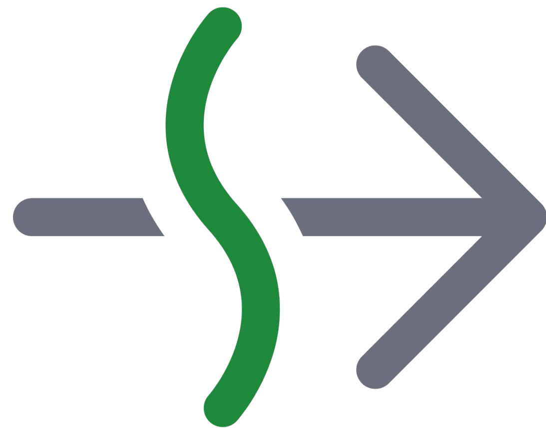
А вы можете сказать, что всё-таки такое «корутина» в Kotlin?



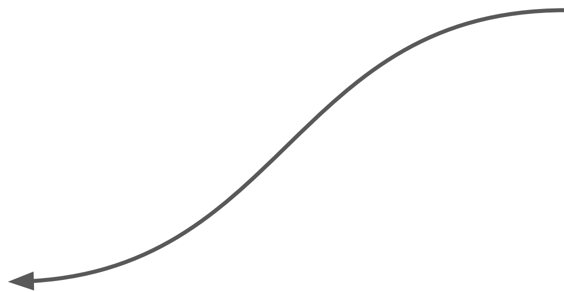
@lounres | Gleb Minaev

Привет! Давайте знакомиться!

- Исследователь-математик-программист в Центре Научного Программирования при МФТИ.
- Люблю заниматься разной алгоритмической математикой. В том числе, решать математические задачи методом делегирования компьютеру.
- Член комиссий по составлению Московской Математической Олимпиады и «Математического праздника».
- Преподаватель математического анализа в 57-ой школе.
- В прошлом дважды призёр Всероссийской Олимпиады Школьников по математике.

Глеб Минаев

Что это?..



```kotlin
var a = 0
var b = 1

while (true) {
    println(a)
    a = b.also { b = a + b }
}
```

```kotlin
var a = 0
var b = 1

while (true) {
    println(a)
    a = b.also { b = a + b }
}
```

Что это?..

Числа Фибоначчи!

Вывод:

```
0
1
1
2
3
5
8
13
21
34
55
89
```

```kotlin
val fibonacci = sequence {
    var a = 0
    var b = 1

    while (true) {
        yield(a)
        a = b.also { b = a + b }
    }
}

fibonacci.forEach { println(it) }
```

Вывод (всё ещё):

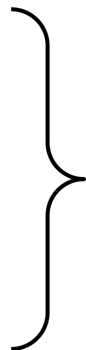
0
1
1
2
3
5
8
13
21
34
55
89

```kotlin
val fibonacci = sequence {
    var a = 0
    var b = 1

    while (true) {
        yield(a)
        a = b.also { b = a + b }
    }
}

fibonacci.take(6).forEach { println(it) }
```

Вывод (неожиданно (нет)):

0
1
1
2
3
5

```kotlin
val fibonacci = sequence {
    var a = 0
    var b = 1

    while (true) {
        yield(a)
        a = b.also { b = a + b }
    }
}

fibonacci.take(6).forEach { println(it) }
```



Если лямбда вычисляется целиком, то как она закончила свою работу?

```kotlin
val fibonacci = sequence {
    var a = 0
    var b = 1

    while (true) {
        yield(a)
        a = b.also { b = a + b }
    }
}

fibonacci.take(6).forEach { println(it) }
```

Остановите,
я сойду!

Если лямбда вычисляется целиком,
то как она закончила свою работу?

О чём будет доклад?

1. Общее представление о том, как пользоваться корутинами.

О чём будет доклад?

1. Общее представление о том, как пользоваться корутинами.
2. Захват и подвешивание корутины.

О чём будет доклад?

1. Общее представление о том, как пользоваться корутинами.
2. Захват и подвешивание корутины.
3. Пример использования: интеграция с асинхронными API.

О чём будет доклад?

1. Общее представление о том, как пользоваться корутинами.
2. Захват и подвешивание корутины.
3. Пример использования: интеграция с асинхронными API.
4. Создание и запуск корутины.

О чём будет доклад?

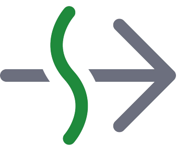
1. Общее представление о том, как пользоваться корутинами.
2. Захват и подвешивание корутины.
3. Пример использования: интеграция с асинхронными API.
4. Создание и запуск корутины.
5. Пример использования: имплементация `iterator`.

О чём будет доклад?

1. Общее представление о том, как пользоваться корутинами.
2. Захват и подвешивание корутины.
3. Пример использования: интеграция с асинхронными API.
4. Создание и запуск корутины.
5. Пример использования: имплементация `iterator`.
6. Нулевое приближение подкапотной корутин.

О чём будет доклад?

1. Общее представление о том, как пользоваться корутинами.
2. Захват и подвешивание корутины.
3. Пример использования: интеграция с асинхронными API.
4. Создание и запуск корутины.
5. Пример использования: имплементация `iterator`.
6. Нулевое приближение подкапотной корутин.
7. Выводы.



