puyangsky
厚积薄发

【Java】关于Java8 parallelStream并发安全的思考

背景

Java8的stream接口极大地减少了for循环写法的复杂性，stream提供了map/reduce/collect等一系列聚合接口，还支持并发操作：parallelStream。

在爬虫开发过程中，经常会遇到遍历一个很大的集合做重复的操作，这时候如果使用串行执行会相当耗时，因此一般会采用多线程来提速。Java8的paralleStream用fork/join框架提供了并发执行能力。但是如果使用不当，很容易陷入误区。

Java8的paralleStream是线程安全的吗

一个简单的例子,在下面的代码中采用stream的forEach接口对1-10000进行遍历，分别插入到3个ArrayList中。其中对第一个list的插入采用串行遍历，第二个使用paralleStream，第三个使用paralleStream的同时用ReentryLock对插入列表操作进行同步：

```
private static List<Integer> list1 = new ArrayList<>();
private static List<Integer> list2 = new ArrayList<>();
private static List<Integer> list3 = new ArrayList<>();
private static Lock lock = new ReentrantLock();

public static void main(String[] args) {
    IntStream.range(0, 10000).forEach(list1::add);

    IntStream.range(0, 10000).parallel().forEach(list2::add);

    IntStream.range(0, 10000).forEach(i -> {
    lock.lock();
    try {
        list3.add(i);
    }finally {
        lock.unlock();
    }
    });

    System.out.println("串行执行的大小：" + list1.size());
    System.out.println("并行执行的大小：" + list2.size());
    System.out.println("加锁并行执行的大小：" + list3.size());
}
```

执行结果：

```
串行执行的大小：10000
并行执行的大小：9595
加锁并行执行的大小：10000
```

并且每次的结果中并行执行的大小不一致，而串行和加锁后的结果一直都是正确结果。显而易见，stream.parallel.forEach()中执行的操作并非线程安全。

那么既然paralleStream不是线程安全的，是不是在其中的进行的非原子操作都要加锁呢？我在stackOverflow上找到了答案：

- https://codereview.stackexchange.com/questions/60401/using-java-8-parallel-streams
- https://stackoverflow.com/questions/22350288/parallel-streams-collectors-and-thread-safety

在上面两个问题的解答中，证实paralleStream的forEach接口确实不能保证同步，同时也提出了解决方案：使用collect和reduce接口。

http://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

在Javadoc中也对stream的并发操作进行了相关介绍：

昵称：puyangsky
园龄：2年5个月
粉丝：26
关注：16
+加关注

< 2017年10月 >

日	一	二	三	四	五	六
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

最新随笔

1. 【Java】关于Java8 parallelStream并发安全的思考
2. 【C++】bazel的使用
3. 近期项目计划
4. 【杂谈】研究生最后一年学习计划
5. 【mac】ssh免登录密码
6. 【分布式】ZooKeeper学习之一：安装及命令行使用
7. 【SpringBoot】关闭HttpClient无用日志
8. 【Leetcode】142. Linked List Cycle II
9. 【数据结构】Trie树
10. 【算法】动态规划经典题之最长公共子序列

我的标签

leetcode(27)
java(25)
算法题(10)
C++(6)
go(6)
spring(4)
python(3)
Linux(3)
docker(3)
Android(3)
更多

> The Collections Framework provides synchronization wrappers, which add automatic synchronization to an arbitrary collection, making it thread-safe.

