kotlin协程

协程创建

- 第一步:创建gradle工程
- 第二步:添加依赖

  compile 'org.jetbrains.kotlinx:kotlinx-coroutines-core:0.22.5'
- 第三步:创建协程

```
fun main(args: Array<String>) {
    println("主线程开始")
    //开启协程
    launch { this: CoroutineScope
        println("协程代码")
    }
    println("主线程结束")
    Thread.sleep( millis: 2000L)
}
```

launch函数分析

```
public actual fun launch(
    context: CoroutineContext = DefaultDispatcher,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    parent: Job? = null,
    block: suspend CoroutineScope.() -> Unit
): Job {
```

协程上下文

协程返回值

CommonPool

- 默认的协程上下文,通过ForkJoinPool*实现*

ForkJoinPool

- 开启的线程都是守护线程

Job

- 返回的协程引用

协程启动的处理

```
fun main(args: Array<String>) {
    println("主线程开始")
    //开启协程
    val job : Job = launch { this: CoroutineScope
        println("协程代码")
    }
    println("主线程结束")
    //第一种:可以让主线程睡眠
    Thread.sleep( millis: 2000L)
}
```

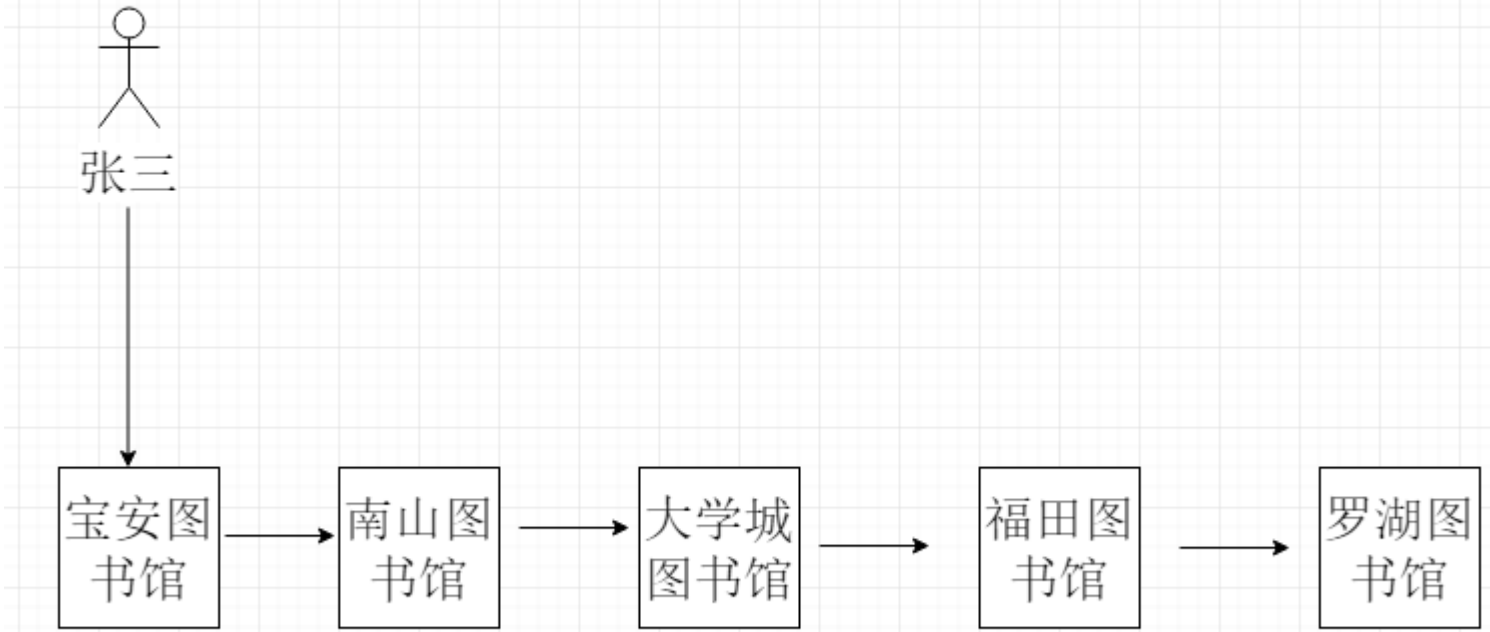
协程启动的处理

```
fun main(args: Array<String>) : Unit = runBlocking{ this: CoroutineScope
    println("主线程开始")
    //开启协程
    val job : Job = launch { this: CoroutineScope
        println("协程代码")
    }
    println("主线程结束")
    //第二种:加入到主线程中
    job.join()
}
```

