To avoid confusion, we should understand, that **configuration definition** and **bean definition** are two *different things*. There are three ways to define configuration, available in Spring 4 by default:

- **xml-based** configuration, when you describe configuration in xml file;
- **java-based** configuration, when configuration is Java class, marked with specific annotations;
- **groovy-based** configuration, when configuration is file with Groovy code;

And there are two ways to add bean definitions into application:

- **configuration inside** bean definition, when you add beans manually by declaration right in configuration.

  In this case definition will be based on configuration type. For xml-config it will be `<bean/>` tag, for java-based config - method with `@Bean` annotation and `beans {...}` construction for Groovy.
- **annotation based** bean definition, when you mark bean classes with specific annotations (like `@Component`, `@Service`, `@Controller` etc). This type of config uses classpath scanning.

In this case you have to specify directive for scanning classpath. For xml-config it will be `<context:component-scan base-package="..."/>`, for java-config - `@ComponentScan` annotation, for Groovy `ctx.'component-scan'(...)` invocation.

As you see, you can use configurations and bean definitions in different combinations.

Note, that if you use xml based config, you can choose approach to drive dependency injection: manually in xml, or by using annotations (`@Autowire`, `@Required` etc). In late case you have to define `<context:annotation-config/>`. But do not confuse bean definition and dependency injection control.

The easiest way to understand this is to look into the long history of the framework how this was developed.

- XML based configuration - this was there from the the beginning - version 1 - see javadoc for ClassPathXmlApplicationContext. This was around march 2004, the time of J2EE 1.4, which had HUGE xml configuration and Spring was big simplification (XML as well, but simpler). This uses XML for everything, including specifying autowiring, or what dependencies go where directly (your ref="accoundDao" example).
- Annotation based configuration - in Spring 2.5 - this came as a reaction to Java EE 5, new anotations like @Autowired were introduced, there was still some context configuration in XML files - usually you would define which packages were to be scanned and rest of it was done automatically based on annotations - hence the name.

Java based configuration came with Spring 3, was improved in later versions. This is when AnnotationConfigApplicationContext and Configuration annotation were introduced - you could potentially drop XML entirely -> java based config. Although this became practical only later with version 4+, because of large number of xml helper tags for aop, jdbc etc.