

10 | 如何使用异步设计提升系统性能？

2019-08-13 李玥

消息队列高手课

[进入课程 >](#)



讲述：李玥

时长 12:36 大小 11.55M



你好，我是李玥，这一讲我们来聊一聊异步。

对于开发者来说，异步是一种程序设计的思想，使用异步模式设计的程序可以显著减少线程等待，从而在高吞吐量的场景中，极大提升系统的整体性能，显著降低时延。

因此，像消息队列这种需要超高吞吐量和超低时延的中间件系统，在其核心流程中，一定会大量采用异步的设计思想。

接下来，我们一起来通过一个非常简单的例子学习一下，使用异步设计是如何提升系统性能的。

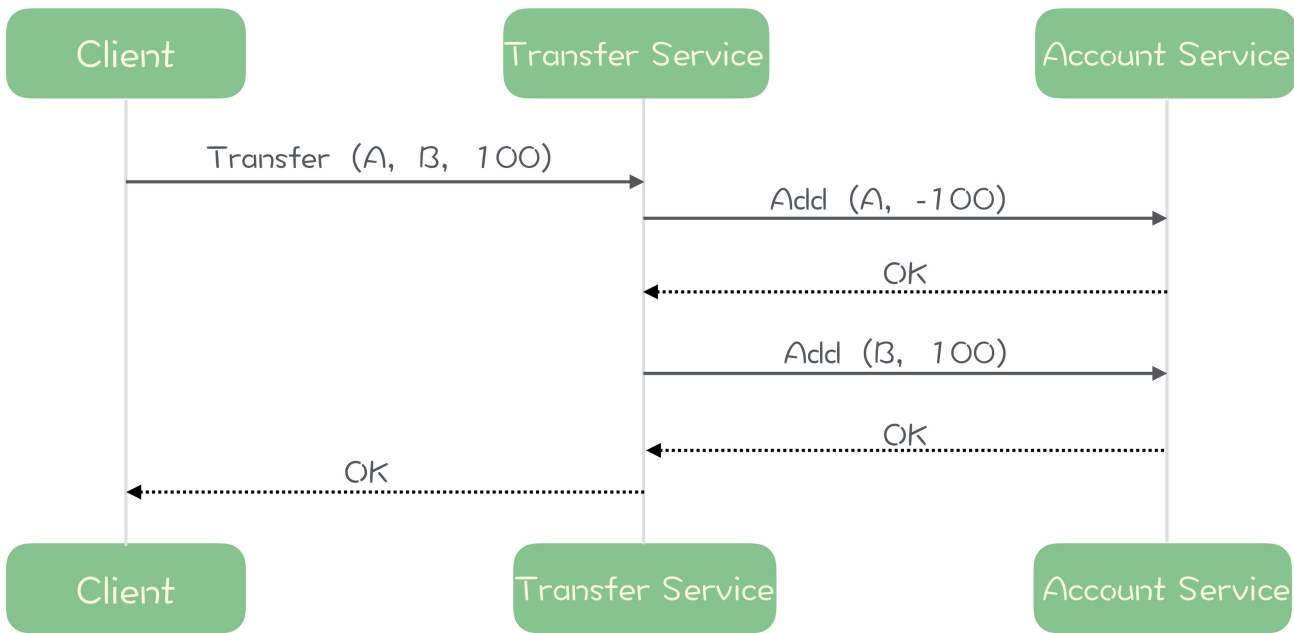
异步设计如何提升系统性能？

假设我们要实现一个转账的微服务 `Transfer(accountFrom, accountTo, amount)`，这个服务有三个参数：分别是转出账户、转入账户和转账金额。

实现过程也比较简单，我们要从账户 A 中转账 100 元到账户 B 中：

- 1. 先从 A 的账户中减去 100 元；
- 2. 再给 B 的账户加上 100 元，转账完成。

对应的时序图是这样的：



在这个例子的实现过程中，我们调用了另外一个微服务 `Add(account, amount)`，它的功能是给账户 `account` 增加金额 `amount`，当 `amount` 为负值的时候，就是扣减响应的金额。

