Pivotal

Core Spring 5.0 Certification Study Guide

COMPLETED: JANUARY 2018

Table of Contents

Overview

This guide is designed to help you prepare for the **Core Spring 5.0** certification exams. Please be aware that it should *not* be used if you have attended a Core-Spring course that was using a *previous* version – they have their own dedicated certification guides at http://pivotal.io/academy. At this time, previous versions are:

- CoreSpringV4.3 based on Spring 4.3
- CoreSpringV4.2 based on Spring 4.2
- CoreSpringV4.1 based on Spring 4.1
- CoreSpringV4.0 based on Spring 4.0
- CoreSpringV3.2 based on Spring 3.2

The certification exam is based on the 4-day Core Spring training and the materials provided with it are the ideal source to use for preparation. Of course, as with any certification, the most valuable part, besides recognition, is the learning process. Hence we encourage you to take time to experiment and follow your curiosity when questions arise.

A 4-day course contains a lot of material. To help you focus your efforts and to know when you're ready, we've put together this guide. The guide contains a list of topics and a list of further resources. Topics are organized by subject area, where each topic contains a description of what you should make sure you know.

The list of topics can be used as a check-list. The training materials can be used as a point of reference and as a learning ground. The list of resources is where you can go further for getting answers. Everything in the exam is covered somewhere in the course notes.

One possible way to prepare is to do the following for a given training module:

1. Review the slides, making notes of questions
2. Work through the labs
3. Review the list of topics that matches to the module by subject area
4. Use the lab to experiment with anything you need to spend more time on
5. Use the provided list of resources to look for further answers
6. Reading (at least partially) the reference documentation
7. Memorize the "big pictures", tables, overviews, etc

Of course there are many more ways to organize your efforts. You can pair up with someone else planning to take the exam or review all presentations for a given subject area before going through the labs. Or maybe you have access to actual applications you can review to test your knowledge.

Please keep in mind that you are expected to have good working knowledge of all the topics listed. Most of the questions will be very general, however you will be asked a few advanced questions.

Logistics

When you are ready to test and validate your product knowledge, please visit pivotal.io/training/certification to *purchase* an industry-recognized Pivotal certification exam.

Pivotal partners with Exams Local to remotely proctor our exams. Our certification exams may be taken from a location of your choosing anywhere in the world provided you can meet the basic system and test environment requirements and have a valid form of photo identification.

For help with your exam purchase, exam registration process, credentials verification, or other questions related to our certification program, process, and procedures, please visit pivotal.io/training/faq/certification.

The Exam

The exam itself is a computer-based exam. The exam software first gives you some general instructions: how to navigate, how to mark a question, and so forth - please read it carefully.

Once you have agreed that you want to start, you have 90 minutes to answer 50 multiple-choice questions. You must answer 38 questions correctly (76%) in order to pass the exam.

Basic exam technique applies: read each question *carefully* and answer the question that was asked *not* what you *thought* was asked.

*In particular the questions refer to pure Spring Framework **without** Spring Boot, unless the question explicitly says Spring Boot is involved.*

Exam FAQ

1. Is there anything in the exam, which was not covered in the course?
   Mostly no. BUT please note that `@Secured` and the Spring Security JSP tag library may be referenced in the exam and are not in the main course notes.
2. Do I have to know class names and method signatures?
   No. We think that this is why you are using an IDE - for us it's much more important that you've understood the concepts rather than learning API and method signatures.
3. Do I have to write, complete or rearrange source code?
   No. The only thing you should be able to do is read a snippet of code and understand what it's doing. For example, we might show you a class implementing a Spring callback and you will then see a couple of related questions. We do not ask you questions on things an IDE can do for you, like checking if the code will compile.
4. Do I have to know any other APIs like *AspectJ* expression language in detail?
   No. Of course you should be able to read, understand and use AspectJ expression

language (pointcut expressions) wherever it is necessary to configure Spring AOP – but this is not an exam about AspectJ.

5. Are the advanced slides part of the exam?

   No. Only the content presented before each chapter lab slide will be on the exam. Any course content presented after the chapter lab will not be in the exam. No content from the optional chapters will be on the exam.

Topics

The following is a list of topics, each of which is likely to have questions on the exam.

