# 并发任务编排的实现与设计

以 Java CompletableFuture 为例

尺规 @oldratlee

2025-10

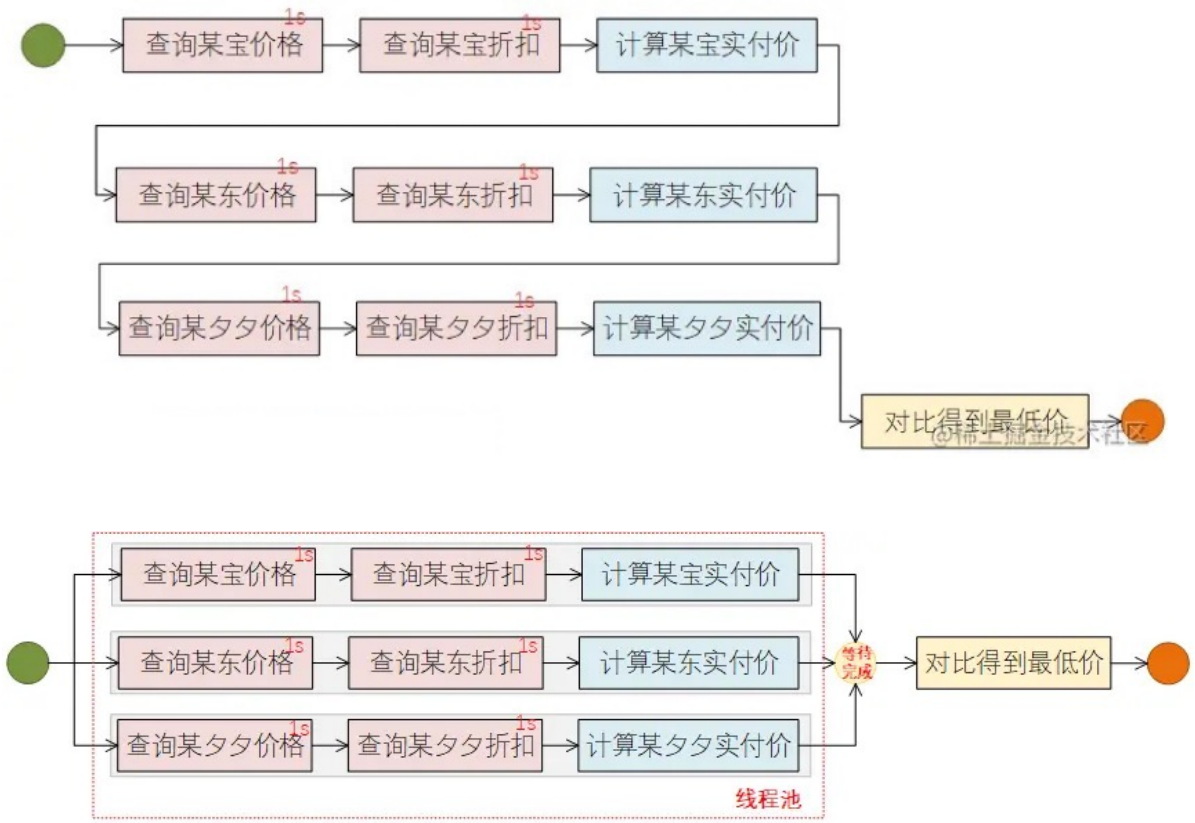
# 0.0 通过 并发编排 缩短RT 的例子

**串行**

等待 **6s**

**并行**

等待 **2s**



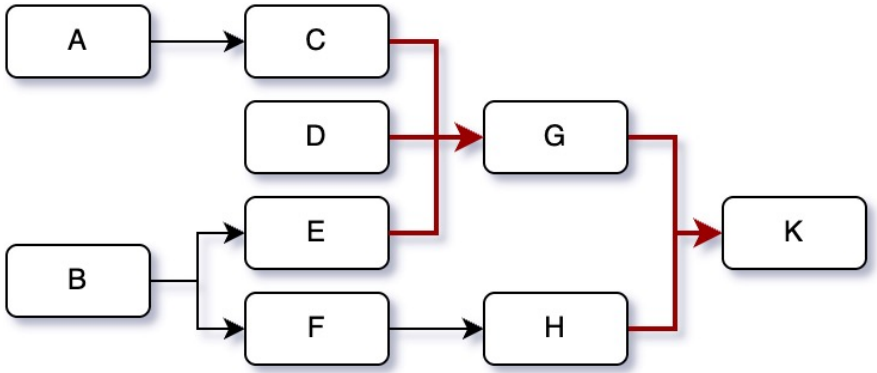

图片来源：Java 基于 CompletableFuture 的流水线并行处理深度实践

juejin.cn/post/7124124854747398175

# 0.1 关于编排

这里讨论的 编排 指:

- 协调多个(并发/异步)任务
  - 任务可以有 执行逻辑 与 返回结果
- 然后触发其它的任务
  - 可以有不同 触发/编排策略




- **简单编排**，即：一个任务 触发 其它(不同)任务，不作讨论
  - 因为 理解与使用 都很简单自然
- 不同说法如「**异步**（任务）编排」「**并发**（任务）编排」等，不作区分

```
CompletableFuture<String> cf1 = CF.supplyAsync(() -> "1");
CompletableFuture<String> cf2 = CF.supplyAsync(() -> {throw new Exception("2");});
CompletableFuture<String> cf3 = CF.supplyAsync(() -> "3");

CompletableFuture<Void> all = CF.allOf(cf1, cf2, cf3);


all.join();
println(cf1.join() + cf2.join() + cf3.join());
```

# 0.2 关于编排策略

标准库 CompletableFuture 提供了两种策略：

- All(Complete)
- Any(Complete)

```
// All(Complete)
CompletableFuture<Void> all = CF.allOf(cf1, cf2, cf3);
