Spring Framework X Kotlin

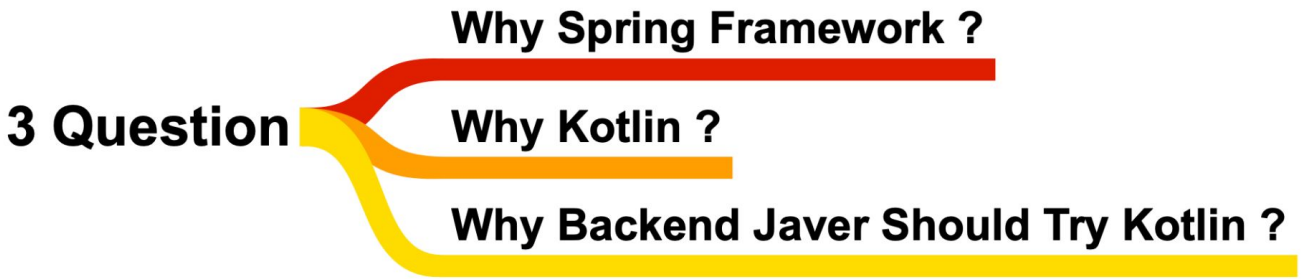
Jian Min (Vincent) Huang

黃健旻

About Me

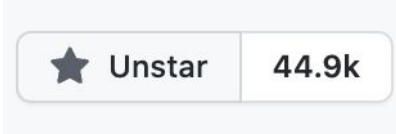
- 💎 https://jianminhuang.cc
- 👨‍💻 Coder, 🎤 Speaker, 👨‍🏫 Mentor and 🚀 Leader
- ♠️ Joint of JVM Backend and DevOps
- 🏆 Google Developer Expert (Kotlin)
- 💼 Cofounder of PureFunc Inc.
- 🎓 https://ithelp.ithome.com.tw/users/20119361/ironman/3973

Agenda


3 Question
Why Spring Framework ?
Why Kotlin ?
Why Backend Javer Should Try Kotlin ?

Why Spring Framework

- Most Popular & Comprehensive JVM Framework



Why Spring Framework

Spring is everywhere

Spring's flexible libraries are trusted by developers all over the world. Spring delivers delightful experiences to millions of end-users every day—whether that's streaming TV, online shopping, or countless other innovative solutions. Spring also has contributions from all the big names in tech, including Alibaba, Amazon, Google, Microsoft, and more.

Spring is flexible

Spring's flexible and comprehensive set of extensions and third-party libraries let developers build almost any application imaginable. At its core, Spring Framework's Inversion of Control (IoC) and Dependency Injection (DI) features provide the foundation for a wide-ranging set of features and functionality. Whether you're building secure, reactive, cloud-based microservices for the web, or complex streaming data flows for the enterprise, Spring has the tools to help.

Why Spring Framework

Spring is productive

Spring Boot transforms how you approach Java programming tasks, radically streamlining your experience. Spring Boot combines necessities such as an application context and an auto-configured, embedded web server to make microservice development a cinch. To go even faster, you can combine Spring Boot with Spring Cloud's rich set of supporting libraries, servers, patterns, and templates, to safely deploy entire microservices-based architectures into the cloud, in record time.

Spring is fast

Our engineers care deeply about performance. With Spring, you'll notice fast startup, fast shutdown, and optimized execution, by default. Increasingly, Spring projects also support the reactive (nonblocking) programming model for even greater efficiency. Developer productivity is Spring's superpower. Spring Boot helps developers build applications with ease and with far less toil than other competing paradigms. Embedded web servers, auto-configuration, and "fat jars" help you get started quickly, and innovations like LiveReload in Spring DevTools mean developers can iterate faster than ever before. You can even start a new Spring project in seconds, with the Spring Initializr at start.spring.io.

Why Spring Framework

Spring is secure

Spring has a proven track record of dealing with security issues quickly and responsibly. The Spring committers work with security professionals to patch and test any reported vulnerabilities. Third-party dependencies are also monitored closely, and regular updates are issued to help keep your data and applications as safe as possible. In addition, Spring Security makes it easier for you to integrate with industry-standard security schemes and deliver trustworthy solutions that are secure by default.