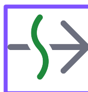
Общее представление о том, как пользоваться корутинами.

```kotlin
public suspend fun <T> Flow<T>.all(predicate: suspend (T) -> Boolean): Boolean {
    var foundCounterExample = false
    collectWhile {
        val satisfies = predicate(it)
        if (!satisfies) foundCounterExample = true
        satisfies
    }
    return !foundCounterExample
}
```

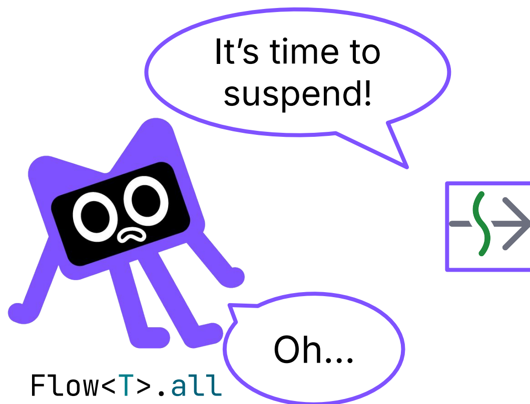
```kotlin
public suspend fun <T> Flow<T>.all(predicate: suspend (T) → Boolean): Boolean {
    var foundCounterExample = false
    collectWhile {
        val satisfies = predicate(it)
        if (!satisfies) foundCounterExample = true
        satisfies
    }
    return !foundCounterExample
}
```

```kotlin
public suspend fun <T> Flow<T>.all(predicate: suspend (T) → Boolean): Boolean {
    var foundCounterExample = false
    collectWhile {
        val satisfies = predicate(it)
        if (!satisfies) foundCounterExample = true
        satisfies
    }
    return !foundCounterExample
}
```

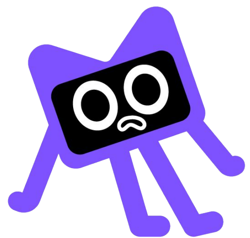
```kotlin
public suspend fun <T> Flow<T>.all(predicate: suspend (T) → Boolean): Boolean {
    var foundCounterExample = false
    collectWhile {
        val satisfies = predicate(it)
        if (!satisfies) foundCounterExample = true
        satisfies
    }
    return !foundCounterExample
}
```

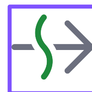


```kotlin
public suspend fun <T> Flow<T>.all(predicate: suspend (T) → Boolean): Boolean {
    var foundCounterExample = false
    collectWhile {
        val satisfies = predicate(it)
        if (!satisfies) foundCounterExample = true
        satisfies
    }
    return !foundCounterExample
}
```



```kotlin
public suspend fun <T> Flow<T>.all(predicate: suspend (T) → Boolean): Boolean {
    var foundCounterExample = false
    collectWhile {
        val satisfies = predicate(it)
        if (!satisfies) foundCounterExample = true
        satisfies
    }
    return !foundCounterExample
}
```

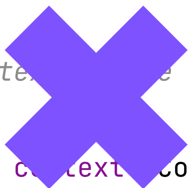




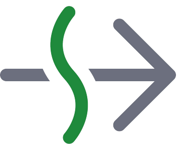
```kotlin
/**
* Interface representing a continuation after a suspension point that returns a value of type `T`.
*/
@SinceKotlin("1.3")
public interface Continuation<in T> {
    /**
     * The context of the coroutine that corresponds to this continuation.
     */
    public val context: CoroutineContext

    /**
     * Resumes the execution of the corresponding coroutine passing a successful or failed [result] as the
     * return value of the last suspension point.
     */
    public fun resumeWith(result: Result<T>)
}
```

```kotlin
/**
* Interface representing a continuation after a suspension point that returns a value of type `T`.
*/
@SinceKotlin("1.3")
public interface Continuation<in T> {
    /**
     * The context of the coroutine that corresponds to this continuation.
     */
    public val context: CoroutineContext

    /**
     * Resumes the execution of the corresponding coroutine passing a successful or failed [result] as the
     * return value of the last suspension point.
     */
    public fun resumeWith(result: Result<T>)
}
```

```kotlin
/**
* Interface representing a continuation after a suspension point that returns a value of type `T`.
*/
@SinceKotlin("1.3")
public interface Continuation<in T> {
    /**
     * The conte  e coroutine that corresponds to this continuation.
     */
    public val co  text  CoroutineContext

    /**
     * Resumes the execution of the corresponding coroutine passing a successful or failed [result] as the
     * return value of the last suspension point.
     */
    public fun resumeWith(result: Result<T>)
}
```



По сути: T | Throwable



Захват и подвешивание корутины.

```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) -> Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) → Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

```kotlin
suspend fun foo() {
    val r = suspendCoroutineUninterceptedOrReturn {



    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) -> Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

```kotlin
suspend fun foo() {
    val r = suspendCoroutineUninterceptedOrReturn<Int> { // it: Continuation<Int>



    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) -> Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

```kotlin
suspend fun foo() {
    val r = suspendCoroutineUninterceptedOrReturn<Int> { // it: Continuation<Int>
        // Если кинуть здесь исключение, оно пробросится выше.