需要特别说明的是，在这段代码中，我为了使问题简化以便我们能专注于异步和性能优化，省略了错误处理和事务相关的代码，你在实际的开发中不要这样做。

1. 同步实现的性能瓶颈

首先我们来看一下同步实现，对应的伪代码如下：

```
1 Transfer(accountFrom, accountTo, amount) {
2     // 先从 accountFrom 的账户中减去相应的钱数
```

```
3    Add(accountFrom, -1 * amount)
4    // 再把减去的钱数加到 accountTo 的账户中
5    Add(accountTo, amount)
6    return OK
7 }
```

上面的伪代码首先从 accountFrom 的账户中减去相应的钱数，再把减去的钱数加到 accountTo 的账户中，这种同步实现是一种很自然方式，简单直接。那么性能表现如何呢？接下来我们就来一起分析一下性能。

假设微服务 Add 的平均响应时延是 50ms，那么很容易计算出我们实现的微服务 Transfer 的平均响应时延大约等于执行 2 次 Add 的时延，也就是 100ms。那随着调用 Transfer 服务的请求越来越多，会出现什么情况呢？

在这种实现中，每处理一个请求需要耗时 100ms，并在这 100ms 过程中是需要独占一个线程的，那么可以得出这样一个结论：每个线程每秒钟最多可以处理 10 个请求。我们知道，每台计算机上的线程资源并不是无限的，假设我们使用的服务器同时打开的线程数量上限是 10,000，可以计算出这台服务器每秒钟可以处理的请求上限是： $10,000 \text{ (个线程)} * 10 \text{ (次请求每秒)} = 100,000 \text{ 次每秒}$ 。

如果请求速度超过这个值，那么请求就不能被马上处理，只能阻塞或者排队，这时候 Transfer 服务的响应时延由 100ms 延长到了：排队的等待时延 + 处理时延 (100ms)。也就是说，在大量请求的情况下，我们的微服务的平均响应时延变长了。


这是不是已经到了这台服务器所能承受的极限了呢？其实远远没有，如果我们监测一下服务器的各项指标，会发现无论是 CPU、内存，还是网卡流量或者是磁盘的 IO 都空闲的很，那我们 Transfer 服务中的那 10,000 个线程在干什么呢？对，绝大部分线程都在等待 Add 服务返回结果。

也就是说，采用同步实现的方式，整个服务器的所有线程大部分时间都没有在工作，而是都在等待。

如果我们能减少或者避免这种无意义的等待，就可以大幅提升服务的吞吐能力，从而提升服务的总体性能。

2. 采用异步实现解决等待问题

接下来我们看一下，如何用异步的思想来解决这个问题，实现同样的业务逻辑。

 复制代码

```
1 TransferAsync(accountFrom, accountTo, amount, OnComplete()) {
2     // 异步从 accountFrom 的账户中减去相应的钱数，然后调用 OnDebit 方法。
3     AddAsync(accountFrom, -1 * amount, OnDebit(accountTo, amount, OnAllDone(OnComplete())))
4 }
5 // 扣减账户 accountFrom 完成后调用
6 OnDebit(accountTo, amount, OnAllDone(OnComplete())) {
7     // 再异步把减去的钱数加到 accountTo 的账户中，然后执行 OnAllDone 方法
8     AddAsync(accountTo, amount, OnAllDone(OnComplete()))
9 }
10 // 转入账户 accountTo 完成后调用
11 OnAllDone(OnComplete()) {
12     OnComplete()
13 }
```

细心的你可能已经注意到了，TransferAsync 服务比 Transfer 多了一个参数，并且这个参数传入的是一个回调方法 OnComplete()（虽然 Java 语言并不支持将方法作为方法参数传递，但像 JavaScript 等很多语言都具有这样的特性，在 Java 语言中，也可以通过传入一个回调类的实例来变相实现类似的功能）。