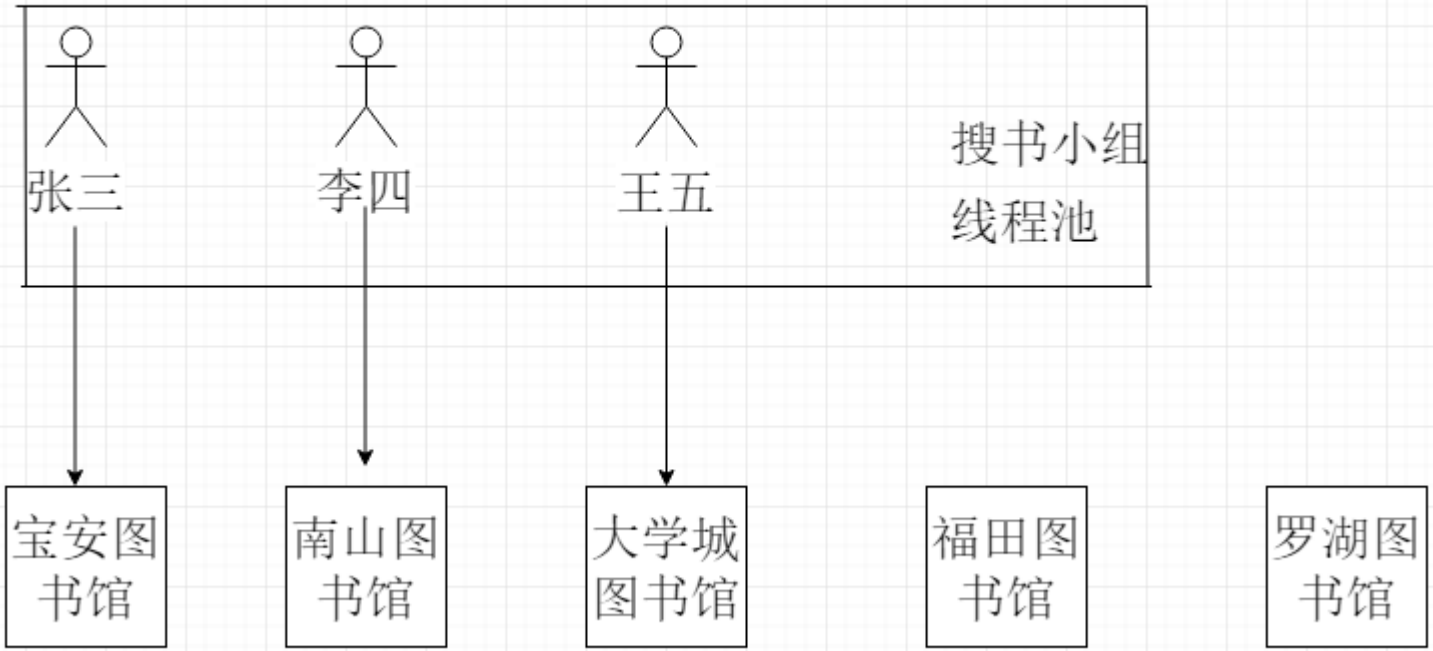
协程原理和优势

- 同步和异步
- 阻塞和非阻塞

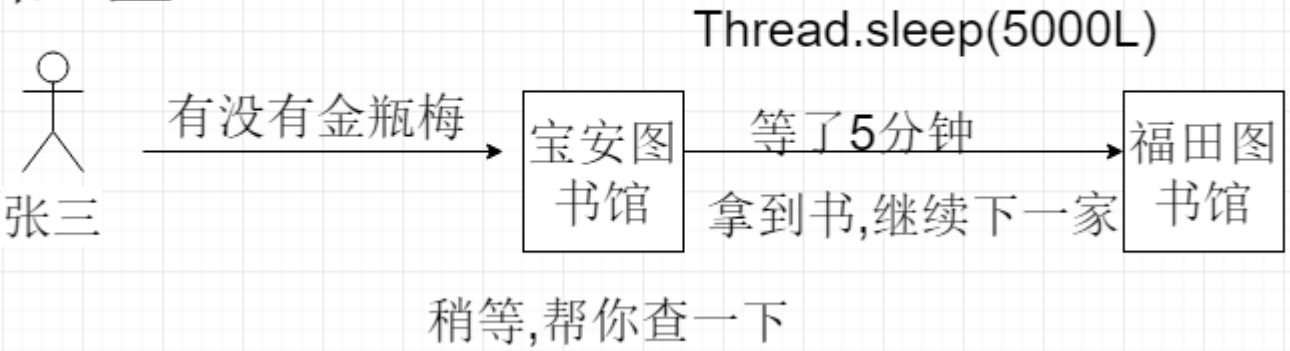
同步



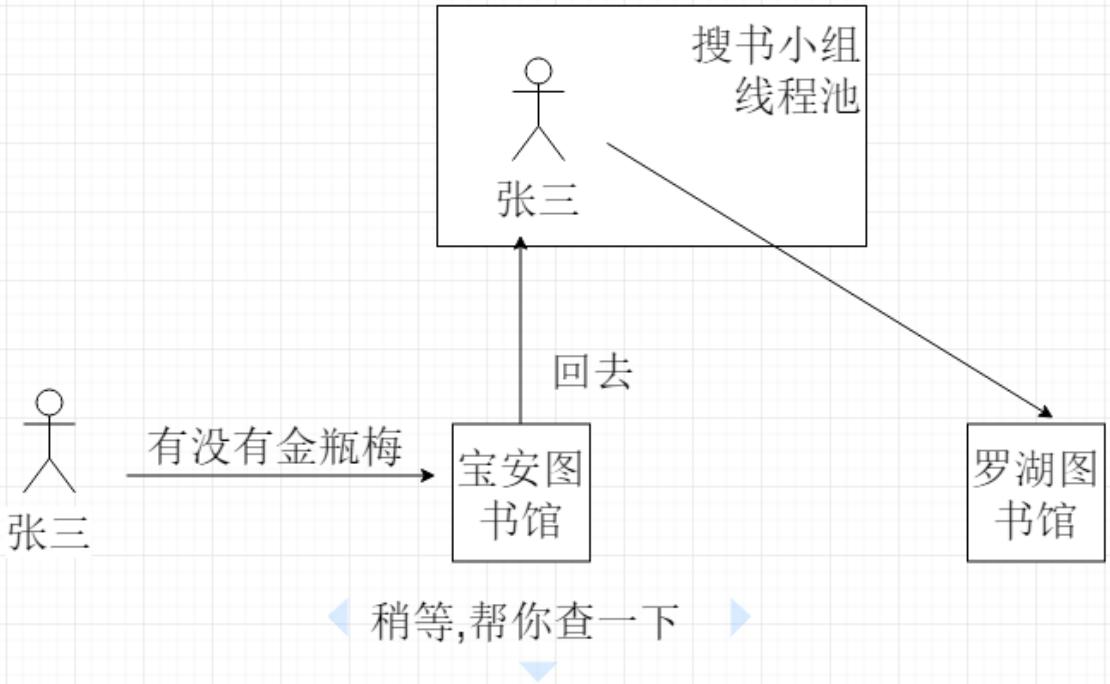
异步



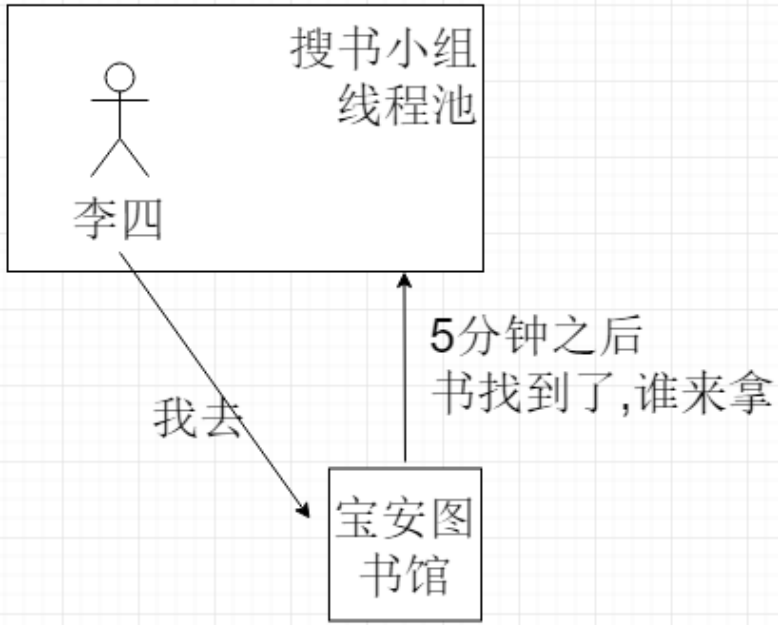
阻塞


Thread.sleep(5000L)
有没有金瓶梅
宝安图书馆
等了5分钟
福田图书馆
张三
拿到书,继续下一家
稍等,帮你查一下

非阻塞



非阻塞



协程原理

- 可以把耗时任务先挂起
- 等时间到了再从线程池中空闲的线程执行
- 必须是挂起函数才能挂起

挂起函数

- 挂起函数必须在协程代码中才能调用
- 挂起函数中也可以调用挂起函数

主协程

```
fun main(args: Array<String>) : Unit  = runBlocking{ this: CoroutineScope
    println("主线程执行前")
    delay( time: 2000L)
    println("主线程执行后")
}
```

协程和线程效率对比

```
val threadList : List<Thread> = List( size: 100000) { it: Int
    Thread(Runnable {
        println(".")
    })
}
val startTime : Long = System.currentTimeMillis()
threadList.forEach{ it: Thread
    it.start()
}
val endTime : Long = System.currentTimeMillis()
println("使用时间:${endTime-startTime}")
```

协程和线程效率对比

```