Spring is supportive

The Spring community is enormous, global, diverse, and spans folks of all ages and capabilities, from complete beginners to seasoned pros. No matter where you are on your journey, you can find the support and resources you need to get you to the next level: quickstarts, guides & tutorials, videos, meetups, support, or even formal training and certification.

Why Spring Framework

- Spring Language Support
  a. DSL
  b. Coroutine
  c. Transactions
- But Spring 4 Mix Kotlin ? (beware kapt, all open, no args, etc.)

Why Kotlin

- JVM Ecosystem
  a. Tools, Libraries, Resources, etc.
- Java -> Scala, Scala <-x-> Java
- Java -> Kotlin, Kotlin -> Java

Why Kotlin

- constructor, getter & setter, hashcode, equals, toString
- Lombok Framework
- data class
    a. Java record class
    b. Scala case class

```
data class User(val username: String, val password: String)
```

Why Kotlin

- var and val
- fun
- : vs extends and implement
- no semicolon and return
- single return can skip curly brackets
- if is statement no expression
- Singleton object

Why Kotlin

- Elvis Operator
- Type Inference and Aliases
- Powerful Collections
- Named Arguments
- Null Check and Safety
- Pretty Lambda as Java 😂
- String Interpolation

Why Kotlin

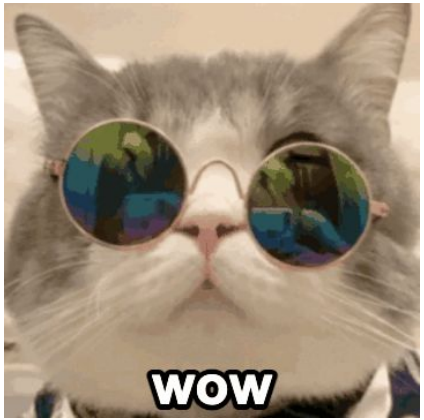
- Pattern Matching
- Operator Overloading
- Infix Notation
- Destructuring Declaration
- Scoped Function
- Extension Function
- Free to choose OO or FP

Should Backend Javer Use Kotlin ?

- Kotlin is a fantastic language
- But if your Java Spring techinque stack still on Web MVC
- It's a reasonable choice for keeping Java

But...

- If your Java Spring techinque stack is on Web Reactive



- Linus Torvalds: “Talk is cheap. Show me the code.”

Servlet, Per Req Per Thread (Blocking)

```
private fun getBlockResponse(ms: Long) = webClient.get()
    .uri("http://$domain:8888/delay/ms/$ms")
    .retrieve()
    .bodyToMono(object : ParameterizedTypeReference<Map<*, *>>() {})
    .block()
```

Servlet, Per Req Per Thread (Blocking)

```
fun blockMvc(time1: Long, time2: Long, time3: Long) =
    run {
        val delayTime1Res = getBlockResponse(time1)!!["totalTimeMillis"]
        val delayTime2Res = getBlockResponse(time2)!!["totalTimeMillis"]
        val delayTime3Res = getBlockResponse(time3)!!["totalTimeMillis"]

        mapOf(
            "delay${time1}Res" to delayTime1Res,
            "delay${time2}Res" to delayTime2Res,
            "delay${time3}Res" to delayTime3Res
        )
    }
```

Servlet, AsyncContext (Future)

```
private fun getFutureResponse(ms: Long) = webClient.get()
    .uri("http://$domain:8888/delay/ms/$ms")
    .retrieve()
    .bodyToMono(object : ParameterizedTypeReference<Map<*, *>>() {})
    .toFuture()
```