    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) -> Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

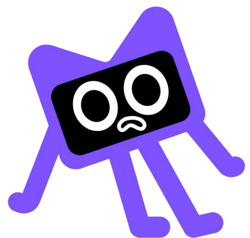
```kotlin
suspend fun foo() {
    val r = suspendCoroutineUninterceptedOrReturn<Int> { // it: Continuation<Int>
        // Если кинуть здесь исключение, оно пробросится выше.
        // Если вернуть в лямбду значение, то оно вернётся из suspendCoroutineUninterceptedOrReturn.

    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) -> Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

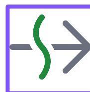
```kotlin
suspend fun foo() {
    val r = suspendCoroutineUninterceptedOrReturn<Int> { // it: Continuation<Int>
        // Если кинуть здесь исключение, оно пробросится выше.
        // Если вернуть в лямбду значение, то оно вернётся из suspendCoroutineUninterceptedOrReturn.
        // Если вернуть в лямбду COROUTINE_SUSPENDED, то корутина подвесится и её можно будет запустить позже.
    }
}
```

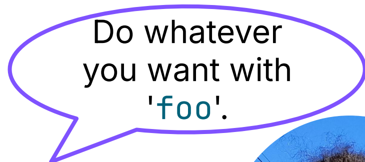
```kotlin
@SinceKotlin("1.3")
@InlineOnly
@Suppress("WRONG_INVOCATION_KIND", "UNUSED_PARAMETER", "RedundantSuspendModifier")
public suspend inline fun <T> suspendCoroutineUninterceptedOrReturn(crossinline block: (Continuation<T>) → Any?): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    throw NotImplementedError("Implementation of suspendCoroutineUninterceptedOrReturn is intrinsic")
}
```

```kotlin
suspend fun foo() {
    val r = suspendCoroutineUninterceptedOrReturn<Int> { // it: Continuation<Int>
        // Если кинуть здесь исключение, оно пробросится выше.
        // Если вернуть в лямбду значение, то оно вернётся из suspendCoroutineUninterceptedOrReturn.
        // Если вернуть в лямбду COROUTINE_SUSPENDED, то корутина подвесится и её можно будет запустить позже.
    }
}
```



foo





User with acquired coroutine

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) -> Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) -> Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = ...
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) → Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) → Unit
): T = ...
```

Так как обе функции всегда подвешивают корутину

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) → Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) → Unit
): T = ...
```

Может быть дополнительно использован, чтобы контролировать отмену Job'ы

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) -> Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = ...
```

```kotlin
suspend fun foo() {
    val r = suspendCancellableCoroutine {



    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) → Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) → Unit
): T = ...
```

```kotlin
suspend fun foo() {
    val r = suspendCancellableCoroutine<Int> { // it: CancellableContinuation<Int>



    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) -> Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = ...
```

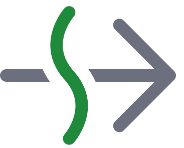
```kotlin
suspend fun foo() {
    val r = suspendCancellableCoroutine<Int> { // it: CancellableContinuation<Int>
        // it.cancel(), чтобы отменить корутину



    }
}
```

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) -> Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = ...
```

```kotlin
suspend fun foo() {
    val r = suspendCancellableCoroutine<Int> { // it: CancellableContinuation<Int>
        // it.cancel(), чтобы отменить корутину
        // it.resumeWithException(...), чтобы suspendCancellableCoroutine кинуло исключение

    }
}
```

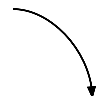
```kotlin
@SinceKotlin("1.3")
@InlineOnly
public suspend inline fun <T> suspendCoroutine(crossinline block: (Continuation<T>) -> Unit): T {
    contract { callsInPlace(block, InvocationKind.EXACTLY_ONCE) }
    return ...
}

public suspend inline fun <T> suspendCancellableCoroutine(
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = ...
```

```kotlin
suspend fun foo() {
    val r = suspendCancellableCoroutine<Int> { // it: CancellableContinuation<Int>
        // it.cancel(), чтобы отменить корутину
        // it.resumeWithException(...), чтобы suspendCancellableCoroutine кинуло исключение
        // it.resume(...), чтобы получить этот аргумент из suspendCancellableCoroutine
    }
}
```



Пример использования: интеграция с асинхронными API.

java.util.concurrent.CompletionStage



```kotlin
suspend fun <T> CompletionStage<T>.await(): T = ???
```

```kotlin
suspend fun <T> CompletionStage<T>.await(): T =
    suspendCoroutine {




    }
```

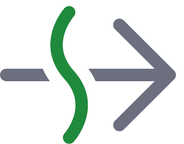
```kotlin
suspend fun <T> CompletionStage<T>.await(): T =
   suspendCoroutine<T> { continuation: Continuation<T> ->




   }
```

```kotlin
suspend fun <T> CompletionStage<T>.await(): T =
   suspendCoroutine { continuation ->
       handle {


       }
   }
```

```kotlin
suspend fun <T> CompletionStage<T>.await(): T =
    suspendCoroutine { continuation ->
        handle { result: T?, throwable: Throwable? ->


        }
    }
```

```kotlin
suspend fun <T> CompletionStage<T>.await(): T =
   suspendCoroutine { continuation ->
       handle { result: T?, throwable: Throwable? ->
           if (throwable == null) continuation.resume(result as T)

