异步并行框架ParSeq

--李俊林

目录

[一、简介 3](#_Toc459233603)

[二、关键概念解释 3](#_Toc459233604)

[三、如何创建和运行一个Task 5](#_Toc459233605)

[四、Task的转换和组合 6](#_Toc459233606)

[五、异常处理和错误恢复 10](#_Toc459233607)

[六、超时处理 12](#_Toc459233608)

[七、Task取消 12](#_Toc459233609)

[八、执行过程跟踪 14](#_Toc459233610)

[九、单元测试 14](#_Toc459233611)

[十、集成ParSeq 14](#_Toc459233612)

[十一、总结 15](#_Toc459233613)

[十二、个人代码库 15](#_Toc459233614)

# 一、简介

ParSeq是LinkedIn开源的异步并行框架。具有如下优点：

1、异步操作并行化处理；

2、顺序执行非阻塞性计算；

3、通过任务组合实现代码重用；

4、简单的错误传播和恢复机制；

5、执行跟踪和可视化

获取ParSeq，目前最新版本是v2.0.3，使用ParSeq的v2.x需要jdk1.8.x以上支持

<dependency>  
  <groupId>com.linkedin.parseq</groupId>  
  <artifactId>parseq</artifactId>  
  <version>2.0.0</version>  
</dependency>

# 二、关键概念解释

1、Task：是ParSeq系统中一系列工作的基础，类似于Java的Callable，但是Task可以异步的获取结果。Task不能被用户直接执行，必须通过Engine执行。Task实现了类似于Java Future的Promise接口。Task可以被转换和组合并最终执行得到预期的结果。

2、Plan：Plan是一系列Task的集合，作为一个运行根Task的结果。

3、Engine：Task的执行者，通常一个普通的应用程序应用一个Engine实例。

Engine实例的代码如下：

import com.linkedin.parseq.Engine;  
import com.linkedin.parseq.EngineBuilder;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.ScheduledExecutorService;  
// ...  
final int numCores = Runtime.getRuntime().availableProcessors();  
final ExecutorService taskScheduler = Executors.newFixedThreadPool(numCores + 1);  
final ScheduledExecutorService timerScheduler =

Executors.newSingleThreadScheduledExecutor();

final Engine engine = new EngineBuilder()  
        .setTaskExecutor(taskScheduler)  
        .setTimerScheduler(timerScheduler)  
        .build();

//注：ParSeq将会用numCores + 1个线程执行所有与Task，一个线程用于调度定时器。

这是一个比较合理的配置，但是你也可以自定义。

//停止一个Engine

engine.shutdown();  
engine.awaitTermination(1, TimeUnit.SECONDS);  
taskScheduler.shutdown();  
timerScheduler.shutdown();

//执行这段代码，Engine将等待一秒钟之后关闭，在这个过程之中新的任务不被执行，允许正在运行的任务执行完成。这个操作也会关闭被ParSeq使用的

Executors，但是ParSeq本身并不会管理这些Executors的生命周期。

# 三、如何创建和运行一个Task

初始化任务通过集成ParSeq现有库创建，如果Task包含非阻塞的计算，将使用Task.action()或Task.callable()创建Task。对于大多数普通的Task，我们也提供Task.value和Task.failure()。新的Task是通过传输和组合现有Task完成的。

关于ParSeq API的建议：大多数创建带有版本Task的方法都接收对任务进行描述，建议给每一个Task都附上简单、清晰的描述，因为当使用ParSeq的跟踪机制来调试、解决问题的时候这会非常有用。

几乎Task接口的方法都创建一个新的Task实例，其中也许依赖的结果，以及被引擎执行的时间。

Task是懒惰的，它只是对能够被引擎执行的计算进行了描述，包括做什么，什么时候做。一旦你创建了一个Task你可以通过提交到ParSeq的执行引擎来执行该Task（engine.run(task)）。

# 四、Task的转换和组合

**1、转换**

转换Task的主要机制是通过map()方法。加入我们仅仅需要google首页HTTP内容类型，使用Task<Response>就能创建一个HTTP HEAD的请求：

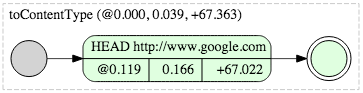
Task<Response> head = HttpClient.head("http://www.google.com").task();

我们可以使用下面的代码将其转换成一个可返回内容类型的Task。

Task<String> contentType = head.map("toContentType", response -> response.getContentType());