Collections框架提供了同步的包装，使得其中的操作线程安全。

所以下一步，来看看collect接口如何使用。

stream的collect接口

闲话不多说直接上源码吧，Stream.java中的collect方法句柄：

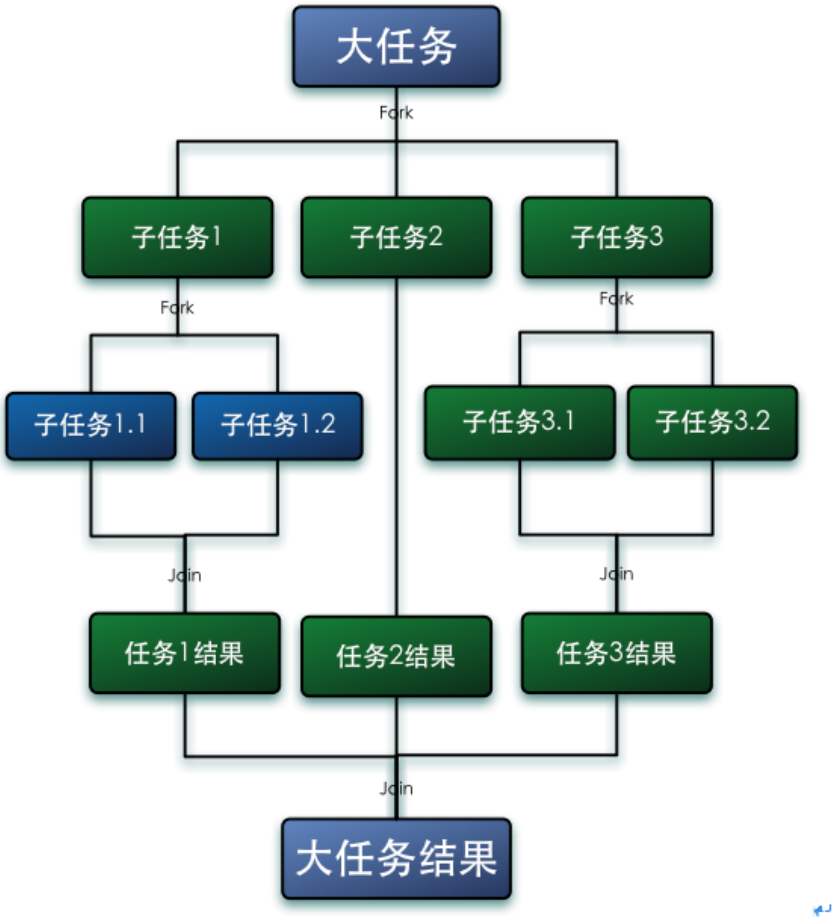
```
<R, A> R collect(Collector<? super T, A, R> collector);
```

在该实现方法中，参数是一个Collector对象，可以使用Collectors类的静态方法构造Collector对象，比如Collectors.toList()，toSet(),toMap()，etc，这块很容易查到API故不细说了。

除此之外，我们如果要在collect接口中做更多的事，就需要自定义实现Collector接口，需要实现以下方法：

```
Supplier<A> supplier();
BiConsumer<A, T> accumulator();
BinaryOperator<A> combiner();
Function<A, R> finisher();
Set<Characteristics> characteristics();
```

要轻松理解这三个参数，要先知道fork/join是怎么运转的，一图以蔽之：



上图来自：http://www.infoq.com/cn/articles/fork-join-introduction

简单地说就是大任务拆分成小任务，分别用不同线程去完成，然后把结果合并后返回。所以第一步是拆分，第二步是分开运算，第三步是合并。这三个步骤分别对应的就是Collector的supplier,accumulator和combiner。talk is cheap show me the code，下面用一个例子来说明：

输入是一个10个整型数字的ArrayList，通过计算转换成double类型的Set，首先定义一个计算组件：

Compute.java:

```
public class Compute {
public Double compute(int num) {
    return (double) (2 * num);
}
}
```

随笔分类

C++学习(1)
Golang学习(2)
Java多线程(2)
Java学习(14)
js学习(1)
JVM学习笔记(1)
Leetcode(9)
分布式(1)
剑指offer(5)
设计模式(1)

随笔档案

2017年9月 (6)
2017年8月 (1)
2017年6月 (4)
2017年5月 (2)
2017年4月 (5)
2017年3月 (7)
2017年2月 (2)
2017年1月 (3)
2016年12月 (6)
2016年11月 (3)
2016年9月 (2)
2016年8月 (6)
2016年7月 (3)
2016年6月 (9)
2016年5月 (8)
2016年4月 (7)
2016年3月 (13)
2016年1月 (6)
2015年12月 (1)
2015年11月 (6)
2015年10月 (3)
2015年9月 (1)
2015年8月 (3)
2015年7月 (14)

相册

1(1)

积分与排名

积分 - 48077
排名 - 6764

最新评论

1. Re:【Java】关于Java8 parallelStream并发安全的思考
文章写的不错赞一个~！ public Container accumulate(int num) { this.set.add(compute.compute(num)); retu.....
--祈求者-
2. Re:【Java】关于Java8 parallelStream并发安全的思考
牛
--wangshoumao
3. Re:【爬虫】python requests模拟登录知乎
@zrzhou你直接自己登陆一下，然后用chrome的F12工具去看network里发送的http请求，肯定可以找到登陆的请求url，我这写的有点久了说不定知乎首页登陆方式都变了...
--puyangsky
4. Re:【爬虫】python requests模拟登录知乎
你好，可以问一个问题吗？你这个表单提交地址 baseurl += "/login/email"中的/lo

接下来在Main.java中定义输入的类型为ArrayList的nums和类型为Set的输出结果result：

```
private List<Integer> nums = new ArrayList<>();
private Set<Double> result = new HashSet<>();
```

定义转换list的run方法，实现Collector接口，调用内部类Container中的方法，其中characteristics()方法返回空set即可：

```
public void run() {
    // 填充原始数据，nums中填充0-9 10个数
    IntStream.range(0, 10).forEach(nums::add);
    //实现Collector接口
    result = nums.stream().parallel().collect(new Collector<Integer, Container,
Set<Double>>() {

    @Override
    public Supplier<Container> supplier() {
        return Container::new;
    }

    @Override
    public BiConsumer<Container, Integer> accumulator() {
        return Container::accumulate;
    }

    @Override
    public BinaryOperator<Container> combiner() {
        return Container::combine;
    }

    @Override
    public Function<Container, Set<Double>> finisher() {
        return Container::getResult;
    }

    @Override
    public Set<Characteristics> characteristics() {
        // 固定写法
        return Collections.emptySet();
    }
    });
}
```