       }
   }
```

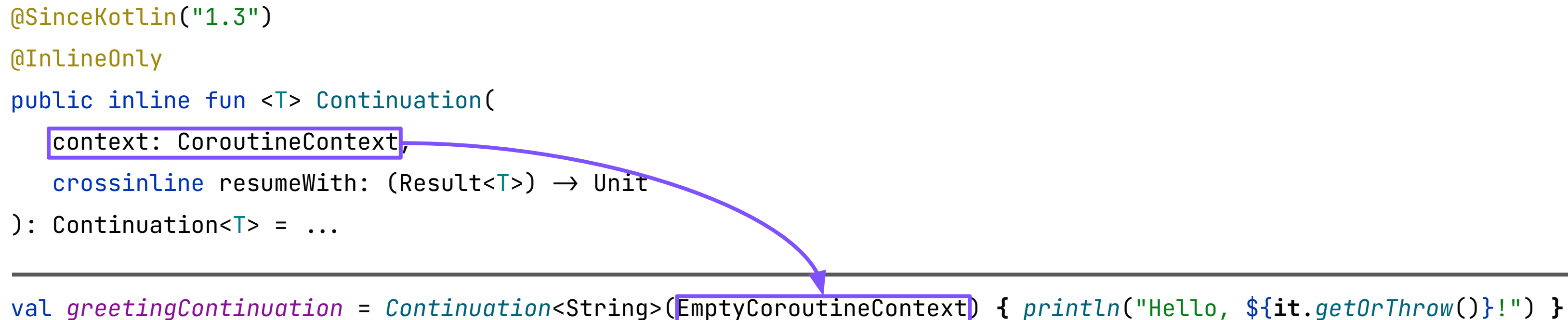
```kotlin
suspend fun <T> CompletionStage<T>.await(): T =
   suspendCoroutine { continuation ->
       handle { result: T?, throwable: Throwable? ->
           if (throwable == null) continuation.resume(result as T)
           else continuation.resumeWithException(throwable)
       }
   }
```



Создание и запуск корутины.

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public inline fun <T> Continuation(
    context: CoroutineContext,
    crossinline resumeWith: (Result<T>) -> Unit
): Continuation<T> = ...
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
```



@SinceKotlin("1.3")
@InlineOnly
public inline fun <T> Continuation(
context: CoroutineContext,
crossinline resumeWith: (Result<T>) → Unit
): Continuation<T> = ...
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }

```kotlin
@SinceKotlin("1.3")
@InlineOnly
public inline fun <T> Continuation(
    context: CoroutineContext,
    crossinline resumeWith: (Result<T>) → Unit
): Continuation<T> = ...
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () -> T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() -> T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
```

Запускаемая корутина

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

То, что корутина выполнит по завершении

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

Дополнительно: аргумент корутины

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
   completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
   receiver: R,
   completion: Continuation<T>
): Continuation<Unit>
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
suspend fun getConferenceName(): String = "JPoint"

val conferenceGreetingContinuation = ::getConferenceName.createCoroutineUnintercepted(greetingContinuation)

conferenceGreetingContinuation.resume(Unit)
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
suspend fun getConferenceName(): String = "JPoint"

val conferenceGreetingContinuation = ::getConferenceName.createCoroutineUnintercepted(greetingContinuation)

conferenceGreetingContinuation.resume(Unit)
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
suspend fun getConferenceName(): String = "JPoint"

val conferenceGreetingContinuation = ::getConferenceName.createCoroutineUnintercepted(greetingContinuation)

conferenceGreetingContinuation.resume(Unit)
```

```kotlin
@SinceKotlin("1.3")
public expect fun <T> (suspend () → T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

@SinceKotlin("1.3")
public expect fun <R, T> (suspend R.() → T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

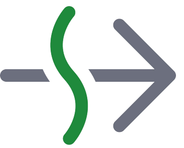
```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
suspend fun getConferenceName(): String = "JPoint"

val conferenceGreetingContinuation = ::getConferenceName.createCoroutineUnintercepted(greetingContinuation)

conferenceGreetingContinuation.resume(Unit)
```

createCoroutineUnintercepted → createCoroutine

```kotlin
@SinceKotlin("1.3")
public fun <T> (suspend () → T).createCoroutine(
    completion: Continuation<T>
): Continuation<Unit> = ...

@SinceKotlin("1.3")
public fun <R, T> (suspend R.() → T).createCoroutine(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit> = ...
```

```kotlin
val greetingContinuation = Continuation<String>(EmptyCoroutineContext) { println("Hello, ${it.getOrThrow()}!") }
suspend fun getConferenceName(): String = "JPoint"

val conferenceGreetingContinuation = ::getConferenceName.createCoroutine(greetingContinuation)

conferenceGreetingContinuation.resume(Unit)
```

→ Hello, JPoint!

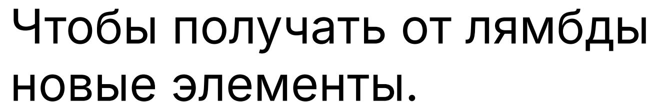


Пример использования: имплементация `iterator`.

```kotlin
@RestrictsSuspension
interface IteratorBuilder<in T> {
    suspend fun yield(element: T)
}

fun <T> myIterator(builder: suspend IteratorBuilder<T>.() -> Unit): Iterator<T> = TODO()

myIterator { this: IteratorBuilder<Int>
    yield(57)
    yield(179)
}
```

```kotlin
private class IteratorBuilderMachine<T> : IteratorBuilder<T>, Iterator<T>, Continuation<Unit>
```

```kotlin
private class IteratorBuilderMachine<T> : IteratorBuilder<T>, Iterator<T>, Continuation<Unit>
```



Чтобы получать от пользователя запрос на новые элементы.

Чтобы получить от лямбды сигнал завершения.

```kotlin
private class IteratorBuilderMachine<T> : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {
   sealed interface State<out T> {
       data class Waits(
           val continuation: Continuation<Unit>
       ) : State<Nothing>
       data class WaitsWithElement<out T>(
           val continuation: Continuation<Unit>,
           val element: T
       ) : State<T>
       data object Finished : State<Nothing>
   }

   var state: State<T>

}
```

```kotlin
private class IteratorBuilderMachine<T> : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {
    sealed interface State<out T> {
        data class Waits(
            val continuation: Continuation<Unit>
        ) : State<Nothing>
        data class WaitsWithElement<out T>(
            val continuation: Continuation<Unit>,
            val element: T
        ) : State<T>
        data object Finished : State<Nothing>
    }

    var state: State<T>

}
```

Вычисление не закончено. Существование следующего элемента неизвестно.

