Kotlin 的协程 Coroutines

扔物线（朱凯)
扔物线学堂

协程是什么？

在网上搜「协程是什么」

在网上搜「协程是什么」

- 「协程和线程类似，是一种在程序开发中处理多任务的组件。」
- 「协程就像一种轻量级的线程。」
- 「协程很像线程，但它不是线程。」
- 「协程是『用户态』的，它的切换不需要和操作系统交互，因此协程切换的成本比线程切换低。」
- 「协程由于是『协作式』的，所以不需要线程的同步操作。」

协程是什么

协程是什么

- 其实就是一套**由 Kotlin 官方提供的线程 API**。

```
// Thread
Thread {
  ...
}.start()
```

```
// Thread
Thread {
  ...
}.start()

// Executor
val executor = Executors.newCachedThreadPool()
executor.execute {
  ...
}
```

```
// Thread
Thread {
   ...
}.start()

// Executor
val executor = Executors.newCachedThreadPool()
executor.execute {
   ...
}

// 协程
launch {
   ...
}
```

协程好在哪?

协程好在哪?

- 方便

```
...
val user = api.getUser()
nameTv.text = user.name
...
```

```
...
val user = api.getUser() // 网络请求（后台线程）
nameTv.text = user.name // 更新 UI（主线程）
...
```

「非阻塞式挂起」

「非阻塞式挂起」

协程长什么样

```
launch(Dispatchers.IO) {
  saveToDatabase(data)
}
```

```
launch(Dispatchers.IO) {
  saveToDatabase(data)
}

launch(Dispatchers.Main) {
  updateViews(data)
}
```

```
launch(Dispatchers.IO) {
  saveToDatabase(data)
}

launch(Dispatchers.Main) {
  updateViews(data)
}

thread {
  ...
}
```

```
launch(Dispatchers.Main) {
  val user = api.getUser()
  nameTv.text = user.name
}
```

```
launch(Dispatchers.Main) { // 开始：主线程
  val user = api.getUser() // 网络请求：后台线程
  nameTv.text = user.name // 更新 UI：主线程
}
```

```
launch(Dispatchers.Main) { // 开始：主线程
  val user = api.getUser() // 网络请求：后台线程
  nameTv.text = user.name // 更新 UI：主线程
}
```

```
launch(Dispatchers.Main) { // 开始：主线程
  val user = api.getUser() // 网络请求：后台线程
  nameTv.text = user.name // 更新 UI：主线程
}

api.getUser()
    .enqueue(object : Callback<User> {
      ...

      override fun onResponse(call: Call<User>,
                              response: Response<User>) {
        runOnUiThread {
          nameTv.text = response.body()?.name
        }
      }
    })
```

```
launch(Dispatchers.Main) { // 开始：主线程
  val user = api.getUser() // 网络请求：后台线程
  nameTv.text = user.name // 更新 UI：主线程
}

api.getUser()
    .enqueue(object : Callback<User> {
      ...

      override fun onResponse(call: Call<User>,
                              response: Response<User>) {
        runOnUiThread {
          nameTv.text = response.body()?.name
        }
      }
    })
```

```
launch(Dispatchers.Main) {
  val token = api.getToken()
  val user = api.getUser(token)
  nameTv.text = user.name
}
```

```
launch(Dispatchers.Main) { // 开始：主线程
  val token = api.getToken() // 网络请求：后台线程
  val user = api.getUser(token) // 网络请求：后台线程
  nameTv.text = user.name // 更新 UI：主线程
}
```

```
api.getToken()
    .enqueue(object : Callback<Token> {
      ...

      override fun onResponse(call: Call<Token>,
                              response: Response<Token>) {
        api.getUser()
            .enqueue(object : Callback<User> {
              ...

              override fun onResponse(call: Call<User>,
                                      response: Response<User>) {
                runOnUiThread {
                  nameTv.text = response.body()?.name
                }
              }
            })
      }
    })
```

```
api.getToken()
    .enqueue(object : Callback<Token> {
      ...

      override fun onResponse(call: Call<Token>,
                              response: Response<Token>) {
        api.getUser()
            .enqueue(object : Callback<User> {
              ...

              override fun onResponse(call: Call<User>,
                                      response: Response<User>) {
                runOnUiThread {
                  nameTv.text = response.body()?.name
                }
              }
            })
      }
    })
```

协程的「1 到 0」

```
// API 1: 获取用户头像
api.getAvatar(user)
// API 2: 获取用户所在公司的 logo
api.getCompanyLogo(user)
```

```
launch(Dispatchers.Main) {
  val avatar = async { api.getAvatar(user) } // 获取用户头像
  val logo = async { api.getCompanyLogo(user) } // 获取公司的 logo
  val merged = suspendingMerge(avatar, logo) // 合并
  show(merged) // 显示
}
```

协程怎么用

```
launch(Dispatchers.IO) {
  val image = getImage(imageId)
}
```

```
launch(Dispatchers.IO) {
  val image = getImage(imageId)
}
```

```
launch(Dispatchers.IO) {
  val image = getImage(imageId)
}
```

```
launch(Dispatchers.IO) {
  val image = getImage(imageId)
  launch(Dispatchers.Main) {
    avatarIv.setImageBitmap(image)
  }
}
```

```
launch(Dispatchers.IO) {
  val image = getImage(imageId)
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.Main) {
  val image = getImage(imageId)
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.Main) {
  val image = withContext(Dispatchers.IO) {
    getImage(imageId)
  }
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.IO) {
  ...
  launch(Dispatchers.Main) {
    ...
    launch(Dispatchers.IO) {
      ...
      launch(Dispatchers.Main) {
        ...
      }
    }
  }
}
```

```
launch(Dispatchers.Main) {
  withContext(Dispatchers.IO) {
    ...
  }
  ...
  withContext(Dispatchers.IO) {
    ...
  }
  ...
}
```

```
launch(Dispatchers.Main) {
  val image = withContext(Dispatchers.IO) {
    getImage(imageId)
  }
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.Main) {
  val image =


  avatarIv.setImageBitmap(image)
}

fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