构造内部类Container，该类的作用是一个存放输入的容器，定义了三个方法：

- accumulate方法对输入数据进行处理并存入本地的结果
- combine方法将其他容器的结果合并到本地的结果中
- getResult方法返回本地的结果

Container.java:

```
class Container {
    // 定义本地的result
    public Set<Double> set;

    public Container() {
    this.set = new HashSet<>();
    }

    public Container accumulate(int num) {
    this.set.add(compute.compute(num));
    return this;
    }

    public Container combine(Container container) {
    this.set.addAll(container.set);
    return this;
    }

    public Set<Double> getResult() {
    return this.set;
    }
}
```

在Main.java中编写测试方法：

gin/email是怎么知道的啊?我在页面中找了好久就只有/register/email...

--zrzhou

5. Re:leetcode 287 Find the Duplicate Number

@zzysli可以了解一下floyd判圈算法...

--zxzhang

阅读排行榜

1. 【ModelMap】jsp中显示springmvc modelmap传递的对象(6708)
2. 【Java】一次SpringMVC+ Mybatis配置多数据源经历(5252)
3. 【Jersey】IntelliJ IDEA + Maven + Jetty + Jersey搭建RESTful服务(4690)
4. 【Spring学习】在Spring+Maven环境中使用Junit Test(3478)
5. 【java回调】同步/异步回调机制的原理和使用方法(3199)

评论排行榜

1. 【Java】一次SpringMVC+ Mybatis配置多数据源经历(23)
2. 【Java NIO】一文了解NIO(6)
3. 【Spring】利用AOP来做系统性能监控(5)
4. 【vim】mac配置vim，molokai配色(3)
5. leetcode 102 Binary Tree Level Order Traversal(3)

推荐排行榜

1. 【Java NIO】一文了解NIO(4)
2. 【Java】基本I/O的学习总结(3)
3. 【Jersey】IntelliJ IDEA + Maven + Jetty + Jersey搭建RESTful服务(3)
4. 【Numpy】python机器学习包Numpy基础知识学习(1)
5. 【Java】关于Java8 parallelStream并发安全的思考(1)

```
public static void main(String[] args) {
    Main main = new Main();
    main.run();
    System.out.println("原始数据：");
    main.nums.forEach(i -> System.out.print(i + " "));
    System.out.println("\n\ncollect方法加工后的数据：");
    main.result.forEach(i -> System.out.print(i + " "));
}
```

输出：

```
原始数据：
0 1 2 3 4 5 6 7 8 9

collect方法加工后的数据：
0.0 2.0 4.0 8.0 16.0 18.0 10.0 6.0 12.0 14.0
```

我们将10个整型数值的list转成了10个double类型的set，至此验证成功~

本程序参考 http://blog.csdn.net/io_field/article/details/54971555。

一言蔽之

总结就是paralleStream里直接去修改变量是非线程安全的，但是采用collect和reduce操作就是满足线程安全的了。

分类: Java学习

标签: java, stream

puyangsky
关注 - 16
粉丝 - 26
+加关注

1 0

« 上一篇：【C++】bazel的使用

posted @ 2017-09-28 21:40 puyangsky 阅读(191) 评论(2) 编辑 收藏

评论

#1楼 2017-09-29 11:03 | wangshoumao

牛

支持(0) 反对(0)

#2楼 2017-09-30 17:35 | 祈求者-

文章写的不错赞一个~！

```
1  public Container accumulate(int num) {
2      this.set.add(compute.compute(num));
3      return this;
4      }
```

楼主这里改成

```
1  public void accumulate(int num) {
2  this.set.add(compute.compute(num));
3  }
```

感觉可读性会高一些 我试了一下结果是一样的
楼主还是写的很清楚的 就是collector的方法和下面container的方法签名我看了半天：(,主要就是BiConsumer<Container, Integer> accumulator() 这个签名的方法应该是没有返回值的,你原文的有一个返回值,看的时候我感觉不太匹配,自己写了一遍发现编译是可以过的,后来想了一下应该是this关键字的原因,然后我就修改成我上面的,也是可以的T_T,再比如下面的getResult 上面签名是Function<Container, Set<Double>>，下面的签名没有参数,有set<double>的返回值,我理解的是类似于python方法中的的self参数一样,this是在参数里的(如果你用到了的话),只是java将他隐式的表达了,那Container中的getResult的签名其实是<this,Set<Double>>，包括上面的BiConsumer<Container, Integer> 对应下面的两个参数就是<this,Integer>，不晓得这么理解对不对哦。。