CompletableFuture<String> both = cf1.thenCombine(cf2, (x, y) -> x + y);
...
// Any(Complete)
CompletableFuture<String> any = CF.anyOf(cf1, cf2, cf3);
CompletableFuture<String> either = cf1.applyToEither(cf2, x -> x + "!");
...
```

- AllComplete vs AllFailFast
  - AllComplete 会等待所有输入 CF 运行完成；
    即使有 CF 失败了也要等待后续 CF 都运行完成，再返回一个失败的 CF。
  - 对于业务逻辑来说，失败且继续等待的策略，降低了业务响应性。
  - 业务需要的是，当有输入 CF 失败了则快速失败不再做于事无补的等待。
- AnyComplete vs AnySuccess
  - AnyComplete 返回首个完成的 CF，不会等待后续没有完成的 CF；
    即使首个完成的 CF 是失败的，也会返回这个失败的 CF 结果。
  - 业务逻辑一般需要的是首个成功的 CF 结果，而不是首个完成但**失败**的 CF。

关于 AllFailFast / AnySuccess 的更多介绍与讨论：

CompletableFuture 如何实现异步任务编排中**最常用的**模式 —— 快速失败
juejin.cn/post/7420597224546091059

# 1.A 如何实现 AllFailFast？

用什么并发工具都可以，比如使用

任务执行：

- **Thread** (new / Run )
- **ThreadPool** (execute / submit)
- **CompletableFuture**
- ...

任务协调：

- **CountDownLatch**
- **Future** (f.get / f.cancel)
- **CompletableFuture**
- ...

