Coroutine Internals



Kevin Jones

@kevinrjones www.rocksolidknowledge.com



How Does This Actually Work?



Scheduling coroutines

Creating suspension points

What happens when a function is modified with 'suspend'

- Continuation Passing Style



Definitions

Coroutine

- Instance of a suspendable computation

Suspending function

 Function marked with the suspend keyword

Coroutine builder

- Bridge from non-suspending to suspending
- Takes a suspending lambda as a parameter



Definitions

Suspension point

- Point where coroutine may be suspended
- To suspend must call standard lib primitive to suspend execution

Continuation

State of the suspend coroutine at the suspension point



How Does a
Coroutine
Suspend
Operation?

Call suspendCoroutine

- Captures state of coroutine in continuation instance
- Continuation is passed into the block



suspendCoroutine (simplified)

```
public suspend fun <T> suspendCoroutine(
    block: (Continuation<T>) -> Unit): T { ... }
```



How does it Continue?

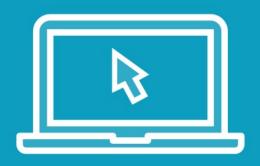
Calls resumeWith on the continuation

- Or calls resume
- or resumeWithException
- Helpers for resumeWith

Result passed to resumeWith is return from coroutine



Demo



Suspending coroutines



What are Coroutine Builders?

'Normal' (non-suspending) function

- Takes a suspending lambda as a parameter
- Calls startCoroutineCancellable on the lambda (indirectly)



startCoroutine?

Creates the initial continuation

Eventually runs the block as a function

- In the depths of the Kotlin libraries
- IntrinsicsJvm.kt



Invoking the Suspend Function



launch (not the actual code)

```
fun launch(
    context: CoroutineContext = EmptyCoroutineContext,
    block: suspend () -> Unit) =
        block.startCoroutine(Continuation(context) {
            result -> ...
})
```



Suspending Functions

Function marked with suspend modifier

- Compile transforms the function
- Uses CPS
- Continuation Passing Style



What is Continuation Passing Style?

A fancy name for callbacks

- Sort of

Rather than using direct code

- Use callbacks instead
- Suspension points are the callbacks



Using Callbacks This ...

```
suspend fun processValue(InitialValue: Int) {
    val value = getAValue()
    val anotherValue = getAnotherValue(initialValue, value)
    useTheValue(anotherValue)
suspend fun getAValue(): Int {}
suspend fun getAnotherValue(initialValue: Int, firstValue: Int): Int {}
suspend fun useTheValue(value: Int) {}
```



Would Look Like This

```
fun processValue(initialValue) {
    getAValue { value1 ->
       getAnotherValue(initialValue, value1) { value2 ->
           useTheValue(value2)
```



Not How Kotlin(or other languages) Does

This has performance issues

- Many objects created

Also difficult to create looping code etc



What Does Kotlin Do?

Transfoms code to use

- Continuation object
- State Machine



So What Actually Happens?

```
suspend fun processValue(initialValue: Int) {
   val value = getAValue()
   val anotherValue = getAnotherValue(initialValue, value)
   useTheValue(anotherValue)
}
```



Add Labels

```
suspend fun processValue(initialValue: Int) {
    // label 0
    val value = getAValue()
    // label 1
    val anotherValue = getAnotherValue(initialValue, value)
    // label 2
    useTheValue(anotherValue)
```



```
Create a Switch (logically)
suspend fun processValue(initialValue: Int) {
  switch(label) {
    case 0:
     val value = getAValue()
    case 1:
     val anotherValue = getAnotherValue(initialValue, value)
    case 2:
     useTheValue(anotherValue)
```

Add a Continuation Parameter

```
fun processValue(initialValue: Int, cont: Continuation) {
  switch(label) {
    case 0:
      getAValue(cont)
    case 1:
      getAnotherValue(initialValue, value, cont)
    case 2:
      useTheValues(anotherValue, cont)
```

Use Own Continuation

```
fun processValue(initialValue: Int, cont: Continuation) {
 val sm = object: ContinuationImpl {...}
  switch(label) {
    case 0:
      getAValue(sm)
    case 1:
      getAnotherValue(initialValue, value, sm)
    case 2:
      useTheValues(anotherValue, sm)
```



```
Capture State
fun processValue(initialValue: Int, cont: Continuation) {
    val sm = object: ContinuationImpl {...}
    switch(label) {
    case 0:
        sm.label = 1
        sm.initialValue = initialValue
        getAValue(sm)
    case 1:
        getAnotherValue(initialValue, value, sm)
```



Continuation Drives Callback

```
fun processValue(initialValue: Int, cont: Continuation) {
 val sm = object: ContinuationImpl {
   fun invokeSuspend(...) {
        processValue(null, this)
  switch(label) {
    case 0:
        useTheValues(anotherValue, sm)
```



Picking the Right Continuation

```
fun processValue(initialValue: Int, cont: Continuation) {
 val sm = cont as ThisSM ?: object: ThisSM {
   fun invokeSuspend(...) {
        processValue(null, this)
  switch(label) {
    case 0:
        useTheValues(anotherValue, sm)
```



Next State

```
fun processValue(initialValue: Int, cont: Continuation) {
    val sm = ...
    switch(label) {
      case 0:
      case 1:
        val initialValue = sm.initialValue
        val value = sm.result as Int
        getAnotherValue(initialValue, value, sm)
```

Summary



Coroutine Builders

- launch, async and others

Suspend Functions

- Compiled to use CPS

Structured Concurrency

- Track coroutines