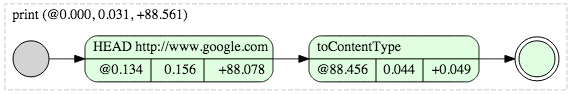
请注意，现有的head Task没有被修改。代替的是，一个新的任务被创建，head Task什么时候第一次被运行，运行完成之后提供的转换是什么。如果这个head Task因为各种原因失败，那么contentType 也应该失败，而且提供的转换不应该被调用。这个机制将在错误处理这一块进行详细解说。

使用ParSeq的跟踪工具我们可以得到如下的图形：



如果这里需要处理Task产生的结果，使用andThen方法就可以了：

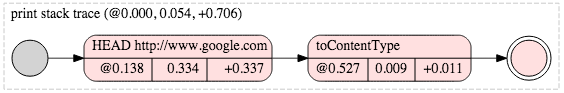
Task<String> printContentType = contentType.andThen("print", System.out::println);



上面的例子中，我们使用JAVA8的方法引用，当然我们也可以使用Lambda 表达式来完成：

Task<String> printContentType = contentType.andThen("print", s->System.out.println(s));

类似的，如果我们需要处理潜在的错误，我们可以使用onFailure()方法:

Task<String> logFailure = contentType.onFailure("print stack trace", e -> e.printStackTrace())

在处理更大潜在错误的时候使用更简单的toTry()方法更加有用，将Task<T>转换成Task<Try<T>>。

Task<Try<String>> contentType = head.map("toContentType", response -> response.getContentType.toTry());

Task<Try<String>> logContentType = contentType.andThen("log", type->{

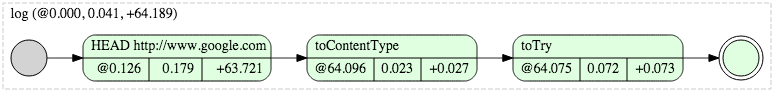
if (type.isEnabled()) {

type.getError().printStackTrace();

} else {

System.out.println("ContentType:" + type.get());

} });

上面的例子中，我们使用JAVA8的方法引用，当然我们也可以使用Lambda 表达式来完成：

Task<String> printContentType = contentType.andThen("print", s->System.out.println(s));

类似的，如果我们需要处理潜在的错误，我们可以使用onFailure()方法:

Task<String> logFailure = contentType.onFailure("print stack trace", e -> e.printStackTrace());

最后，transform()方法将组合toTry()和map();

Task(Response) get = HttpClient.get("http://www.google.com").task();

Task<Optional<String>> contents = get.transform("getContents", tryGet -> {

if (tryGet.isFailed()) {

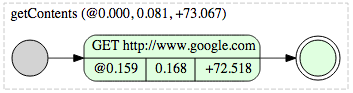
return Success.of(Optional.empty());

  } else {

return Success.of(Optional.of(tryGet.get().getResponseBody()));

}

});



在上面的列子中，如果HTTP GET请求失败，"contents" Task总是成功完成并返回Optional 或Optional.empty()包装的谷歌首页内容。

2、组合

很多Task是由其他许多串行或并行的Task组成。

并行组合

假如希望异步抓取几个不同页面内容类型组成。首页，我们需要创建一个帮助性的方法来负责从一个URL获取内容类型。

private Task <String> getContentType(String url) {

return HttpClient.get(url).task().map("getContentType", response -> response.getContentType());

}

然后我们可以使用Tasl.par()方法组合这些任务来异步运行。

final Task<String> googleContentType = getContentType("http://www.google.com");

final Task<String> bingContentType = getContentType("www.bing.com"); final Task<String> contentTypes = Task.par(googleContentType, bingContentType).map("concatenate", (google, bing)-> "Google:" + goolge + "\n" + "Bing:" + bing + "\n");

Task.par()创建一个新的Task异步运行"googleContentType"和"bingContentType"。使用map()方法将执行的结果转换成一个字符串。

# 五、异常处理和错误恢复

ParSeq中的一个重要原则错误总是传播给他们依赖的Task。通常，这里不需要Catch或重新抛出异常。

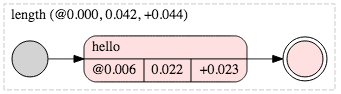
Task<String> failing = Task.callable("hello", () -> {

return "Hello World".sbustring(100);

  });

Task<Integer> length = failing.map("length", s-> s.length());

上面关于Lenght的列子会因为java.lang.StringIndexOutOfBoundsException失败，并且从failling Task中传播出来。



通常降级行为是一个更好的选择相对简单的错误传播。如果存在一个合理错误回滚值，可以使用recover()从错误中恢复。

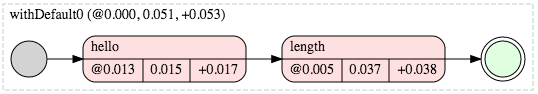
Task<String> failing = Tasl.callable("hello", ()->{

return "Hello World".substring(100);