Вычисление не закончено. Следующий элемент был выдан лямбдой, но не передан пользователю.

Вычисление закончено.

```kotlin
private class IteratorBuilderMachine<T> : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {

    // ...

    override fun hasNext(): Boolean { TODO() }

    override fun next(): T { TODO() }

    override suspend fun yield(element: T) { TODO() }

    override fun resumeWith(result: Result<Unit>) { TODO() }

}
```

```
override suspend fun yield(element: T) {




}

override fun resumeWith(result: Result<Unit>) {

}
```

Сигнал лямбды о новом элементе.

```
override suspend fun yield(element: T) {

}
```

Сигнал лямбды о своём конце вычисления.

```
override fun resumeWith(result: Result<Unit>) {

}
```

Сигнал лямбды о новом элементе.

```
override suspend fun yield(element: T) {
    suspendCoroutineUninterceptedOrReturn {


    }
}
```

Сигнал лямбды о своём конце вычисления.

```
override fun resumeWith(result: Result<Unit>) {

}
```

Сигнал лямбды о новом элементе.

```
override suspend fun yield(element: T) {
    suspendCoroutineUninterceptedOrReturn<Unit> { continuation: Continuation<Unit> ->



    }
}
```

Сигнал лямбды о своём конце вычисления.

```
override fun resumeWith(result: Result<Unit>) {

}
```

Сигнал лямбды о новом элементе.

```
override suspend fun yield(element: T) {
    suspendCoroutineUninterceptedOrReturn { continuation ->
        state = State.WaitsWithElement(continuation, element)

    }
}
```

Сигнал лямбды о своём конце вычисления.

```
override fun resumeWith(result: Result<Unit>) {

}
```

Сигнал лямбды о новом элементе.

```
override suspend fun yield(element: T) {
    suspendCoroutineUninterceptedOrReturn { continuation ->
        state = State.WaitsWithElement(continuation, element)
        COROUTINE_SUSPENDED
    }
}
```

Сигнал лямбды о своём конце вычисления.

```
override fun resumeWith(result: Result<Unit>) {

}
```

Сигнал лямбды о новом элементе.

```
override suspend fun yield(element: T) {
    suspendCoroutineUninterceptedOrReturn { continuation ->
        state = State.WaitsWithElement(continuation, element)
        COROUTINE_SUSPENDED
    }
}
```

Сигнал лямбды о своём конце вычисления.

```
override fun resumeWith(result: Result<Unit>) {
    state = State.Finished
}
```

```
override fun hasNext(): Boolean =
```

Запрос пользователя о присутствия элемента

```
override fun hasNext(): Boolean =
```

Запрос пользователя о присутствия элемента

```
override fun hasNext(): Boolean =
    when (val currentState = state) {
        is State.Waits →







        is State.WaitsWithElement<T> →
        State.Finished →
    }
```

Запрос пользователя о присутствия элемента

```
override fun hasNext(): Boolean =
    when (val currentState = state) {
        is State.Waits ->







        is State.WaitsWithElement<T> ->
        State.Finished -> false
    }
```

Запрос пользователя о присутствия элемента

```kotlin
override fun hasNext(): Boolean =
    when (val currentState = state) {
        is State.Waits →







        is State.WaitsWithElement<T> → true
        State.Finished → false
    }
```

Запрос пользователя о присутствия элемента

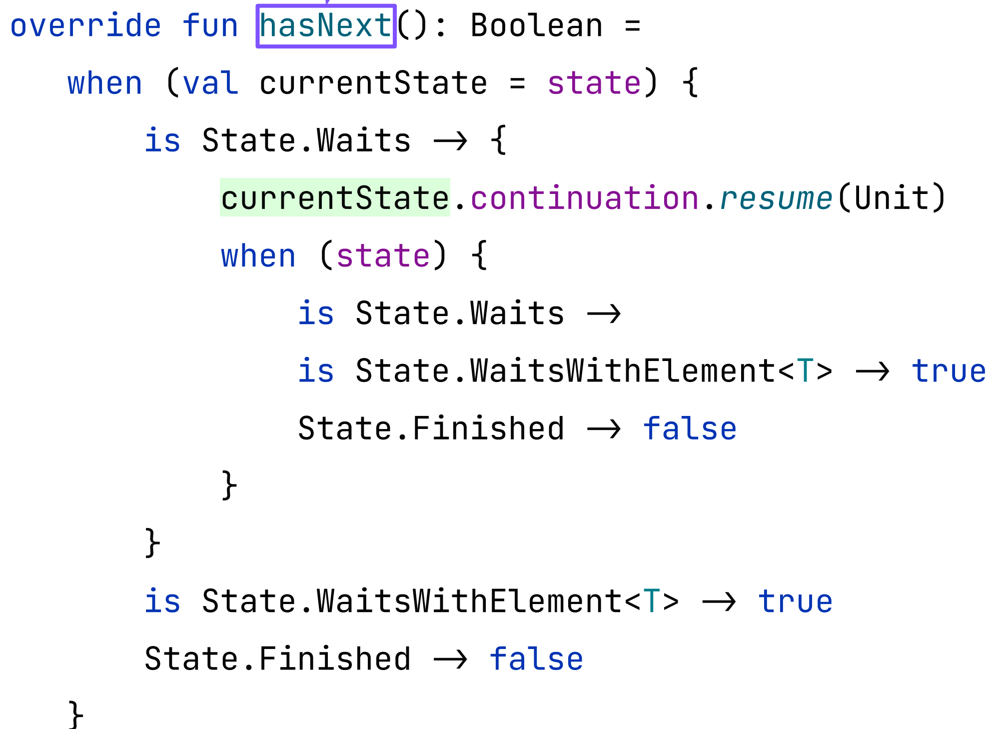
```
override fun hasNext(): Boolean =
    when (val currentState = state) {
        is State.Waits -> {
            currentState



        }
        is State.WaitsWithElement<T> -> true
        State.Finished -> false
    }
```

```
data class Waits(
    val continuation: Continuation<Unit>
) : State<Nothing>
```