Servlet, AsyncContext (Future)

```
fun async(time1: Long, time2: Long, time3: Long) =
    arrayOf(getFutureResponse(time1), getFutureResponse(time2), getFutureResponse(time3))
        .let {
            allOf(*it)
                .thenApply { v ->
                    return@thenApply it.map { v1 -> v1.get() }
                }.thenApply { v ->
                    mapOf(
                        "delay${time1}Res" to v[0]["totalTimeMillis"],
                        "delay${time2}Res" to v[1]["totalTimeMillis"],
                        "delay${time3}Res" to v[2]["totalTimeMillis"]
                    )
                }
        }
```

WebFlux, Project Reactor

```
private fun getMonoResponse(ms: Long) = webClient.get()
    .uri("http://$domain:8888/delay/ms/$ms")
    .retrieve()
    .bodyToMono(object : ParameterizedTypeReference<Map<*, *>>() {})
```

WebFlux, Project Reactor

```
fun flux(time1: Long, time2: Long, time3: Long) =
    Flux.mergeSequential(getMonoResponse(time1), getMonoResponse(time2), getMonoResponse(time3))
        .collectList()
        .map { v ->
            mapOf(
                "delay${time1}Res" to v[0]["totalTimeMillis"],
                "delay${time2}Res" to v[1]["totalTimeMillis"],
                "delay${time3}Res" to v[2]["totalTimeMillis"]
            )
        }
```

WebFlux, Coroutine

```
private suspend fun getAwaitResponse(ms: Long) = webClient.get()
    .uri("http://$domain:8888/delay/ms/$ms")
    .retrieve()
    .awaitBody<Map<String, Long>>()
```

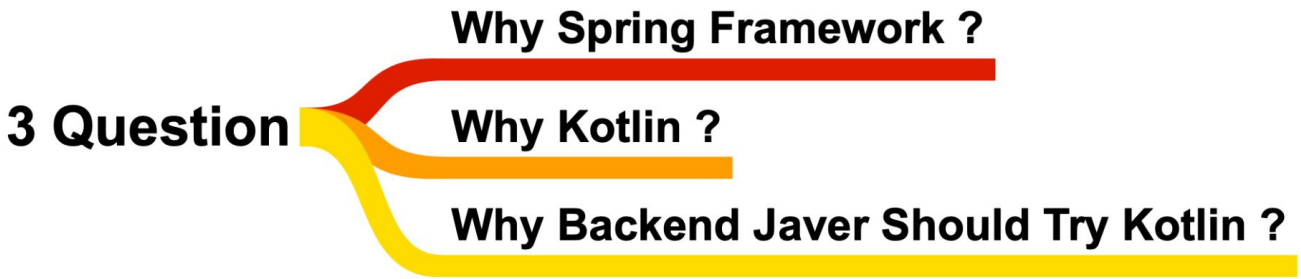
WebFlux, Coroutine

```
suspend fun coroutine(time1: Long, time2: Long, time3: Long) =
    coroutineScope {
        val delayTime1Res = async(Dispatchers.IO) { getAwaitResponse(time1) }
        val delayTime2Res = async(Dispatchers.IO) { getAwaitResponse(time2) }
        val delayTime3Res = async(Dispatchers.IO) { getAwaitResponse(time3) }

        mapOf(
            "delay${time1}Res" to delayTime1Res.await()["totalTimeMillis"]!!,
            "delay${time2}Res" to delayTime2Res.await()["totalTimeMillis"]!!,
            "delay${time3}Res" to delayTime3Res.await()["totalTimeMillis"]!!
        )
    }
```

Servlet, Per Req Per Thread (Blocking)

```
fun blockMvc(time1: Long, time2: Long, time3: Long) =
    run {
        val delayTime1Res = getBlockResponse(time1)!!["totalTimeMillis"]
        val delayTime2Res = getBlockResponse(time2)!!["totalTimeMillis"]
        val delayTime3Res = getBlockResponse(time3)!!["totalTimeMillis"]

        mapOf(
            "delay${time1}Res" to delayTime1Res,
            "delay${time2}Res" to delayTime2Res,
            "delay${time3}Res" to delayTime3Res
        )
    }
```

Conclusion


3 Question
Why Spring Framework ?
Why Kotlin ?
Why Backend Javer Should Try Kotlin ?


Thank you