这个 TransferAsync() 方法的语义是：请帮我执行转账操作，当转账完成后，请调用 OnComplete() 方法。调用 TransferAsync 的线程不必等待转账完成就可以立即返回了，待转账结束后，TransferService 自然会调用 OnComplete() 方法来执行转账后续的工作。

异步的实现过程相对于同步来说，稍微有些复杂。我们先定义 2 个回调方法：

OnDebit()：扣减账户 accountFrom 完成后调用的回调方法；

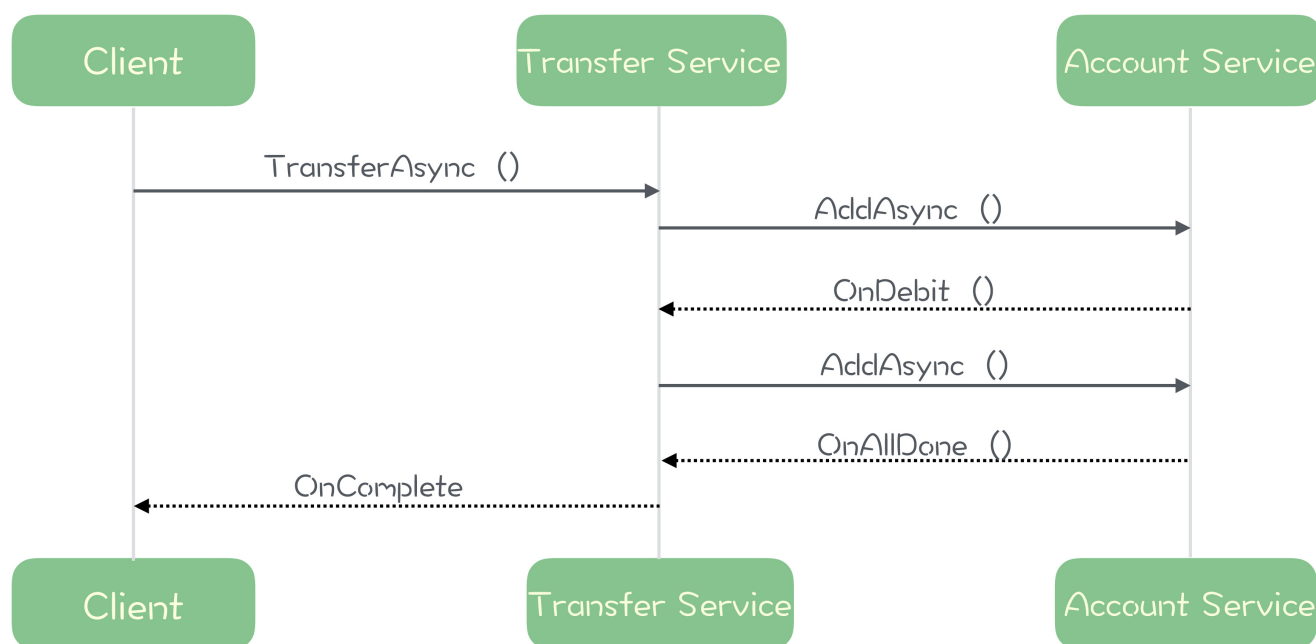
OnAllDone()：转入账户 accountTo 完成后调用的回调方法。

整个异步实现的语义相当于：

1. 异步从 accountFrom 的账户中减去相应的钱数，然后调用 OnDebit 方法；
2. 在 OnDebit 方法中，异步把减去的钱数加到 accountTo 的账户中，然后执行 OnAllDone 方法；

3. 在 OnAllDone 方法中，调用 OnComplete 方法。

绘制成时序图是这样的：



你会发现，异步化实现后，整个流程的时序和同步实现是完全一样的，**区别只是在线程模型上由同步顺序调用改为了异步调用和回调的机制。**

接下来我们分析一下异步实现的性能，由于流程的时序和同步实现是一样，在低请求数量的场景下，平均响应时延一样是 100ms。在超高请求数量场景下，异步的实现不再需要线程等待执行结果，只需要个位数量的线程，即可实现同步场景大量线程一样的吞吐量。

