Speed up Spring Boot startup time

Asked 9 years, 11 months ago Modified 1 year, 7 months ago Viewed 169k times

183

I have a Spring Boot application. I've added a lot of dependencies (unfortunately, looks I need all of them) and the startup time went up quite a lot. Just doing a `SpringApplication.run(source, args)` takes 10 seconds.

While that might not be much compared to what are "used" to, I'm unhappy that it takes that much, mostly because it breaks the development flow. The application itself is rather small at this point, so I assume most of the time is related to the added dependencies, not the app classes themselves.

I assume the issue is classpath scanning, but I am not sure how to:

- Confirm that is the issue (i.e. how to "debug" Spring Boot)
- If it really is the cause, how can I limit it, so it gets faster? For example, if I know that some dependency or package does not contain anything that Spring should be scanning, is there a way to limit that?

I assume that enhancing Spring to have parallel bean initialization during startup would speed up things, but that enhancement request has been open since 2011, without any progress. I see some other efforts in Spring Boot itself, such as Investigate Tomcat JarScanning speed improvements, but that is Tomcat specific and has been abandoned.

This article:

- http://www.nurkiewicz.com/2010/12/speeding-up-spring-integration-tests.html

although aimed at integration tests, suggests using `lazy-init=true`, however I do not know how to apply this to all beans in Spring Boot using Java configuration - any pointers here?

Any (other) suggestions would be welcome.

java **performance** **spring-boot** **startup**

Share Edit Follow

edited Mar 20, 2019 at 13:39

Raedwald
48.4k • 47 • 157 • 245

asked Dec 1, 2014 at 14:37

steady rain
2,326 • 3 • 16 • 18

1 Post your code. Normally only the package the application runner is defined is is scanned. If you have other packages defined for `@ComponentScan` those are scanned as well. Another thing is to make sure you

haven't enabled debug or trace logging as generally logging is slow, very slow. – M. Deinum Dec 2, 2014 at 7:46

1 If you use Hibernate it also tends to eat significant time at application start. – Knut Forkalsrud Mar 20, 2015 at 3:08

Spring's auto binding by type coupled with factory beans has the potential to be slow one you add a lot of beans and dependencies. – Knut Forkalsrud Mar 20, 2015 at 3:10

Or you can use caching, spring.io/guides/gs/caching – Cassian Mar 20, 2015 at 9:47

3 Thanks all for the comments - I would not be able to post the code unfortunately (a lot of internal jars), however I'm still looking for a way to debug this. Yes, I might be using A or B or doing X or Y, which slows it down. How do I determine this? If I add a dependency X, which has 15 transitive dependencies, how do I know which of those 16 slowed it down? If I can find out, is there anything I can do later to stop Spring from examining them? Pointers like that would be useful! – steady rain Mar 20, 2015 at 12:14

11 Answers

Sorted by: Highest score (default)

95

The other answers don't go into the depth I like to see and provide no scientific evidence. The Spring Boot team went through an exercise for reducing startup time for Boot 2.0, and ticket 11226 contains a lot of useful information. There is also a ticket 7939 open to adding timing information to condition evaluation, but it doesn't seem to have a specific ETA.

The most useful, and methodical approach for debugging Boot startup has been done by Dave Syer. https://github.com/dsyer/spring-boot-startup-bench

I had a similar use case as well, so I took Dave's approach of micro-benchmarking with JMH and ran with it. The result is the boot-benchmark project. I designed it such that it can be used to measure startup time for any Spring Boot application, using the executable jar produced by `bootJar` (previously called `bootRepackage` in Boot 1.5) Gradle task. Feel free to use it and provide feedback.

My findings are as follows:

1. CPU matters. A lot.
2. Starting the JVM with -Xverify:none helps significantly.
3. Excluding unnecessary autoconfigurations helps.
4. Dave recommended JVM argument -XX:TieredStopAtLevel=1, but my tests didn't show significant improvement with that. Also, `-XX:TieredStopAtLevel=1` would probably slow down your first request.
5. There have been reports of hostname resolution being slow, but I didn't find it to be a problem for the apps I tested.