Several of the bullet points below overlap, asking the same (or a related) question in a different way. The answer to more than one bullet point question may be the *same* as the answer to another question just before or after. *Don't let this confuse you*.

Container, Dependency, and IOC

- What is dependency injection and what are the advantages?
- What is a pattern? What is an anti-pattern. Is dependency injection a pattern?
- What is an interface and what are the advantages of making use of them in Java?
  - Why are they recommended for Spring beans
- What is meant by “application-context?
- What is the concept of a “container” and what is its lifecycle?
- How are you going to create a new instance of an *ApplicationContext*?
- Can you describe the lifecycle of a Spring Bean in an *ApplicationContext*?
- How are you going to create an *ApplicationContext* in an integration test test?
- What is the preferred way to close an application context? Does Spring Boot do this for you?

- Can you describe:
  - Dependency injection using Java configuration?
  - Dependency injection using annotations (`@Component`, `@Autowired`)?
  - Component scanning, Stereotypes and Meta-Annotations?
  - Scopes for Spring beans? What is the default scope?
- Are beans lazily or eagerly instantiated by default? How do you alter this behavior?
- What is a property source? How would you use `@PropertySource`?

- What is a `BeanFactoryPostProcessor` and what is it used for? When is it invoked?
  - Why would you define a static `@Bean` method?
  - What is a `ProperySourcesPlaceholderConfigurer` used for?
- What is a `BeanPostProcessor` and how is it different to a `BeanFactoryPostProcessor`? What do they do? When are they called?
  - What is an initialization method and how is it declared on a Spring bean?
  - What is a destroy method, how is it declared and when is it called?
    - Consider how you enable JSR-250 annotations like `@PostConstruct` and `@PreDestroy`? When/how will they get called?

    - How else can you define an initialization or destruction method for a Spring bean?

- What does component-scanning do?
- What is the behavior of the annotation `@Autowired` with regards to field injection, constructor injection and method injection?
- What do you have to do, if you would like to inject something into a private field? Ho does this impact testing?
- How does the `@Qualifier` annotation complement the use of `@Autowired`?

- What is a proxy object and what are the two different types of proxies Spring can create?
    - What are the limitations of these proxies (per type)?
    - What is the power of a proxy object and where are the disadvantages?

- What are the advantages of Java Config? What are the limitations?
- What does the `@Bean` annotation do?
- What is the default bean id if you only use `@Bean`? How can you override this?
- Why are you not allowed to annotate a final class with `@Configuration`
    - How do `@Configuration` annotated classes support singleton beans?
    - Why can't `@Bean` methods be final either?

- How do you configure profiles?, What are possible use cases where they might be useful?
- Can you use `@Bean` together with `@Profile`?
- Can you use `@Component` together with `@Profile`?
- How many profiles can you have?

- How do you inject scalar/literal values into Spring beans?
- What is `@Value` used for?
- What is Spring Expression Language (SpEL for short)?
- What is the *Environment* abstraction in Spring?
- Where can properties in the environment come from – there are many sources for properties – check the documentation if not sure. Spring Boot adds even more.
- What can you reference using SpEL?
- What is the difference between $ and # in `@Value` expressions?

Aspect oriented programming

- What is the concept of AOP? Which problem does it solve? What is a cross cutting concern?
    - Name three typical cross cutting concerns.
    - What two problems arise if you don't solve a cross cutting concern via AOP?
- What is a pointcut, a join point, an advice, an aspect, weaving?
- How does Spring solve (implement) a cross cutting concern?
- Which are the limitations of the two proxy-types?
    - What visibility must Spring bean methods have to be proxied using Spring AOP?
- How many advice types does Spring support. Can you name each one?
    - What are they used for?
    - Which *two* advices can you use if you would like to try and catch exceptions?

- What do you have to do to enable the detection of the `@Aspect` annotation?
  - What does `@EnableAspectJAutoProxy` do?
- If shown pointcut expressions, would you understand them?
  - For example, in the course we matched getter methods on Spring Beans, what would be the correct pointcut expression to match both getter *and* setter methods?
- What is the `JoinPoint` argument used for?
- What is a `ProceedingJoinPoint`? When is it used?