```
launch(Dispatchers.Main) {
  val image = spendingGetImage(imageId)
  avatarIv.setImageBitmap(image)
}

fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

```
launch(Dispatchers.Main) {
    val image = spendingGetImage(imageId)
    avatarIv.setImageBitmap(image)
}

fun suspendingGetImage(imageId: String) {
    withContext(Dispatchers.IO) {
        getImage(imageId)
    }
}
```

Suspend function 'withContext' should be called only from a coroutine or another suspend function

```
public suspend fun <T> withContext(
    context: CoroutineContext,
    block: suspend CoroutineScope.() -> T
): T = ...
```

```
public suspend fun <T> withContext(
    context: CoroutineContext,
    block: suspend CoroutineScope.() -> T
): T = ...
```

```
launch(Dispatchers.Main) {
    val image = spendingGetImage(imageId)
    avatarIv.setImageBitmap(image)
}

fun suspendingGetImage(imageId: String) {
    withContext(Dispatchers.IO) {
        getImage(imageId)
    }
}
```

Suspend function 'withContext' should be called only from a coroutine or another suspend function

```
launch(Dispatchers.Main) {
    val image = spendingGetImage(imageId)
    avatarIv.setImageBitmap(image)
}

suspend fun suspendingGetImage(imageId: String) {
    withContext(Dispatchers.IO) {
        getImage(imageId)
    }
}
```

Suspend function 'withContext' should be called only from a coroutine or another suspend function

```
launch(Dispatchers.Main) {
  val image = spendingGetImage(imageId)
  avatarIv.setImageBitmap(image)
}

suspend fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

「挂起」的本质

```
launch(Dispatchers.Main) {
  val image = spendingGetImage(imageId)
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.Main) {
  val image = spendingGetImage(imageId)
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.Main) {
  val image = spendingGetImage(imageId)
  avatarIv.setImageBitmap(image)
}
```

```
launch(Dispatchers.Main) {
    val image = spendingGetImage(imageId)
    avatarIv.setImageBitmap(image)
}
```

「挂起」的本质

「挂起」的本质

「挂起」的本质

- 协程从线程脱离

「挂起」的本质

- 协程从线程脱离
  - 线程：该干嘛干嘛去

「挂起」的本质

- 协程从线程脱离
  - 线程：该干嘛干嘛去
  - 协程：换个线程继续工作

「挂起」的本质

- 协程从线程脱离
  - 线程：该干嘛干嘛去
  - 协程：换个线程继续工作，然后……再切回来

怎么就「挂起」了？

```
suspend fun suspendingPrint() {
  println("Thread: ${Thread.currentThread().name}")
}
```

```
suspend fun suspendingPrint() {
  println("Thread: ${Thread.currentThread().name}")
}
```

```
I/System.out: Thread: main
```

```
suspend fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

```
suspend fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

「suspend」的意义?

```
suspend fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

```
suspend fun suspendingGetImage(imageId: String) {
  withContext(Dispatchers.IO) {
    getImage(imageId)
  }
}
```

```
suspend fun suspendingPrint() {
  println("Thread: ${Thread.currentThread().name}")
}
```

```
I/System.out: Thread: main
```

```
suspend fun suspendingPrint() {
  println("Thread: ${Thread.currentThread().name}")
}
```

```
I/System.out: Thread: main
```

怎么自定义 suspend 函数?

怎么自定义 suspend 函数?

- 什么时候自定义?

怎么自定义 suspend 函数?

- 什么时候自定义?
- 怎么写?

其他 suspend 函数?

协程的「非阻塞式」挂起

协程的「非阻塞式」挂起

- 非阻塞式：指的是不卡线程

协程的「非阻塞式」挂起

- 非阻塞式：指的是不卡线程
  - 那……线程是非阻塞式吗?

协程的「非阻塞式」挂起

- 非阻塞式：指的是不卡线程
  - 那……线程是非阻塞式吗?
- 有人说：「协程的挂起是非阻塞式的，而线程是阻塞式的」

协程的「非阻塞式」挂起

- 非阻塞式：指的是不卡线程
  - 那……线程是非阻塞式吗?
- 有人说：「协程的挂起是非阻塞式的，而线程是阻塞式的」
- 有人说：「协程的这个『非阻塞式』比线程更高效」

协程的「非阻塞式」挂起

- 非阻塞式：指的是不卡线程
  - 那……线程是非阻塞式吗?
- 有人说：「协程的挂起是非阻塞式的，而线程是阻塞式的」
- 有人说：「协程的这个『非阻塞式』比线程更高效」
- 还有人说：「协程是『用户态』的，切换的成本比线程低」

协程的「非阻塞式」挂起

- 非阻塞式：指的是不卡线程
  - 那……线程是非阻塞式吗?
- 有人说：「协程的挂起是非阻塞式的，而线程是阻塞式的」
- 有人说：「协程的这个『非阻塞式』比线程更高效」
- 还有人说：「协程是『用户态』的，切换的成本比线程低」
- 还有人说：「协程由于是『协作式』的，所以不需要线程的同步操作」

还有一个公司说……

高级进阶，没那么难

——扔物线学堂

plus.hencoder.com