Share Edit Follow edited Jan 8, 2023 at 9:29 answered Apr 5, 2018 at 2:07

Abhijit Sarkar
24k ● 22 ● 124 ● 242

It does not seem that your project builds under gradle `4.8.1`. Could you share which gradle version you used in your benchmarks? – filpa Jul 18, 2018 at 18:19

@user991710 Based on my Gradle wrapper, I'm using v4.6. "Does not build" is a very vague statement, if you've something more specific, create a gist and post the link here. Your gist should list the steps you followed, and the error you're getting. – Abhijit Sarkar Jul 18, 2018 at 23:44

2 To add on to this, could you please add an example as to how someone might use your benchmark with a custom application? Does it have to be added as a project similar to `minimal`, or may the jar simply be supplied? I attempted to do the former but did not get very far. – filpa Jul 19, 2018 at 17:52

4 Don't run `-Xverify:none` on production as it breaks code verification and you can run into trouble. `-XX:TieredStopAtLevel=1` is OK if you run an application for a small duration (a few seconds) otherwise it will be less productive as it will provide the JVM with long running optimizations. – loicmathieu Feb 15, 2019 at 15:16

2 Many pools (Oracle UCP for sure, but in my testing also Hikari and Tomcat) encrypt data in the pool. I actually don't know if they are encrypting the connection info, or wrapping the stream. Regardless, encryption uses random number generation and so having a highly available, high throughput entropy source makes a noticeable difference in performance. – Daniel May 2, 2019 at 15:30

82

Spring Boot does a lot of auto-configuration that may not be needed. So you may want to narrow down only auto-configuration that is needed for your app. To see full list of auto-configuration included, just run logging of `org.springframework.boot.autoconfigure` in DEBUG mode (`logging.level.org.springframework.boot.autoconfigure=DEBUG` in `application.properties`). Another option is to run spring boot application with `--debug` option: `java -jar myproject-0.0.1-SNAPSHOT.jar --debug`

There would be something like this in output:

```
=========================
AUTO-CONFIGURATION REPORT
=========================
```

Inspect this list and include only autoconfigurations you need:

```
@Configuration
@Import({
        DispatcherServletAutoConfiguration.class,
        EmbeddedServletContainerAutoConfiguration.class,
        ErrorMvcAutoConfiguration.class,
        HttpEncodingAutoConfiguration.class,
        HttpMessageConvertersAutoConfiguration.class,
        JacksonAutoConfiguration.class,
        ServerPropertiesAutoConfiguration.class,
        PropertyPlaceholderAutoConfiguration.class,
        ThymeleafAutoConfiguration.class,
        WebMvcAutoConfiguration.class,
```

```
        WebSocketAutoConfiguration.class,
})
public class SampleWebUiApplication {
```

Code was copied from this blog post.

Share Edit Follow

edited Oct 31, 2016 at 5:19

answered Mar 14, 2016 at 17:24

luboskrnac
24.5k ● 10 ● 84 ● 93

1 did you measure this ??? Was it much faster?? In my opinion this is an exceptional case, much more important to make sure that Spring test context cache works – idmitriev Jul 9, 2017 at 20:49

@idmitriev I just measured this on my application and my application started up at 53 seconds, compared to without exluding the autoconfiguration classes was 73 seconds. I did exclude many more classes than listed above though. – apkisbossin Mar 6, 2019 at 17:00

3 How to deal with Private configuration classes ? – user1767316 Jan 27, 2020 at 8:34

1 Is there a way to automatically know which auto-configurations are actually used? Maybe some long-running thing that adds up all auto-configs used throughout the app life-time and then you can poll an actuator endpoint to see that list? – payne Dec 22, 2021 at 16:24

@payne, I am not aware of anything like you are describing. – luboskrnac Dec 23, 2021 at 12:47

33

Spring Boot 2.2.M1 has added feature to support Lazy Initialization in Spring Boot.

By default, when an application context is being refreshed, every bean in the context is created and its dependencies are injected. By contrast, when a bean definition is configured to be initialized lazily it will not be created and its dependencies will not be injected until it's needed.

Enabling Lazy Initialization Set `spring.main.lazy-initialization` to **true**

When to Enable Lazy Initialization

lazy initialization can offer significant improvements in start up time but there are some notable downsides too and it's important to enable it with care

For more details please check Doc

Update:

Spring Boot Spring Boot 2.4.0 - Startup Endpoint

Spring Boot 2.4.0 has added a new Startup endpoint that can be used to identify beans that are taking longer than expected to start. You can get more details about the Application Startup tracking here

Share Edit Follow

edited Jan 17, 2021 at 18:10

answered Mar 20, 2019 at 13:00

Niraj Sonawane

11k ● 11 ● 80 ● 106

8 if you enable lazy initialization, first time loading is super fast, but when client accessing for the first time it may notice some delay. I really recommend this for the development not for the production. – Isuru Dewasurendra Jan 13, 2020 at 5:51

2 As @IsuruDewasurendra suggested, it is rightly not a recommended way, it can significantly increase the latency when the app starts serving load. – Narendra Jaggi Apr 2, 2020 at 7:25

1 It just kicks the can down the road. – Abhijit Sarkar Aug 3, 2020 at 21:27

I use Lazy Initialization only on development because the first access is so lazy, but it is good feature in Spring Boot. – Caio Nov 13, 2020 at 0:32

2 Even though first time access for users would be slow. This is got way to kick the startup that do not need any beans. Even in production I find this very acceptable compared to waiting for startup time EVERY time by the same person – Anddo Aug 8, 2022 at 14:15

13

As described in this question/answer, I think the best approach is to instead of adding only those you think you need, exclude the dependencies you know you don't need.