Data Management: JDBC, Transactions, JPA, Spring Data

- What is the difference between checked and unchecked exceptions?
  - Why does Spring prefer unchecked exceptions?
  - What is the data access exception hierarchy?
- How do you configure a `DataSource` in Spring? Which bean is very useful for development/test databases?

- What is the Template design pattern and what is the JDBC template?
- What is a callback? What are the three `JdbcTemplate` callback interfaces that can be used with queries? What is each used for? (You would not have to remember the interface names in the exam, but you should know what they do if you see them in a code sample).
- Can you execute a plain SQL statement with the JDBC template?
- When does the JDBC template acquire (and release) a connection - for every method called or once per template? Why?
- How does the `JdbcTemplate` support generic queries? How does it return objects and lists/maps of objects?

- What is a transaction? What is the difference between a local and a global transaction?
- Is a transaction a cross cutting concern? How is it implemented by Spring?
- How are you going to define a transaction in Spring?
  - What does @Transactional do? What is the `PlatformTransactionManager`?
- Is the JDBC template able to participate in an existing transaction?
- What is a transaction isolation level? How many do we have and how are they ordered?
- What is `@EnableTransactionManagement` for?
- What does transaction propagation mean?
- What happens if one `@Transactional` annotated method is calling another `@Transactional` annotated method on the *same* object instance?
- Where can the `@Transactional` annotation be used? What is a typical usage if you put it at class level?
- What does declarative transaction management mean?
- What is the default rollback policy? How can you override it?
- What is the default rollback policy in a JUnit test, when you use the `@RunWith(SpringJUnit4ClassRunner.class)` in JUnit 4 or `@ExtendWith(SpringExtension.class)` in JUnit 5, and annotate your `@Test` annotated method with `@Transactional`?
- Why is the term "*unit of work*" so important and why does JDBC *AutoCommit* violate this pattern?

- What does JPA stand for - what about ORM?
  - What is the idea behind an ORM? What are benefits/disadvantages or ORM?

    - What is a `PersistenceContext` and what is an `EntityManager`. What is the relationship between both?
    - Why do you need the `@Entity` annotation. Where can it be placed?
- What do you need to do in Spring if you would like to work with JPA?
- Are you able to participate in a given transaction in Spring while working with JPA?
- Which `PlatformTransactionManager`(s) can you use with JPA?
- What does `@PersistenceContext` do?
- What do you have to configure to use JPA with Spring? How does Spring Boot make this easier?

- What is an "instant repository"? (*hint:* recall Spring Data)
- How do you define an “instant” repository? Why is it an interface *not* a class?
- What is the naming convention for finder methods?
- How are Spring Data repositories implemented by Spring at runtime?
- What is `@Query` used for?

Spring Boot

- What is Spring Boot?
- What are the advantages of using Spring Boot?
- Why is it “opinionated”?
- How does it work? How does it know what to configure?
- What things affect what Spring Boot sets up?
- How are properties defined? Where is Spring Boot’s default property source?
- Would you recognize common Spring Boot annotations and configuration properties if you saw them in the exam?
- What is the difference between an embedded container and a WAR?
- What embedded containers does Spring Boot support?
- What does `@EnableAutoConfiguration` do?
- What about `@SpringBootApplication`?
- Does Spring Boot do component scanning? Where does it look by default?
- What is a Spring Boot starter POM? Why is it useful?
- Spring Boot supports both Java properties and YML files. Would you recognize and understand them if you saw them?
- Can you control logging with Spring Boot? How?

Note that the second Spring Boot section (*Going Further*) is not required for this exam.

*Remember: Unless a question explicitly references Spring Boot (like those in this section) you can assume Spring Boot is **not** involved in any question.*

Spring MVC and the Web Layer

- MVC is an abbreviation for a *design pattern*. What does it stand for and what is the idea behind it?
- Do you need *spring-mvc.jar* in your classpath or is it part of spring-core?
- What is the `DispatcherServlet` and what is it used for?
- Is the `DispatcherServlet` instantiated via an application context?

- What is a web application context? What extra scopes does it offer?