由于没有了线程的数量的限制，总体吞吐量上限会大大超过同步实现，并且在服务器 CPU、网络带宽资源达到极限之前，响应时延不会随着请求数量增加而显著升高，几乎可以一直保持约 100ms 的平均响应时延。

看，这就是异步的魔力。


简单实用的异步框架: CompletableFuture

在实际开发时，我们可以使用异步框架和响应式框架，来解决一些通用的异步编程问题，简化开发。Java 中比较常用的异步框架有 Java8 内置的 [CompletableFuture](#) 和 ReactiveX 的 [RxJava](#)，我个人比较喜欢简单实用易于理解的 CompletableFuture，但是 RxJava 的功能更加强大。有兴趣的同学可以深入了解一下。


Java 8 中新增了一个非常强大的用于异步编程的类：CompletableFuture，几乎囊括了我们在开发异步程序的大部分功能，使用 CompletableFuture 很容易编写出优雅且易于维护的异步代码。

接下来，我们来看下，如何用 CompletableFuture 实现的转账服务。

首先，我们用 CompletableFuture 定义 2 个微服务的接口：

 复制代码


```
1 /**
2  * 账户服务
3  */
4 public interface AccountService {
5     /**
6      * 变更账户金额
7      * @param account 账户 ID
8      * @param amount 增加的金额，负值为减少
9      */
10    CompletableFuture<Void> add(int account, int amount);
11 }
```

 复制代码

```
1 /**
2  * 转账服务
3  */
4 public interface TransferService {
5     /**
6      * 异步转账服务
7      * @param fromAccount 转出账户
8      * @param toAccount 转入账户
9      * @param amount 转账金额，单位分
10     */
11    CompletableFuture<Void> transfer(int fromAccount, int toAccount, int amount);
12 }
```

可以看到这两个接口中定义的方法的返回类型都是一个带泛型的 CompletableFuture，尖括号中的泛型类型就是真正方法需要返回数据的类型，我们这两个服务不需要返回数据，所以直接用 Void 类型就可以。

然后我们来实现转账服务：


 复制代码

```
1  /**
2   * 转账服务的实现
3   */
4  public class TransferServiceImpl implements TransferService {
5      @Inject
6      private AccountService accountService; // 使用依赖注入获取账户服务的实例
7      @Override
8      public CompletableFuture<Void> transfer(int fromAccount, int toAccount, int amount)
9          // 异步调用 add 方法从 fromAccount 扣减相应金额
10         return accountService.add(fromAccount, -1 * amount)
11         // 然后调用 add 方法给 toAccount 增加相应金额
12         .thenCompose(v -> accountService.add(toAccount, amount));
13     }
14 }
```

在转账服务的实现类 `TransferServiceImpl` 里面，先定义一个 `AccountService` 实例，这个实例从外部注入进来，至于怎么注入不是我们关心的问题，就假设这个实例是可用的就好了。

然后我们看实现 `transfer()` 方法的实现，我们先调用一次账户服务 `accountService.add()` 方法从 `fromAccount` 扣减响应的金额，因为 `add()` 方法返回的就是一个 `CompletableFuture` 对象，可以用 `CompletableFuture` 的 `thenCompose()` 方法将下一次调用 `accountService.add()` 串联起来，实现异步依次调用两次账户服务完整转账。

客户端使用 `CompletableFuture` 也非常灵活，既可以同步调用，也可以异步调用。

 复制代码

```
1  public class Client {
2      @Inject
3      private TransferService transferService; // 使用依赖注入获取转账服务的实例
4      private final static int A = 1000;
5      private final static int B = 1001;
6
7      public void syncInvoke() throws ExecutionException, InterruptedException {
8          // 同步调用
9          transferService.transfer(A, B, 100).get();
10         System.out.println(" 转账完成! ");
11     }
12 }
```



```
13     public void asyncInvoke() {
14         // 异步调用
15         transferService.transfer(A, B, 100)
16             .thenRun(() -> System.out.println(" 转账完成! "));
17     }
18 }
```