还有 如何实现 AnySuccess？

类似，讨论一者就好

API 与实现 可以与工具 CF 没有关系；实现 AllFailFast 功能效果即可

对于基于 CF 的实现，这个功能的方法签名会是：

```
public static <T> CompletableFuture<List<T>>
allResultsFailFastOf(CompletableFuture<? extends T>... cfs);
```

- AllFailFast
  - 当有输入 CF 失败了，则快速失败
- AllComplete
  - 等待所有输入 CF 运行完成
  - 即使有 CF 失败了也要等待后续CF都运行完成
- 两者都是，仅当所有的输入都成功时，才返回成功的结果

# 1.B 如何通过 CF.allOf + CF.anyOf 实现 AllFailFast？

实现上可以假定 CF 有了方法 AllResultsOf，会返回多个输入的结果 (List)：

```
public static <T> CompletableFuture<List<T>>
allResultsOf(CompletableFuture<? extends T>... cfs);
```

而不像 CF.allOf 方法没有返回结果 (Void)：

```
public static CompletableFuture<Void>
allOf(CompletableFuture<?>... cfs);
```

**先自己想一下？** 🤔

参见 cffu 库的代码实现：
CompletableFutureUtils.java#allFailFastOf0()

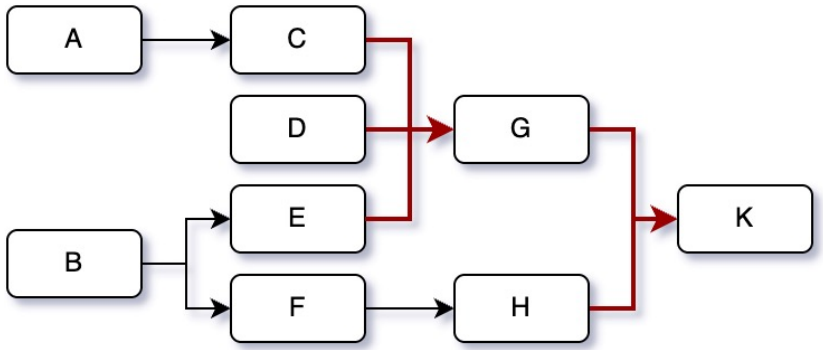
- AllFailFast
  - 当有输入 CF 失败了，则快速失败
- AllComplete
  - 等待所有输入 CF 运行完成
  - 即使有 CF 失败了也要等待后续CF都运行完成
- 两者都是，仅当所有的输入都成功时，才返回成功的结果

# 2. 还有 哪些 编排策略？

前面提到了 4种编排策略:

- AllComplete vs AllFailFast
- AnyComplete vs AnySuccess

还有哪些？只有哪些？




## 先自己想一下？🤔

参见 cffu 库的文档：高效灵活的并发执行策略

# 3. 并发编排实现设计的注意点（以 AllFailFast 为例）

- 即时触发
  - 时序要实现的基本功能，否则是 Bug
- 不吞异常，方便排查业务问题
  - 当多个输入抛出多个异常时，这些异常至多只能有一个能通过返回反馈给业务
  - 其它的异常不能被默默地吞掉，要报告如打日志出来，否则影响业务问题的排查
- 尽快释放内存
  - AllFailFast 会返回(持有)多个输入任务的结果；当有输入任务失败时，所有结果不再需要立即释放
  - 避免因有长时间运行的任务而持有其它任务的(大)结果对象
- 当输入0个时，如何返回、返回什么？
  - 关注好 平凡情况、边界 Case；设计一致性、实现可靠性

这些注意点可以看看 cffu 库的文档 或 源码，有讨论与答案：

- github.com/foldright/cffu/.../docs/README_CN.md
- github.com/foldright/cffu/.../cffu-core/src/main/java/io/foldright/cffu2/CompletableFutureUtils.java

# 参考资料

- Java 基于 CompletableFuture 的流水线并行处理深度实践
  juejin.cn/post/7124124854747398175

- CompletableFuture 如何实现异步任务编排中最常用的模式——快速失败
  juejin.cn/post/7420597224546091059

- 🦡 Java CompletableFuture-Fu ("CF-Fu", pronounced "Shifu") is a tiny 0-dependency library that improves the CompletableFuture (CF) usage experience and reduces misuse, enabling more convenient, efficient, and safe use of CF in your application. 😋🚀🦺
  github.com/foldright/cffu

# Thanks

# &

# QA