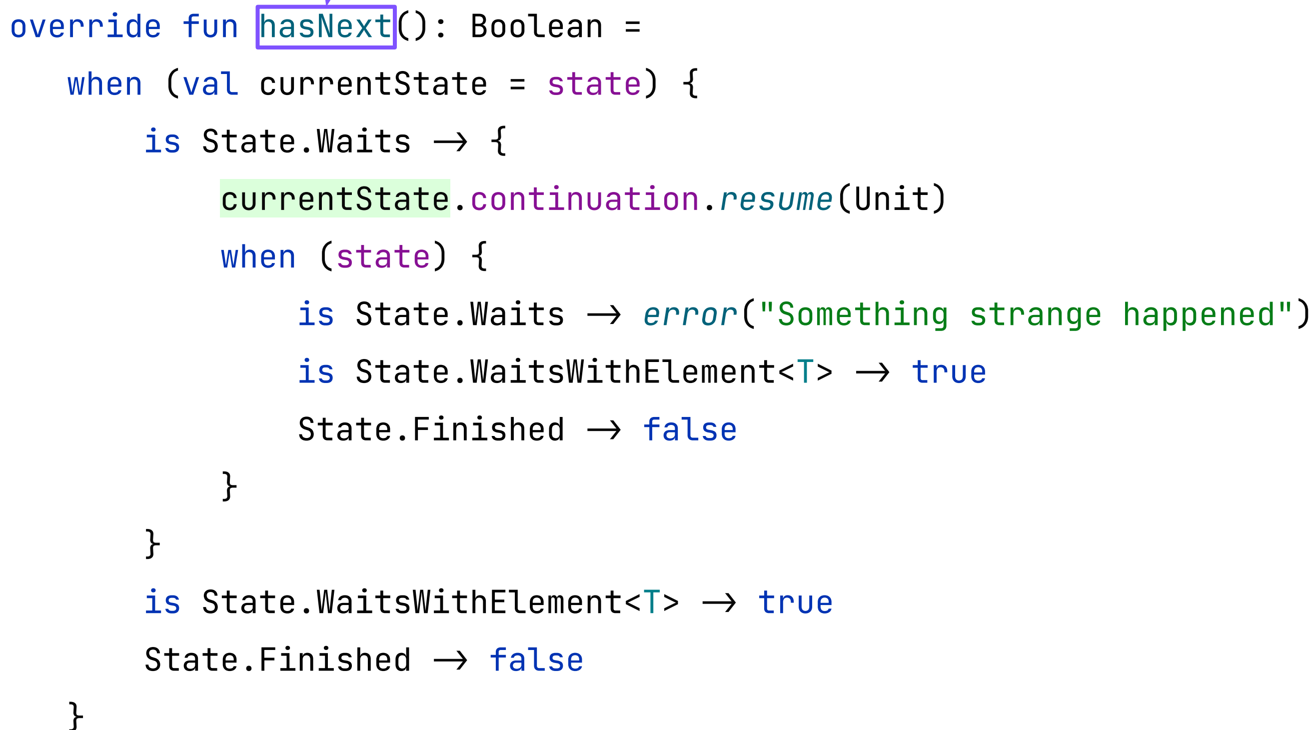
Запрос пользователя о присутствия элемента

```
override fun hasNext(): Boolean =
   when (val currentState = state) {
       is State.Waits → {
           currentState.continuation.resume(Unit)







       }
       is State.WaitsWithElement<T> → true
       State.Finished → false
   }
```

Запрос пользователя о присутствия элемента



Запрос пользователя о присутствия элемента



Запрос пользователя о присутствия элемента

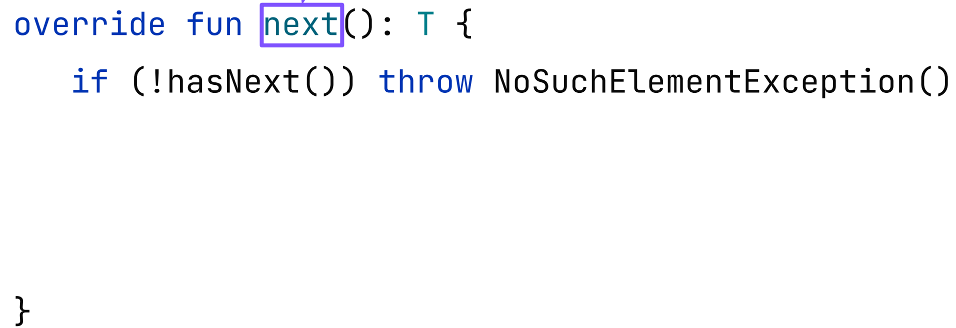
```
override fun hasNext(): Boolean =
    when (val currentState = state) {
        is State.Waits → {
            currentState.continuation.resume(Unit)
            when (state) {
                is State.Waits → error("Something strange happened")
                is State.WaitsWithElement<T> → true
                State.Finished → false
            }
        }
        is State.WaitsWithElement<T> → true
        State.Finished → false
    }
```