在调用异步方法获得返回值 `CompletableFuture` 对象后，既可以调用 `CompletableFuture` 的 `get` 方法，像调用同步方法那样等待调用的方法执行结束并获得返回值，也可以像异步回调的方式一样，调用 `CompletableFuture` 那些以 `then` 开头的一系列方法，为 `CompletableFuture` 定义异步方法结束之后的后续操作。比如像上面这个例子中，我们调用 `thenRun()` 方法，参数就是将转账完成打印在控台上这个操作，这样就可以实现在转账完成后，在控制台打印“转账完成！”了。

小结

简单的说，异步思想就是，**当我们要执行一项比较耗时的操作时，不去等待操作结束，而是给这个操作一个命令：“当操作完成后，接下来去执行什么。”**

使用异步编程模型，虽然并不能加快程序本身的速度，但可以减少或者避免线程等待，只用很少的线程就可以达到超高的吞吐能力。

同时我们也需要注意到异步模型的问题：相比于同步实现，异步实现的复杂度要大很多，代码的可读性和可维护性都会显著的下降。虽然使用一些异步编程框架会在一定程度上简化异步开发，但是并不能解决异步模型高复杂度的问题。

异步性能虽好，但一定不要滥用，只有类似在像消息队列这种业务逻辑简单并且需要超高吞吐量的场景下，或者必须长时间等待资源的地方，才考虑使用异步模型。如果系统的业务逻辑比较复杂，在性能能够满足业务需求的情况下，采用符合人类自然的思路且易于开发和维护的同步模型是更加明智的选择。

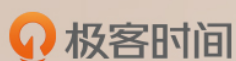
思考题

课后给你留 2 个思考题：

第一个思考题是，我们实现转账服务时，并没有考虑处理失败的情况。你回去可以想一下，在异步实现中，如果调用账户服务失败时，如何将错误报告给客户端？在两次调用账户服务的 Add 方法时，如果某一次调用失败了，该如何处理才能保证账户数据是平的？

第二个思考题是，在异步实现中，回调方法 OnComplete() 是在什么线程中运行的？我们是否能控制回调方法的执行线程数？该如何做？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。



消息队列高手课

从源码角度全面解析 MQ 的设计与实现

李玥

京东零售技术架构部资深架构师



新版升级：点击「👉 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 学习开源代码该如何入手？

下一篇 11 | 如何实现高性能的异步网络传输？

精选留言 (22)

写留言



笑傲流云

2019-08-13

个人的思路，欢迎老师点评下哈。

1, 调用账户失败, 可以在异步callBack里执行通知客户端的逻辑;
2, 如果是第一次失败, 那后面的那一步就不用执行了, 所以转账失败; 如果是第一次成功但是第二次失败, 首先考虑重试, 如果转账服务是幂等的, 可以考虑一定次数的重试, 如果不能重试, 可以考虑采用补偿机制, undo第一次的转账操作。...

展开 ▾

作者回复: 👍👍👍



13



senekis

2019-08-13

老师, 我一直有一个困惑, 就是想不明白为何异步可以节省线程。每次发起一个异步调用不都会创建一个新的线程吗? 我理解了好几次, 感觉只是异步处理线程在等待时可以让出时间片给其他线程运行啊? 一直想不明白这个问题, 困扰了好久, 求老师解惑。

展开 ▾

作者回复: 太多的线程会造成频繁的cpu上下文切换, 你可以想象一下, 假设你的小公司只有8台电脑, 你雇8个程序员一直不停的工作显然是效率最高的。考虑到程序员要休息不可能连轴转, 雇佣24个人, 每天三班倒, 效率也还行。

但是, 你要雇佣10000个人, 他们还是只能用这8台电脑, 大部分时间不都浪费在换人、交接工作上吗?

12

9



付永强

2019-08-13

老师可能里面过多提到线程这两个字, 所以很多人把异步设计理解成节约线程, 其实李玥老师这里想说明的是异步是用来提高cup的利用率, 而不是节省线程。
异步编程是通过分工的方式, 是为了减少了cpu因线程等待的可能, 让CPU一直处于工作状态。换句话说, 如果我们能想办法减少CPU空闲时间, 我们的计算机就可以支持更多的线程。...

