链滴

多个 Iterable<T> 合成一个 Iterable<T>

作者：Gakkiyomi2019
原文链接：https://ld246.com/article/1682479199212
来源网站：链滴

使用Java Stream[1]实现这个功能

```
import java.util.stream.Stream;
import java.util.stream.StreamSupport;

public class IterableUtils {
    public static <T> Iterable<T> concat(Iterable<T>... iterables) {
        return () -> Stream.of(iterables)
            .flatMap(it -> StreamSupport.stream(it.spliterator(), false))
            .iterator();
    }
}
```

1.

Java Stream

parallelStream

在 Java 8 中，可以使用并行流来并行处理数据。并行流是通过将数据分成多个部分来并行处理的，与 **ForkJoin** 框架中的分治算法类似。

下面是一个示例，它使用并行流来查找数组中的最大值：

```
import java.util.Arrays;

int[] array = { ... };
int max = Arrays.stream(array).parallel().max().getAsInt();
```

在这种情况下，并行流会自动将数组分成多个部分，并在多个线程上并行处理这些部分。

也可以在并行流上使用自定义的ForkJoinPool,来控制线程数量：

```
ForkJoinPool customThreadPool = new ForkJoinPool(4);
int max = customThreadPool.submit(() ->
    Arrays.stream(array).parallel().max().getAsInt()
).get();
```

需要注意的是，使用并行流可能会导致额外的内存使用，因为它需要复制数据来在多个线程上并行处。因此，如果数据量很大或内存受限，可能需要使用其他方法来并行处理数据。

并行流是根据什么来分配数组数据到各个线程的呢?

在 Java 8 中，并行流使用的是分块策略来将数据分配到各个线程。分块策略的基本思想是将数据分固定大小的块，并将每个块分配给单独的线程来处理。

具体实现上，并行流会使用 Spliterator 接口来定义数据的分块策略。Spliterator 接口提供了 trySplit(方法，该方法将当前数据分成两部分，返回第一部分的 Spliterator，并将第二部分留在当前 Splitera or 中。

例如，对于数组，并行流会使用 ArraySpliterator 来定义分块策略。ArraySpliterator 类会将数组分固定大小的块，并在 trySplit() 方法中使用这些块来分配数据。

在使用并行流时，可以通过使用 parallelism 属性来控制线程数量，来调整分配数据的粒度，从而影程序的性能

但是需要注意的是,这只是默认的分块策略,并不是所有情况都适用,在某些情况下需要自定义分块策略获得最佳性能.