See: Minimise Spring Boot Startup Time

In summary:

You can see what is going on under the covers and enable debug logging as simple as specifying --debug when starting the application from the command-line. You can also specify debug=true in your application.properties.

Also, you can set the logging level in application.properties as simple as:

logging.level.org.springframework.web: DEBUG logging.level.org.hibernate: ERROR

If you detect an auto-configured module you don't want, it can be disabled. The docs for this can be found here: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using-boot-disabling-specific-auto-configuration

An example would look like:

```
@Configuration
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration {
}
```

Share Edit Follow

edited May 23, 2017 at 12:34

Community Bot

answered Sep 22, 2016 at 14:44

pczeus

1 ● 1 7,867 ● 4 ● 37 ● 53

9

Well there is entire list of possible actions described here:
https://spring.io/blog/2018/12/12/how-fast-is-spring

I will put the most important notes from Spring side (adjusted a little bit):

- Classpath exclusions from Spring Boot web starters:
  - Hibernate Validator
  - Jackson (but Spring Boot actuators depend on it). Use Gson if you need JSON rendering (only works with MVC out of the box).
  - Logback: use slf4j-jdk14 instead
- Use the spring-context-indexer. It's not going to add much, but every little helps.
- Don't use actuators if you can afford not to.
- Use Spring Boot 2.1 and Spring 5.1. Switch to 2.2 and 5.2 when they are available.
- Fix the location of the Spring Boot config file(s) with `spring.config.location` (command line argument or System property etc.). Example for testing in IDE: `spring.config.location=file://./src/main/resources/application.properties`.
- Switch off JMX if you don't need it with `spring.jmx.enabled=false` (this is the default in Spring Boot 2.2)
- Make bean definitions lazy by default. There's a new flag `spring.main.lazy-initialization=true` in Spring Boot 2.2 (use `LazyInitBeanFactoryPostProcessor` for older Spring).
- Unpack the fat jar and run with an explicit classpath.
- Run the JVM with `-noverify`. Also consider `-XX:TieredStopAtLevel=1` (that will slow down the JIT later at the expense of the saved startup time).

The mentioned `LazyInitBeanFactoryPostProcessor` (you can use it for Spring 1.5 if you cannot apply flag `spring.main.lazy-initialization=true` available from Spring 2.2):

```
public class LazyInitBeanFactoryPostProcessor implements BeanFactoryPostProcessor
{

  @Override
  public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory)
{
      for (String beanName : beanFactory.getBeanDefinitionNames()) {
        BeanDefinition definition = beanFactory.getBeanDefinition(beanName);
        definition.setLazyInit(true);
      }
  }
}
```

You can also use (or write your own - it's simple) something to analyse beans initialization time: https://github.com/lwaddicor/spring-startup-analysis

Hope it helps!

Share Edit Follow

answered Jan 8, 2020 at 21:32

Przemek Nowak
7,633 3 55 58

3

Checkout the Spring Boot Startup Report

https://github.com/maciejwalkowiak/spring-boot-startup-report

Spring Boot Startup Report library generates an interactive Spring Boot application startup report that lets you understand what contributes to the application startup time and perhaps helps to optimize it. startup report available in runtime as an interactive HTML page generating startup reports in integration tests flame chart search by class or an annotation

Features

- startup report available in runtime as an interactive HTML page
- generating startup reports in integration tests
- flame chart
- search by class or an annotation

Very cool utility IMHO.

Share Edit Follow

answered Mar 21, 2023 at 16:50

Paul Whelan
16.8k 12 52 84

2

WARNING: If you don't use Hibernate DDL for automatic DB schema generation and you don't use L2 cache, this answer is NOT applicable to you. Scroll ahead.

My finding is that Hibernate adds significant time to application startup. Disabling L2 cache and database initialization results in faster Spring Boot app startup. Leave cache ON for production and disable it for your development environment.

application.yml:

```
spring:
  jpa:
    generate-ddl: false
    hibernate:
```

```
    ddl-auto: none
  properties:
    hibernate:
      cache:
        use_second_level_cache: false
        use_query_cache: false
```

Test results:

1. L2 cache is ON and `ddl-auto: update`: 54 seconds

```
 INFO 5024 --- [restartedMain] o.s.web.context.ContextLoader : Root
WebApplicationContext: initialization completed in 23331 ms
 INFO 5024 --- [restartedMain] b.n.spring.Application : Started Application in
54.251 seconds (JVM running for 63.766)
```