展开 ▾

作者回复: 👍👍👍

线程就是为了能自动分配CPU时间片而生的。



4



linqw

2019-08-13

尝试回答课后习题，老师有空帮忙看下哦

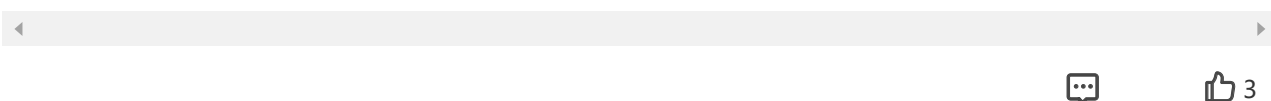
思考题一、如果在异步实现中，如果调用账户服务失败，可以以账单的形式将转账失败的记录下来，比如客户在转账一段时间后

查看账单就可以知道转账是否成功，只要保证转账失败，客户的钱没有少就可以。两次调用账户服务，感觉可以这样写...

展开 ∨

作者回复: 总结的非常好！

有一点需要改进一下，转账服务的实现中，异常处理的部分，还是需要先检查再补偿，否则有可能出现重复补偿的情况。



Better me

2019-08-13

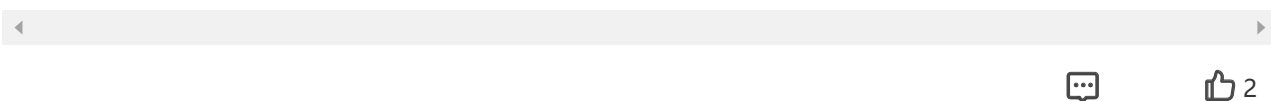
对于思考题:

1、应该可以通过程式事物来保证数据的完整性。如何将错误结果返回给客户端，感觉这边和老师上次答疑网关如何接收服务端秒杀结果有点类似，通过方法回调，在回调方法中保存下转账成功或失败

2、在异步实现中，回调方法 OnComplete()在执行OnAllDone()回调方法的那个线程，...

展开 ∨

作者回复: 666



谢清

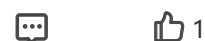
2019-08-13

学习了，一点思路，欢迎老师点评

第一个问题：

两次add方法保持最终一致性，第一次add失败不在调用第二次，告知客户转账失败；第一次成功调第二次失败，告知用户：转账进行中，转账对象收款中；可设置补偿策略，还是失败的话，转账后台人工介入补偿，还是不行则人工还原账户金额并告知用户：转账失...

