Core Spring Framework annotations

* @Required
* @Autowired
* @Qualifier
* @Configuration
* @ComponentScan
* @Bean
* @Lazy
* @Value

@RefreshScope: How do you update properties in a Spring application without redeploying it ?  
  
RefreshScope annotation enables you to dynamically reload environment properties from external source.  
  
The RefreshScope annotated Bean re-initializes itself when '/refresh' endpoint of the application is invoked.  
Latest environment properties are loaded into it with this refresh.  
This feature is a part of spring cloud module.  
  
Some example of such environment properties are database connection details, connection pool size or a feature toggle that you want to update in a running java application.  
  
@RefreshScope  
public class EnvironmentProperties {  
     
  @Value("${[property.name](http://property.name/)}")  
  private String someEnvProperty;  
  
  //getter and setter  
}

Spring Transactional Annotations :

* @EnableTransactionManagement
* @Transactional
  + Isolation
  + Label
  + noRollBackFor
  + noRollBackForClassName
  + rollbackFor
  + rollbackForClassName
  + propagation
  + readOnly
  + timeout
  + timeoutString
  + transactionManager
  + value

StereoType Annotation

* @Component
* @Controller
* @Service
* @Repository

Spring boot Annotations

* @EnableAutoConfiguration
* @SpringBootApplication
* @ConditionalOnClass
* @ConditionalOnMissingClass
* @ConditionalOnBean
* @ConditionalOnMissingBean
* @ConditionalOnResource
* @ConditionalOnProperty
* @ConditionalExpression
* @Conditional

Spring MVC and REST Annotations

* @RequestAttribute
* @RequestBody
* @RequestParam
* @RequestPart
* @ResponseBody
* @RequestHeader
* @ResponseStatus
* @ControllerAdvice
* @RestControllerAdvice
* @SessionAttribute
* @SessionAttributes

Spring Cloud Annotations

* @EnableConfigServer
* @EnableEurekaServer
* @EnableDiscoveryClient
* @EnableCircuitBreaker
* @HystrixCommand

Cache-based Annotations

* @Cachable
* @CachePut
* @CacheEvict
* @CacheConfig

Task Execution and Scheduling

* @Scheduled
* @Async

Springframework testing Annotations

* @BootstrapWith
* @ContextConfiguration
* @WebAppConfiguration
* @Repeat
* @Commit
* @Rollback
* @BeforeTransaction
* @AfterTransaction
* @SpringBootTest
* @DataJpaTest
* @DataMongoTest
* @WebMVCTest
* @AutoConfigureMockMVC
* @MockBean

**@Configuration**

[@Configuration](https://www.java67.com/2018/05/difference-between-springbootapplication-vs-EnableAutoConfiguration-annotations-Spring-Boot.html) is a class-level annotation. It should be noted that this annotation is an annotation of the spring framework, not a spring-boot annotation. Still, an XML configuration file is more popular in the early days of the[spring framework](https://medium.com/javarevisited/top-10-free-courses-to-learn-spring-framework-for-java-developers-639db9348d25).

Therefore, no one knows about the @configuration. The [@configuration annotation](https://www.java67.com/2019/04/top-10-spring-mvc-and-rest-annotations-examples-java.html) is used to declare a class as a configuration class. It is generally used in combination with [@bean](https://www.java67.com/2021/10/pring-bean-example-what-does-bean-annotation-does.html). It is worth mentioning that the initialization priority of the default configuration class is higher than that of ordinary components.

Graphical user interface, application

Description automatically generated

**@Bean**

@Bean is a method-level annotation. When [spring boot](https://medium.com/javarevisited/10-free-spring-boot-tutorials-and-courses-for-java-developers-53dfe084587e) starts, it will call the method using the @bean declaration (only in the configuration class) and register the return value as a special Java Bean in the spring container @Bean can be used with @lazy and @order annotations. If @order is not used, the default initialization order is code order.

Graphical user interface, text, application

Description automatically generated

**@ConfigurationProperties**

In spring boot, when you need to obtain configuration data, in addition to the [@value annotation](https://javarevisited.blogspot.com/2021/10/difference-between-autowired-and.html) provided by spring, spring boot also provides the @ConfigurationProperties annotation.

Compared with @value, @ConfigurationProperties is more convenient and you can obtain configuration in batches.

As long as this annotation is added to the attribute class and the prefix is specified, the corresponding configuration file data will be automatically filled into the bean.

simulator:  
 executor:  
 coreThreadNumber: 4  
 maxThreadNumber: 50

Graphical user interface, text, application

Description automatically generated

**@ComponentScan**

@ComponentScan annotation can scan appropriate components (such as [@component,](https://www.java67.com/2022/06/difference-between-component-service-repository-in-spring.html) [@service](https://javarevisited.blogspot.com/2017/11/difference-between-component-service.html#axzz6ngd8ND25), [@controller](https://javarevisited.blogspot.com/2021/09/how-to-return-different-http-status-from-sprnig-mvc-controller.html)) according to the user configured path and register them in the spring container.

If you use the [@springbootApplication annotation](https://javarevisited.blogspot.com/2018/05/the-springbootapplication-annotation-example-java-spring-boot.html) to start the service, and the startup class is placed in the root package, you do not need to add any arguments, because @springbootApplication already contains the @componentScan annotation, and the default scanned root path is the path where the current class is located.

However, if some components of the third-party library want to register in the spring container, you need to use [@componentScan](https://javarevisited.blogspot.com/2022/03/how-autowiring-of-beans-works-in-spring.html).

Graphical user interface, text, application, email

Description automatically generated

**@ConditionalOnClass and @ConditionalOnMissClass**

[@ConditionaloOnClass](https://www.java67.com/2021/09/conditional-annotations-in-spring-example-tutorial.html) annotation can be declared on classes and methods. If @ConditionalOnClass annotation is used as follows.

Graphical user interface, text, application, email

Description automatically generated

ThenKafkaTemplate.classcan only be found in the classpath, this configuration class will be correctly scanned and registered to the spring container. The @ConditionalOnMissClass is just the opposite. Only if the defined class cannot be found in the [classpath](https://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html" \t "_blank), the configuration class will be built and registered.

**@ConditionalOnBean and @ConditionalOnMissingBean**

Similar to the above annotations of the conditional series, @ConditionalOnBean and @ConditionalOnMissingBean indicate that classes will be built and registered only when a specific bean exists or does not exist.

**@ConditionalOnProperty**

@ConditionalOnProperty is also an annotation of the conditional series. This annotation can determine the conditions for bean construction according to the user’s config.

This example shows that the class SftpPersister will be recognized and built by spring only when the value of key audit.persist.sinkissftpor the key is not defined.

Graphical user interface, text, application, email

Description automatically generated