- What is the `@Controller` annotation used for?
- How is an incoming request mapped to a controller and mapped to a method?
- What is the difference between `@RequestMapping` and `@GetMapping`?
- What is `@RequestParam` used for?
- What are the differences between `@RequestParam` and `@PathVariable`?
- What are some of the parameter types for a controller method?
  - What other annotations might you use on a controller method parameter? (You can ignore form-handling annotations for this exam)
- What are some of the valid return types of a controller method?
-
- What is a `View` and what's the idea behind supporting different types of `View`?
- How is the right `View` chosen when it comes to the rendering phase?
- What is the `Model`?
- Why do you have access to the model in your `View`? Where does it come from?
- What is the purpose of the session scope?
- What is the default scope in the web context?
- Why are controllers testable artifacts?
- What does a `ViewResolver` do?

Security

Please note that `@Secured` and the Spring Security JSP tag library may be referenced in the exam but are not in the main course notes. @Secured is in the advanced section.

- What are authentication and authorization? Which must come first?
- Is security a cross cutting concern? How is it implemented internally?
- What is the delegating filter proxy?
- What is the security filter chain?
- What is a security context?

- Why do you need the `intercept-url`?
- In which order do you have to write multiple intercept-url's?
- What does the `**` pattern in an *antMatcher* or *mvcMatcher* do?
- Why is an *mvcMatcher* more secure than an *antMatcher*?
- Does Spring Security support password hashing? What is salting?

- Why do you need method security? What type of object is typically secured at the method level (think of its purpose not its Java type).
- What do `@PreAuthorized` and `@RolesAllowed` do? What is the difference between them?
  - What does Spring's `@Secured` do?
  - How are these annotations implemented?
  - In which security annotation are you allowed to use SpEL?
- Is it enough to hide sections of my output (e.g. JSP-Page or Mustache template)?
  - Spring security offers a security tag library for JSP, would you recognize it if you saw it in an example?

REST

- What does REST stand for?
- What is a resource?
- What does CRUD mean?
- Is REST secure? What can you do to secure it?
- What are safe REST operations?
- What are idempotent operations? Why is idempotency important?
- Is REST scalable and/or interoperable?
- Which HTTP methods does REST use?
- What is an `HttpMessageConverter`?
- Is REST normally stateless?

- What does `@RequestMapping` do?
- Is `@Controller` a stereotype? Is `@RestController` a stereotype?
  - What is a stereotype annotation? What does that mean?
- What is the difference between `@Controller` and `@RestController`?
- When do you need `@ResponseBody`?
- What does `@PathVariable` do?
- What are the HTTP status return codes for a successful GET, POST, PUT or DELETE operation?
- When do you need `@ResponseStatus`?
- Where do you need `@ResponseBody`? What about `@RequestBody`? Try not to get these muddled up!
- If you saw example Controller code, would you understand what it is doing? Could you tell if it was annotated correctly?
- Do you need Spring MVC in your classpath?
- What Spring Boot starter would you use for a Spring REST application?

- What are the advantages of the `RestTemplate`?
- If you saw an example using `RestTemplate` would you understand what it is doing?

Testing

- Do you use Spring in a unit test?
- What type of tests typically use Spring?
- How can you create a shared application context in a JUnit integration test?
- When and where do you use `@Transactional` in testing?
- How are mock frameworks such as Mockito or EasyMock used?
- How is `@ContextConfiguration` used?
- How does Spring Boot simplify writing tests?
- What does `@SpringBootTest` do? How does it interact with `@SpringBootApplication` and `@SpringBootConfiguration`?

Resources

http://spring.io/blog
Blog: Point your favorite RSS reader or come back for detailed, quality posts by Spring developers.

http://docs.spring.io/spring/docs/current/spring-framework-reference

Reference: The reference documentation (800+ pages) is available as html pages, a single html page and as a PDF document.

http://docs.spring.io/spring/docs/current/javadoc-api
Javadoc API

http://springbyexample.org
Spring By Example: Another good repository with good code samples is *SpringByExample*.

Conclusion

When you worked through this guide and know all the answers, we are pretty confident that you should pass the certification. It's recommended to do it as soon as possible and we wish you good luck with it.

Thank you again for choosing Pivotal as your education partner and good luck with your projects.

If you have encountered any errors, have any suggestions or enquiries please don't hesitate to contact your trainer or send an email to education@pivotal.io.