展开 ∨



蓝魔、



2019-08-13

老师，转账例子代码中给转入账号加钱写错了吧

展开 ▾

作者回复: 感谢指正，我尽快让编辑小姐姐改正。



1



monalisali

2019-08-14

老师，问个问题：如何CPU和内存占有率都很高，用异步可以解决吗？

作者回复: 这个问题异步解决不了。如果说真的cpu或者磁盘占用率达到100%了，并且你的代码逻辑没什么问题，那这就是程序的极限了。



1



monalisali

2019-08-14

思考题一：

可以写一个兜底函数类似与catch，所有异常都走到这个函数中。然后通过传入catch的参数来判断错误类型，并决定后续操作

转入方法也类似，先判断下转出方法是否成功，成功了再执行。

展开 ▾



海罗沃德

2019-08-14

老师能否对比一下异步和StreamingData，据说StreamingData可以让线程使用率更高，效率比异步处理也更高，这是怎么实现的？

展开 ▾

作者回复: 你说的“StreamingData”指的是Lambda表达式还是流计算，或是其他什么技术呢？能具体说一下吗？



1



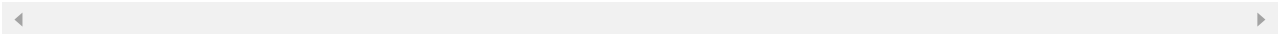
Liam



2019-08-14

异步实现里面还是要用线程池限制一下线程数吧，否则没有达到减少线程的效果

作者回复: 是的，一般都会使用线程池。



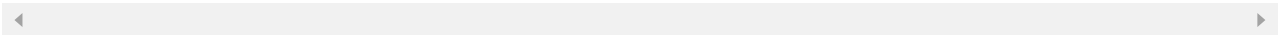
DAV

2019-08-13

请教一下，在整个消息队列的场景里面怎么融合异步调用？举例，A发送消息到消息队列，消费进程处理后如何通过回调形式返回结果给A？

展开

作者回复: 可以继续学习后面的课程，我们会有相应的源码分析。



川杰

2019-08-13

老师，请教个问题，吞吐量增加可以理解，因为请求发生后就直接返回了，从而避免了后续等待的延时；但是，以今天内容为例：

- 1、TransferAsync请求发生，直接返回，并开启新线程处理OnDebit函数；
- 2、OnDebit处理完毕，开启新线程处理OnAllDone函数；
- 3、OnAllDone函数处理完毕；...

展开

作者回复: 第一，OnDebit()和OnAllDone()可以在同一个线程中执行。没必要每个回调方法都开启一个新的线程。

第二，由于不需要等待，执行每个异步方法的耗时会非常短。

第三，可以使用线程池来避免反复创建销毁线程的开销，所以只需要很少的线程。

最后一个问题，“通知转账成功”这个业务逻辑，不一定非得在接收请求的那个线程里面执行，可以直接在OnAllDone()里面通知转账成功。



许童童

2019-08-13

如何将错误报告给客户端？

javascript中用.catch捕获异常

在两次调用账户服务的 Add 方法时，如果某一次调用失败了，该如何处理才能保证账户数据是平的？...

展开

作者回复: 点赞



亚洲舞王.尼古拉斯....

2019-08-13

1.老师，能否解释一下为什么“使用异步编程模型之后，在少量请求之下，时延依旧是100ms，但是在大量请求之下，异步的实现不需要等待线程的执行结果”？少量请求不也不需要等待吗

2.如果使用异步方式实现，我的onComplete()方法在另一个线程里执行，我怎么通知我的客户端我执行成功还是失败呢？

展开

作者回复: A1：是的，异步方式下少量和大量请求都不需要等待执行结果。

A2：在onComplete()方法中通知。



广训

2019-08-13

accountService如果本身是自己处理逻辑，那将其放入一个事务中就解决部分失败。如果调用三方服务，就比较麻烦。需要把两步操作都留存，失败的列表每天都要处理。

一般都有专门的异步线程池来运行异步task任务，比如spring 的async。

展开



Jxin

2019-08-13

1.提个问题，为什么要1w个线程？java的线程模型不是1:1的吗，实际在跑的线程仅有核数

*2, 1w这个数量不是造成了内存浪费和上下文切换成本吗。

2.异步异常回传, 并传回当前操作绑定在线程本地空间的事务实例。只有当两次都成功才提交两个add的事务。(事务不能声明, 只能手动开启和提交了)

展开 ∨



A:春哥大魔王

2019-08-13

jdk1.8之前没有completablefuture应该用什么搞异步呢?

作者回复: 一般是传一个回调方法, 就像我们这节课讲的那样。



1



leslie

2019-08-13

其实异步减少的是等待时间。关于第一个思考题: 调用失败是不是就可以抛出error, 直接不执行第一步; 代码中是否少了相应的查询操作进行核对啊, 否则万一就根本没有执行成功呢。操作完成后相应的查询核对应当时候放到了后面吧: 有问题再进行相应的类似于回滚之类的操作。

第二个问题oncomplete()在什么线程中执行这个不是很清楚: 不过我觉得可以通过线...

展开 ∨



HadesFX

2019-08-13

1.可以增加异步调用后的返回值用来判断是否成功。

2.应该是存在一个默认线程池, 使用线程池中的线程, 可能存在一个配置自定义线程池的方法用来定制控制数量。

不知道理解的对不对。

展开 ∨

