这里需要 @SuppressWarnings("unchecked")，因为不确定我们向 Class.forName(type).newInstance() 传递的 String 到底是什么。编译器不能确保这个操作一定成功。

RLIST 是 HTMLColors.LIST 的逆向版本。因为 Collections.reverse() 通过修改参数来实现逆向，而不是返回一个包含逆向排列元素的新 List，所以这个调用在 static 子句内执行。RLIST 可以防止我们误认为 Set 会对结果进行排序。

HashSet 的输出看上去没有明显的顺序（因为它是基于哈希函数的）。TreeSet 和 ConcurrentSkipListSet 都会对其元素进行排序，而且都实现了 SortedSet 接口来表明这一点。因为这样的 Set 是有序的，所以它们还提供了更多操作。LinkedHashSet 和 CopyOnWriteArraySet 会保留元素插入的顺序，尽管没有接口表明这一点。

ConcurrentSkipListSet 和 CopyOnWriteArraySet 是线程安全的。

在本章最后，我们会了解非 HashSet 实现执行排序所带来的性能开销，以及不同实现中其他功能的开销。

3.4 在 Map 上使用函数式操作

和 Collection 接口一样，Map 接口也内置了 forEach()。但是如果我们想执行像 map()、flatMap()、reduce() 或 filter() 等其他基本操作，又该怎么做呢？看一下 Map 接口，并没有与这些操作相关的线索。

我们通过 entrySet() 连接到这些方法，它会生成一个由 Map.Entry 对象组成的 Set。这个 Set 又包含了 stream() 和 parallelStream() 方法。只需要记住：我们在使用 Map.Entry 对象。

```
// collectiontopics/FunctionalMap.java
// 在 Map 上执行函数式操作
import java.util.*;
import java.util.stream.*;
import java.util.concurrent.*;
import static onjava.HTMLColors.*;

public class FunctionalMap {
  public static void main(String[] args) {
    MAP.entrySet().stream()
      .map(Map.Entry::getValue)
      .filter(v -> v.startsWith("Dark"))
      .map(v -> v.replaceFirst("Dark", "Hot"))
      .forEach(System.out::println);
```