# Asynchronous programming with Kotlin coroutines in Spring

Konrad Kamiński
Allegro.pl

```
@GetMapping("/users/{userId}/products”)
fun getProducts(@PathVariable userId: String): List<Product> {
    if (userValidationService.isValidUser(userId)) {
        return productService.getProducts(userId)
    } else {
        throw UnauthorizedUserException(userId)
    }
}
```

```kotlin
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```
suspend fun simpleFun(param: String): Int {
    delay(100)
    return 5678
}

fun simpleFun(param: String, callback: Continuation<Int>): Any {
    val delayResult = delay(100, object : Continuation<Unit> {
        override val context: CoroutineContext get() = callback.context
        override fun resume(value: Unit) = callback.resume(5678)
        override fun resumeWithException(ex: Throwable) =
                callback.resumeWithException(ex)
    })

    return when (delayResult === COROUTINE_SUSPENDED) {
        true  -> COROUTINE_SUSPENDED
        false -> 5678
    }
}
```

```kotlin
@GetMapping("/users/{userId}/products")
suspend fun getProducts(@PathVariable userId: String): List<Product> {
    if (userValidationService.isValidUser(userId)) {
        return productService.getProducts(userId)
    } else {
        throw UnauthorizedUserException(userId)
    }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: HandlerMethodArgumentResolver {
  override fun supportsParameter(param: MethodParameter) =
    param.method.isSuspend && isContinuationClass(param.parameterType)

  override fun resolveArgument(param: MethodParameter,
      container: ModelAndViewContainer,webRequest: NativeWebRequest,
      binderFactory: WebDataBinderFactory) =
    object: Continuation<Any> {
      val deferredResult = DeferredResult<Any>()

      override val context: CoroutineContext get() =
EmptyCoroutineContext
      override fun resume(value: Any) { deferredResult.setResult(value) }
      override fun resumeWithException(exception: Throwable) {
        deferredResult.setErrorResult(exception)
      }
   }.apply { container.model["deferred"] = deferredResult }
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

```
object: AsyncHandlerMethodReturnValueHandler {
  private val delegate = DeferredResultMethodReturnValueHandler()

  override fun supportsReturnType(returnType: MethodParameter): Boolean =
    returnType.method.isSuspend

  override fun handleReturnValue(returnValue: Any?,
      type: MethodParameter, container: ModelAndViewContainer,
      webRequest: NativeWebRequest) {
    val result = container.model["deferred"] as DeferredResult<*>
    return delegate.handleReturnValue(result, type, mavContainer,
      webRequest)
  }

  override fun isAsyncReturnValue(value: Any,type: MethodParameter) =
    returnValue === COROUTINE_SUSPENDED
}
```

# DEMO

https://github.com/konrad-kaminski/spring-kotlin-coroutine

# Spring 5 + Kotlin coroutines

- Asynchronous application
- Imperative style
- Non-blocking I/O
- Scalability

# spring-kotlin-coroutine

- @Cacheable
- @EventListener & CoroutineEventPublisher
- @Scheduled
- Functional style route definitions

https://github.com/konrad-kaminski/spring-kotlin-coroutine

# Thank you!

Konrad Kamiński
Allegro.pl