val coroutineList : List<Job>  = List( size: 100000) { it: Int
    launch { this: CoroutineScope
        println(".")
    }
}
val startTime : Long  = System.currentTimeMillis()
coroutineList.forEach { it: Job
    it.join()
}
val endTime : Long  = System.currentTimeMillis()
println("用时:${endTime-startTime}")//606
```

协程取消

```
val job : Job = launch { this: CoroutineScope
    while(true){
        println("第及 次")
        delay( time: 500L)
    }
}

delay( time: 2000L)
println("主线程执行")

job.cancel()

job.join()
```

协程定时取消

```
val job : Job = launch { this: CoroutineScope
    withTimeout( time: 2000) { this: CoroutineScope
        while (true) {
            println("正在执行")
            delay( time: 500L)
        }
    }
}

delay( time: 2000L)

job.join()
```

协程取消失效

```
val job : Job = launch { this: CoroutineScope
    while (true){
        println("协程执行了")
        Thread.sleep( millis: 500L)
    }
}
Thread.sleep( millis: 2000L)
//取消协程
job.cancel()

job.join()
```

协程取消前后状态的变化

```
val job : Job = launch { this: CoroutineScope
    while (true){
        println("协程执行了")
        Thread.sleep( millis: 500L)
    }
}
Thread.sleep( millis: 2000L)
println("协程取消前:${job.isActive}")
//取消协程
job.cancel()
println("协程取消后:${job.isActive}")

job.join()
```

协程取消失效的解决

```
val job : Job = launch { this: CoroutineScope
    while (true){
        if(!isActive)return@launch
        println("协程执行了")
        Thread.sleep( millis: 500L)
    }
}
Thread.sleep( millis: 2000L)
println("协程取消前:${job.isActive}")
//取消协程
job.cancel()
println("协程取消后:${job.isActive}")

job.join()
```

协程启动async

- launch启动协程,不能获取协程执行的结果
- async可以获取协程中执行的结果

协程上下文

- Unconfined:无限制运行在主线程中
- coroutineContext:使用父协程的上下文
- CommonPool:默认就是CommonPool
- 自定义线程池上下文: newFixedThreadPoolContext