Следствие: если вернули `true`, то состояние будет в типе `State.WaitsWithElement<T>`

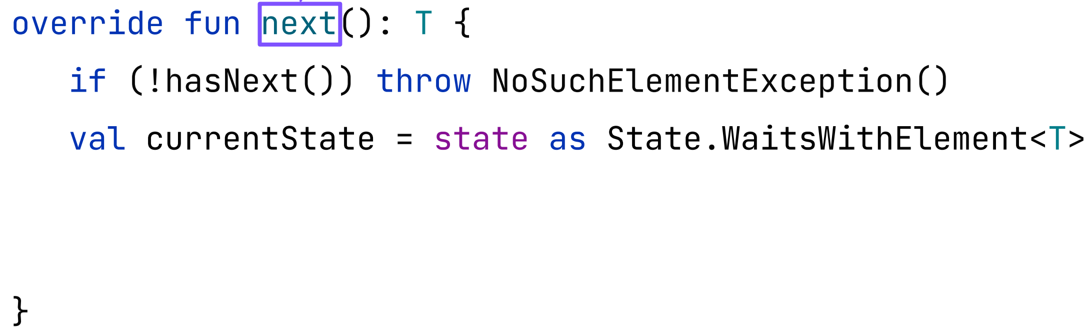
```kotlin
override fun next(): T {



}
```

Запрос пользователя следующего элемента

```
override fun next(): T {



}
```

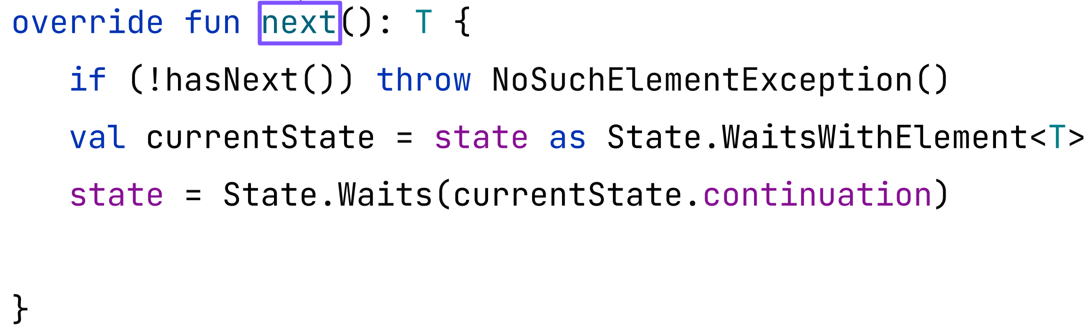



```
override fun next(): T {
    if (!hasNext()) throw NoSuchElementException()



}
```




```
override fun next(): T {
    if (!hasNext()) throw NoSuchElementException()
    val currentState = state as State.WaitsWithElement<T>


}
```

Запрос пользователя следующего элемента



```kotlin
override fun next(): T {
    if (!hasNext()) throw NoSuchElementException()
    val currentState = state as State.WaitsWithElement<T>
    state = State.Waits(currentState.continuation)

}
```

Запрос пользователя следующего элемента

```
override fun next(): T {
    if (!hasNext()) throw NoSuchElementException()
    val currentState = state as State.WaitsWithElement<T>
    state = State.Waits(currentState.continuation)
    return currentState.element
}
```

```kotlin
private class IteratorBuilderMachine<T>(
    suspendableFunction: suspend IteratorBuilder<T>.() -> Unit
) : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {

    // ...

    var state: State<T> = State.Waits(suspendableFunction.createCoroutineUnintercepted(this, this))

    // ...

    override val context: CoroutineContext get() = EmptyCoroutineContext
}

fun <T> myIterator(builder: suspend IteratorBuilder<T>.() -> Unit): Iterator<T> =
    IteratorBuilderMachine(builder)
```

```kotlin
private class IteratorBuilderMachine<T>(
    suspendableFunction: suspend IteratorBuilder<T>.() → Unit
) : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {

    // ...

    var state: State<T> = State.Waits(suspendableFunction.createCoroutineUnintercepted(this, this))

    // ...

    override val context: CoroutineContext get() = EmptyCoroutineContext
}

fun <T> myIterator(builder: suspend IteratorBuilder<T>.() → Unit): Iterator<T> =
    IteratorBuilderMachine(builder)
```

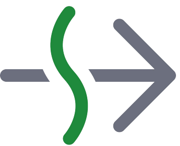
Контекст нам не нужен, возьмём пустой.

```kotlin
private class IteratorBuilderMachine<T>(
    suspendableFunction: suspend IteratorBuilder<T>.() → Unit
) : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {

    // ...

    var state: State<T> = State.Waits(suspendableFunction.createCoroutineUnintercepted(this, this))

    // ...

    override val context: CoroutineContext get() = EmptyCoroutineContext
}

fun <T> myIterator(builder: suspend IteratorBuilder<T>.() → Unit): Iterator<T> =
    IteratorBuilderMachine(builder)
```

```kotlin
private class IteratorBuilderMachine<T>(
    suspendableFunction: suspend IteratorBuilder<T>.() → Unit
) : IteratorBuilder<T>, Iterator<T>, Continuation<Unit> {

    // ...

    var state: State<T> = State.Waits(suspendableFunction.createCoroutineUnintercepted(this, this))

    // ...

    override val context: CoroutineContext get() = EmptyCoroutineContext
}

fun <T> myIterator(builder: suspend IteratorBuilder<T>.() → Unit): Iterator<T> =
    IteratorBuilderMachine(builder)
```

Это ресивер, чтобы подвешиваться и передавать элементы

Это Continuation, чтобы сообщить о конце работы лямбды

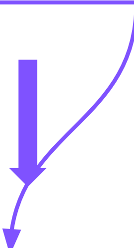


Нулевое приближение подкапотной корутин.

