Bean factory lifeCycle:https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/BeanFactory.html

\_\_\_\_\_\_\_\_\_\_

Bean factory implementations should support the standard bean lifecycle interfaces as far as possible. The full set of initialization methods and their standard order is:

1. BeanNameAware's setBeanName
2. BeanClassLoaderAware's setBeanClassLoader
3. BeanFactoryAware's setBeanFactory
4. EnvironmentAware's setEnvironment
5. EmbeddedValueResolverAware's setEmbeddedValueResolver
6. ResourceLoaderAware's setResourceLoader (only applicable when running in an application context)
7. ApplicationEventPublisherAware's setApplicationEventPublisher (only applicable when running in an application context)
8. MessageSourceAware's setMessageSource (only applicable when running in an application context)
9. ApplicationContextAware's setApplicationContext (only applicable when running in an application context)
10. ServletContextAware's setServletContext (only applicable when running in a web application context)
11. postProcessBeforeInitialization methods of BeanPostProcessors
12. InitializingBean's afterPropertiesSet
13. a custom init-method definition
14. postProcessAfterInitialization methods of BeanPostProcessors

On shutdown of a bean factory, the following lifecycle methods apply:

* postProcessBeforeDestruction methods of DestructionAwareBeanPostProcessors
* DisposableBean's destroy
* a custom destroy-method definition

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

To read input from external source in beans($(jdbc.username) here $ specifies it is from external source):--< [Container Extension Points :: Spring Framework](https://docs.spring.io/spring-framework/reference/core/beans/factory-extension.html#beans-factory-extension-bpp-examples-aabpp)

Example:

<bean class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">

<property name="locations" value="classpath:com/something/jdbc.properties"/>

</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="${jdbc.driverClassName}"/>

<property name="url" value="${jdbc.url}"/>

<property name="username" value="${jdbc.username}"/>

<property name="password" value="${jdbc.password}"/>

</bean>

22/07/24

The annotations:

1. @PostConstruct
2. @PreDestroy
3. @Autowired
4. @Component
5. @ComponentScan
6. @Configuration-javaconfiguration
7. @Bean
8. @Primary
9. @Qualifier
10. @Service
11. @Value
12. @Repository
13. @Controller

Servlets and JSP’s🡪we should learn parallelly.

* Beans are loaded lazily and application context is loaded eagerly.
* JDBCUtil🡪JDBCDAO🡪sService🡪(Appconfig and Main class).

1. @Lookup-CGLIB will do the implementation of that function. The body will be left blank.

@Component

Public class NotificationService {

@Lookup

Public Appnotification getNotification() {

//Return new AppNotification():

}

}

1. @Profile- DEV TEST PROD
2. @Import-Jpaconfig.class,SchedulerConfig.class
3. @ImportResource(“spring-context.xml”)

RESTAPI🡪[Learn REST API Design - REST API Tutorial](https://restapitutorial.com/)

HTTP is stateless, doesn’t support state management.

HTTP METHODS:

1. GET
2. HEAD
3. OPTIONS
4. POST
5. PUT
6. TRACE
7. CONNECT
8. PATCH
9. DELETE

|  |  |
| --- | --- |
| **SOAP API** | **REST API** |
| Relies on SOAP (Simple Object Access Protocol) | Relies on REST (Representational State Transfer) architecture using HTTP. |
| Transports data in standard XML format. | Generally transports data in JSON. It is based on URI. Because REST follows stateless model, REST does not enforces message format as XML or JSON etc. |
| Because it is XML based and relies on SOAP, it works with WSDL | It works with GET, POST, PUT, DELETE |
| Works over HTTP, HTTPS, SMTP, XMPP | Works over HTTP and HTTPS |
| Highly structured/typed | Less structured -> less bulky data |
| Designed with large enterprise applications in mind | Designed with mobile devices in mind |

Rest🡪 app to app

Spring-Configuration files+(Autoconfiguration+Starters+Accurate+Actutors+Devtools+CloudTools+CLI)=SpringTools



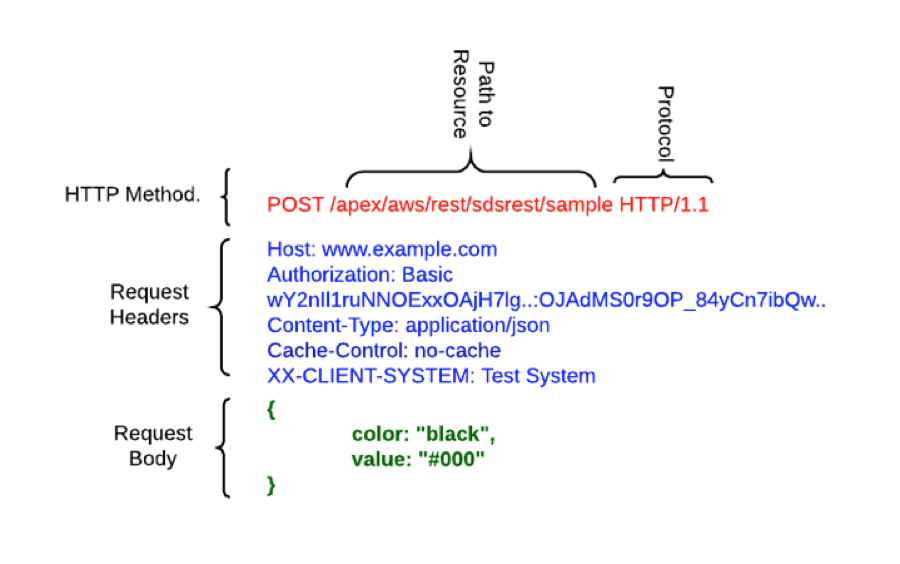
Spring Boot=Ten years of Spring experience

Spring boot= Agile development + dependencies starters + inbuilt tomcat sever +In memory DB-H2db

Spring boot-Microservices🡪RESTful

Client🡪end user🡪browser(user-agent,proxy)🡪application and server communication, http request and request headers

[Java Software | Oracle](https://www.oracle.com/java/#java-card)

* @RestController-controller+response body-<body>, http headers other than GET you will have body, (pay load)
* @Controller🡪commit or render views using view technologies(HTML or JSP or Tiles OR Velocity…)
* We can send or write into response body using ResponseEntity class.
* @GetMapping-handler resource for HTTP GET request.
  + Input through get method:

1. query string(?name=value) is sent via URL.
2. Main URL followed by a forward slash (/)
3. @PathVariable to receive this input.

* @PostMapping-for POST
* @PutMapping-for PUT

[Swagger Documentation](https://swagger.io/docs/)

* @DeleteMapping-for DELETE

HttpHeader: [HttpHeaders (Spring Framework 6.1.11 API)](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpHeaders.html#AGE)

critical security risks to web applications🡪[OWASP Top Ten | OWASP Foundation](https://owasp.org/www-project-top-ten/)