????? explain

**Interchange -->@controller @repository**

According to [documentation](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-stereotype-annotations) @Repository,@Service,@Controller are all synonyms. They are all annotated with @Compoenet. So, generally, they can be used one instead of other. But ... you should not do this.

First reason: any of these annotations *make clear the role* of your component in the application.

Second reason: some of these annotations processed differently by different Spring modules.

Annotations Rest API:

@GET@POST@PUT@DELETE@PATCH

HTTP codes:

**1XX — Informational codes**: The server acknowledges and is processing the request.

This category is informational, temporary, and invisible to the client.

**3XX — Redirection codes**: The server received the request, but there’s a redirect to somewhere else

**2XX — Success codes**: The server successfully received and processed the request.

**4XX — Client error codes**: The server couldn’t find (or reach) the page or website. This is an error on the site’s side.

**5XX — Server error codes:** The client made a valid request, but the server failed to complete the request.

**201 — Created**: The request succeeded and a new resource was created. This is usually the response after POST or PUT requests.

**204 — No Content**: The request was successfully processed, but there is no content. The headers may be useful.

**401 — Unauthorized**: The user doesn’t have valid authentication credentials to get the requested resource.

**403 — Forbidden**: The client doesn’t have access rights to the content. For example, it may require a password. Unlike the 401 HTTP error code, the server does know the client’s identity.

**404 — Not Found**: The server can’t find the requested resource, and no redirection has been set. [404 errors](https://www.semrush.com/blog/what-does-error-404-not-found-mean/) can harm your SEO efforts.

**500** **— Internal Server Error**: The server has encountered an unexpected error and cannot complete the request.

**502 — Bad Gateway**: The server acts as a gateway and gets an invalid response from an inbound host.

**503 — Service Unavailable**: The server is unable to process the request. This often occurs when a server is overloaded or down for maintenance.

**504 — Gateway Timeout**: The server was acting as a gateway or proxy and timed out, waiting for a response.

**what are different bean scopes in spring ?**

**singleton** 🡪this will create only one instance of the bean of the classes annotated with Component. It’s a default behaviour

**Prototype** 🡪 this will create multiple instance of the bean of the classes annotated with Component.

We can do it by annotating with @Scope(“Prototype”)

Request,session,global session – webAware spring app.

**Singleton Class:**