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```



```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Нужен, чтобы хранить:

1. «номер» точки подвешивания,

```
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Нужен, чтобы хранить:

1. «номер» точки подвешивания,
2. значения переменных подвешиваемой функции в промежуточных состояних, когда корутина подвешена,

```
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Нужен, чтобы хранить:

1. «номер» точки подвешивания,
2. значения переменных подвешиваемой функции в промежуточных состояних, когда корутина подвешена,
3. `Result`-значение, которое передаётся корутине для продолжения её работы,

```
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Нужен, чтобы хранить:

1. «номер» точки подвешивания,
2. значения переменных подвешиваемой функции в промежуточных состояниях, когда корутина подвешена,
3. `Result`-значение, которое передаётся корутине для продолжения её работы,
4. `Continuation`, который будет вызыван корутиной по завершении своей работы (completion).

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Также имплементирует `resumeWith` (от `Continuation`), который говорит, как продолжить своё выполнение.

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Нужна, чтобы:

1. если передана «внешняя» корутина `continuation`, обернуть в собственную и запустить её,

```
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Нужна, чтобы:

1. если передана «внешняя» корутина `continuation`, обернуть в собственную и запустить её,
2. если передана собственная корутина, запустить (продолжить выполнять) её до следующей точки остановки.

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Завершает свою работу одним следующих способов:

1. если изначальная функция завершает работу, кидая исключение, то тоже кидает то же исключение,

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Завершает свою работу одним следующих способов:

1. если изначальная функция завершает работу, кидая исключение, то тоже кидает то же исключение,
2. если изначальная функция завершает работу, возвращая значение, то тоже возвращает то же значение,

```kotlin
suspend fun callable(arg1: Arg1, arg2: Arg2, ...): Res
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, ..., continuation: Continuation<Res>): Res | COROUTINE_SUSPENDED

class CallableContinuation(val continuation: Continuation<Any?>) : Continuation<Any?>
```

Завершает свою работу одним следующих способов:

1. если изначальная функция завершает работу, кидая исключение, то тоже кидает то же исключение,
2. если изначальная функция завершает работу, возвращая значение, то тоже возвращает то же значение,
3. если изначальная функция подвешивается, то возвращает *COROUTINE_SUSPENDED*.

```kotlin
class CallableContinuation(val completion: Continuation<Any?>) : Continuation<Any?> {
    var var1: Any? = null
    var var2: Any? = null
    // ...

    var result: Result<Any?>? = null
    var label: Int = 0



    override val context: CoroutineContext get() = completion.context
    override fun resumeWith(result: Result<Any?>) { ... }
}
```

```kotlin
class CallableContinuation(val completion: Continuation<Any?>) : Continuation<Any?> {
    var var1: Any? = null
    var var2: Any? = null
    // ...

    var result: Result<Any?>? = null
    var label: Int = 0

    override val context: CoroutineContext get() = completion.context
    override fun resumeWith(result: Result<Any?>) { ... }
}
```

1. «Номер» точки подвешивания.

2. Значения переменных подвешиваемой функции в промежуточных состояних, когда корутина подвешена.

3. Result-значение, которое передаётся корутине для продолжения её работы.

4. Continuation, который будет вызыван корутиной по завершении своей работы (completion).

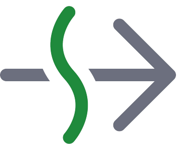
```
fun callable(arg1: Arg1, arg2: Arg2, continuation: Continuation<Res>): Any? {

}
```

```
fun callable(arg1: Arg1, arg2: Arg2, continuation: Continuation<Res>): Any? {
    val continuation: CallableContinuation = ...









}
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, continuation: Continuation<Res>): Any? {
    val continuation: CallableContinuation = ...

    var var1: Var1? = null
    var var2: Var2? = null
    // ...

}
```

```kotlin
fun callable(arg1: Arg1, arg2: Arg2, continuation: Continuation<Res>): Any? {
    val continuation: CallableContinuation = ...

    var var1: Var1? = null
    var var2: Var2? = null
    // ...

    while (true) {
        when (continuation.label) {
            // ...
            else -> error("call to 'resume' before 'invoke' with coroutine")
        }
    }
}
```

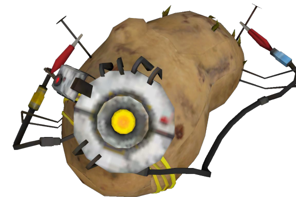
```kotlin
fun callable(arg1: Arg1, arg2: Arg2, continuation: Continuation<Res>): Any? {
    // ...
    while (true) {
        when (continuation.label) {
            // ...
            k -> {
                // <начало перехода>
                // <логика перехода>
                // <возможный конец перехода>
            }
            // ...
        }
    }
}
```



Это последний раздел,

где докладчик ~~вас добивает~~.

(делает выводы)



Выводы!

Вывод 1. Корутины — абстракция, представляющая машину состояний, которая является обычной кодогенерацией компилятора.

Выводы!

Вывод 1. Корутины — абстракция, представляющая машину состояний, которая является обычной кодогенерацией компилятора.

Вывод 2. Корутины сделаны для того, чтобы описывать асинхронную логику в обычном, синхронном стиле, позволяя интегрироваться с другими асинхронными API.

Конец! Вопросы?