2. L2 cache is OFF and `ddl-auto: none`: 32 seconds

```
 INFO 10288 --- [restartedMain] o.s.web.context.ContextLoader : Root
WebApplicationContext: initialization completed in 9863 ms
 INFO 10288 --- [restartedMain] b.n.spring.Application : Started Application in
32.058 seconds (JVM running for 37.625)
```

Gained 22 seconds! *Now I wonder what will I do with all this free time*

Share Edit Follow

edited May 28, 2021 at 8:23

answered Jan 1, 2019 at 22:27

naXa stands with Ukraine

37.6k ● 24 ● 202 ● 272

hibernate.hbm2ddl.auto=update has nothing to do with l2 cache. ddl..=update specifies to scan the current database schema, and compute the neccesary sql in order to update schema to reflect your entities. 'None' doesn't do this verification(also, doesn't try to update schema). Best practices is to use a tool like liquibase, where you will handle your schema changes,and you can also track them. – Radu Toader Apr 20, 2019 at 19:59

@RaduToader this question and my answer are about speeding up Spring Boot startup time. They have nothing to do with Hibernate DDL vs Liquibase discussion; these tools both have their pros and cons. My point is that we can disable the DB schema update and enable only when necessary. Hibernate takes significant time on startup even when model did not change since the last run (for comparing DB schema with autogenerated schema). The same point is true for L2 cache. – naXa stands with Ukraine Apr 20, 2019 at 21:39

yes, i know that, but my point was that this is a bit dangerous to not explain what it truly does. You might very easily end up with your db empty. – Radu Toader Apr 21, 2019 at 7:10

@RaduToader There was a link to a documentation page about DB initialization in my answer. Did you read it? It contains an exhaustive guide, listing all most popular tools (Hibernate and Liquibase, as well as JPA and Flyway). Also today I add a clear warning to the top of my answer. Do you think I need any other changes to explain consequences? – naXa stands with Ukraine Apr 29, 2019 at 15:07

Perfect. Thank you – Radu Toader Apr 30, 2019 at 7:58

1

If you're trying to optimize development turn-around for manual testing, I strongly recommend the use of devtools.

> Applications that use spring-boot-devtools will automatically restart whenever files on the classpath change.

Just recompile -- and the server will restart itself (for Groovy you only need to update the source file). if you're using an IDE (e.g. 'vscode'), it may automatically compile your java files, so just saving a java file can initiate a server restart, indirectly -- and Java becomes just as seamless as Groovy in this regard.

The beauty of this approach is that the incremental restart short-circuits some of the from-scratch startup steps -- so your service will be back up and running much more quickly!

Unfortunately, this doesn't help with startup times for deployment or automated unit testing.

Share Edit Follow

answered Dec 12, 2018 at 21:44

Brent Bradburn
54.5k ● 18 ● 161 ● 183

-3

I find it strange nobody suggested these optimizations before. Here're some general tips on optimizing project build and startup when developing:

- exclude development directories from antivirus scanner:
  - project directory
  - build output directory (if it's outside of project directory)
  - IDE indices directory (e.g. ~/.IntelliJIdea2018.3)
  - deployment directory (webapps in Tomcat)
- upgrade hardware. use faster CPU and RAM, better internet connection (for downloading dependencies) and database connection, switch to SSD (today NVMe SSD is the most performant storage). a video card doesn't matter.
- use latest Gradle and JVM versions. Source: easy performance improvements.
- parallel execution. By using more concurrent processes, parallel builds can reduce the overall build time significantly.

WARNINGS

1. the first option comes for the price of reduced security.

2. the second option costs money (obviously).

Share Edit Follow

edited May 28, 2021 at 8:21

answered Jan 1, 2019 at 22:05

naXa stands with Ukraine

37.6k ● 24 ● 202 ● 272

1 The question is about improving boot time, not compile time. – ArtOfWarfare Oct 15, 2019 at 16:02

@ArtOfWarfare read the question again. the question states the problem as "I'm unhappy that it takes that much [time], mostly because it breaks the development flow". I felt like this is a primary problem and addressed it in my answer. – naXa stands with Ukraine Oct 16, 2019 at 18:59

-5

In my case, there was too much breakpoints. When I clicked "Mute Breakpoints" and restarted application in debug mode, application started in 10 times faster.

Share Edit Follow

answered Apr 21, 2020 at 4:30

Daulet Kadirbekov

19 ● 5

-10

To me it sounds like you're using a wrong configuration setting. Start by checking myContainer and possible conflicts. To determine who is using the most resources you have to check the memory maps (see the amount of data!) for each dependency at a time - and that takes plenty of time, as well... (and SUDO privileges). By the way: are you usually testing the code against the dependencies?

Share Edit Follow

answered Mar 30, 2015 at 13:15

user4415984

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.