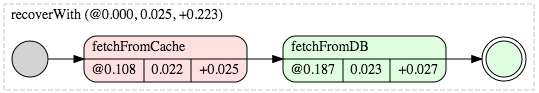
});

Task<Integer> length = failing.map("length", s->length()).recover("withDefault0", e->0);

这次Length Task将恢复默认值0从java.lang.StringIndexOutOfBoundsException中恢复。请注意，错误回滚机制允许将导致错误的异常作为一个参数。



有时候我们没有回退值可以使用，但是我们可以使用另外一个Task继续完成计算。在这种情况下，我们可以使用recoverWith()方法。recover()和recoverWith()方法的区别是后者返回一个包含可退步值将被执行的Task实例。下面的例子将演示，当我们从缓存中获取用户失败之后从数据库中获取用户信息。

Task<Person> user = fetchFromCache(id).recoverWith(e ->fetchFromDB(id)); 

# 六、超时处理

给异步Task设置超时时间是一个好的建议，ParSeq提供了withTimeout来完成这项工作。

final Task<Response> google = HttpClient.get("http://google.com").task().withTimeout(10, TimeUnit.MILLISECONDS);

在上面的列子中，如果抓取google.com的内容超过10ms,Task将会因为TimeoutException失败。

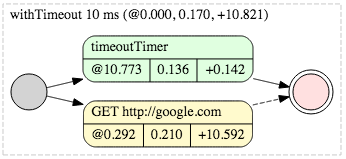
# 七、Task取消

ParSeq支持取消Task。取消一个Task意味着导致这个Task不在有任何相关性。

Task在任何时候都能够被取消。

Task实现了当Task被取消时能够给侦测到，并且做出相应的反应。通过CancellationException完成取消功能，因此，这个行为向一个失败了的Task一样将会将取消传递给所有依赖这个Task的Task。虽然取消动作是一个高效的失败，当Task被取消之后recover()，recoverWith()，onFailure()将不能继续呗调用。原因就是，取消Task意味着导致一个Task无相关，因此不能尝试从这种常见进行错误恢复。使用cancel()取消一个Task。 自动取消：通常一个Task的目的是通过计算得到一个值。你可以将一个Task作为一个异步功能。一旦值被计算出来，就不必继续运行这个Task。因此任务TaskParSeq只运行一次，引擎能够识别已经完成或已经启动的Task并且不再执行他们。

在一个Task执行完成或开始运行之前已经获取结果值这是可能的。其中一种情况是当我们为一个Task设置一个超时时间。指定超时时间的Task可能因为timeoutTime失败，但是原始的Task可能仍然在继续执行。在这种情况下，ParSeq将通过 EarlyFinishException来自动取消原始的Task。



10ms之后，计算结果的Task将会失败，如红色部分；原始的Task会通过EarlyFinishException自动取消，黄色部分。

# 八、执行过程跟踪

https://github.com/linkedin/parseq/wiki/Tracing

# 九、单元测试

ParSeq提供了一个test模块包含一个BaseEngineTest可以被用作ParSeq相关测试用例的基类。将会自动的创建、关闭执行引擎为每一个测试用例 并且提供很多有用的方法用户执行、跟踪Task。

<dependency>

<groupId>com.linkedin.parseq</groupId> <artifactId>parseq</artifactId>

<version>2.0.0</version>

<classifier>test</classifier>

<scope>test</scope>

</dependency>

# 十、集成ParSeq

1、这个部分描述了ParSeq已经存在的异步库，我们提供了下面两种例子也许对进一步的指导比较有用。

parseq-http-client 集成了异步的http client并且提供了执行异步HTTP请求的Task。

parseq-exec 集成了 JAVA`S Process API 并且提供了异步运行本地程序的Task。

2、集成异步API

使用Task.async()方法创建用于异步完成任务的Task实例。接受Callable或Function1参数，返回一个Promise实例。

3、集成阻塞API

不是每个库都提供异步API，如JDBC。我们不应该在ParSeq内部直接阻塞代码，因为这样会影响其他异步Task。我么可以通过Task.blocking()方法集成阻塞API。接受两个参数：

Callable：将被执行的代码

Executor：callable将被调用的实例。

# 十一、总结

**1、异步并行的定义和使用场景**

**2、ForkJoin**

**3、京东的异步并行框架Sirector**

http://jpcloud.jd.com/pages/viewpage.action?pageId=753788

**4、LinkedIn的异步并行框架ParSeq**

[**https://github.com/linkedin/parseq**](https://github.com/linkedin/parseq)

# 十二、个人代码库

**https://github.com/walkerljl**