39 集合相关新增规约的启示

更新时间：2020-10-22 18:05:33

> 富贵必从勤苦得。——杜甫

1. 前言

《手册》泰山版的集合处理相关小节新增了一些规范，如：判断所有集合内的元素是否为空，使用 isEmpty () 方法，而不是 size ()==0 的方式；在使用 java.util.stream.Collectors 类的 toMap () 方法转 Map 集合时，一定要使用含义参数类型为 BinaryOperator ，参数名为 mergeFunction 的方法，否则当出现相同 key 值时会出现 IllegalStateException 。

前面多次讲到，本专栏基于手册又要脱离手册，这些规约给我们带来哪些启示呢？

2. 启示

2.1 先猜想后验证

之前也反复提到 “先猜想后验证” 的学习方法。

大家在学习某个知识时，要想思考为什么这样，然后去看源码、查资料以及和别人交流等验证和猜想是否一致，才能更加深入地理解知识。

如果没有先猜想后验证，我会认为很多做法是理所当的，不会去主动思考，理解就必然不够深刻。

“判断所有集合内的元素是否为空，使用 isEmpty () 方法，而不是 size ()==0 的方式”，我们可以猜想 isEmpty () 更好理解， isEmpty () 时间复杂度可能更低。

“在使用 java.util.stream.Collectors 类的 toMap () 方法转 Map 集合时，一定要使用含义参数类型为 BinaryOperator , 参数名为 mergeFunction 的方法，否则当出现相同 key 值时会出现 IllegalStateException 。”

其实我们要思考，将集合转为 Map 时如果有多个 key 相同的元素应该怎么办？

如果我是 JDK 的设计者，这种情况下，是否应该给出一个参数，让用户将冲突的策略传递进来？ 如果没有传递策略，是否就应该抛出异常？如果抛出异常的话哪种异常最合适呢？

这里的策略就是将两个相同类型的对象合并成一个，因此可以用 BinaryOperator 。如果没有传递策略，出现重复得让用户感知到，因此要抛出异常。因为此时可以选择 IllegalStateException （这里猜错了也没关系）。

但是猜想之后还要进行验证。

2.2 多看经典的图书

相信很多人能够理解“磨刀不误砍材工”、“欲速则不达”的道理。

但是由于学校中多年形成的根深蒂固的“分数至上”惯性思维，让我学校里只学考试考到的知识，毕业了只学面试用到的知识。

建议大家打牢计算机专业基础，多看一些经典的计算机科学相关图书。

比如《*A Philosophy of Software Design*》（软件设计的哲学）一书中就讲到**深模块是降低系统复杂度的重要手段**。

所谓的深模块就是暴露的接口尽可能简单，尽可能隐藏复杂性。

有了这个背景知识之后，我们再看“判断所有集合内的元素是否为空，使用 isEmpty () 方法，而不是 size ()==0 的方式”就有了不一样的理论基础，就更容易理解了。

此书所讲的“深模块”和软件工程领域的“高内聚低耦合”的思想也是一致的。

2.3 读源码

判断所有集合内的元素是否为空，使用 isEmpty () 方法，而不是 size ()==0 的方式。理由是前者的时间复杂度为 O (1) ，而且可读性更好。

很多人看到这里或许就记住了这个结论，这样是不够的。

“前者的时间复杂度为 O (1) ”，所以“后者的时间复杂度不是 O (1)?”, 真的是这样吗？

LinkedList 为例其 size 函数直接返回 size 成员属性：

```
/**
    * Returns the number of elements in this list.
    *
    * @return the number of elements in this list
    */
   public int size() {
       return size;
   }
```

而 isEmpty 则继承自 AbstractCollection

```
public abstract int size();

/**
 * {@inheritDoc}
 *
 * <p>This implementation returns <tt>size() == 0</tt>.
 */
public boolean isEmpty() {
    return size() == 0;
}
```

我们发现对于 LinkedList 而言 size () 函数的时间复杂度也是 O (1)，而且底层也是利用 size () == 0 实现的。

大家可以去 ArrayList 的源码中查看这两个函数的实现：

```
public int size() {
    return size;
}

public boolean isEmpty() {
    return size == 0;
}
```

可以看出很多集合类的 size () 函数的时间复杂度也是 O (1)。

“在使用 java.util.stream.Collectors 类的 toMap () 方法转 Map 集合时，一定要使用含义参数类型为 BinaryOperator , 参数名为 mergeFunction 的方法，否则当出现相同 key 值时会出现 IllegalStateException 。”

java.util.stream.Collectors#toMap 源码：

```
/**
 * Returns a {@code Collector} that accumulates elements into a
 * {@code Map} whose keys and values are the result of applying the provided
 * mapping functions to the input elements.
 *
 * <p>If the mapped
 * keys contains duplicates (according to {@link Object#equals(Object)}),
 * the value mapping function is applied to each equal element, and the
 * results are merged using the provided merging function.
 *
 * @apiNote
 * There are multiple ways to deal with collisions between multiple elements
 * mapping to the same key.  The other forms of {@code toMap} simply use
 * a merge function that throws unconditionally, but you can easily write
 * more flexible merge policies.  For example, if you have a stream
 * of {@code Person}, and you want to produce a "phone book" mapping name to
 * address, but it is possible that two persons have the same name, you can
 * do as follows to gracefully deals with these collisions, and produce a
 * {@code Map} mapping names to a concatenated list of addresses:
 * <pre>{@code
 *     Map<String, String> phoneBook
 *         people.stream().collect(toMap(Person::getName,
 *                                       Person::getAddress,
 *                                       (s, a) -> s + ", " + a));
 * }</pre>
 *
 * // 省略其他
 *
 * @see #toMap(Function, Function)
 * @see #toMap(Function, Function, BinaryOperator, Supplier)
 * @see #toConcurrentMap(Function, Function, BinaryOperator)
 */
public static <T, K, U>
Collector<T, ?, Map<K,U>> toMap(Function<? super T, ? extends K> keyMapper,
                                Function<? super T, ? extends U> valueMapper,
                                BinaryOperator<U> mergeFunction) {
    return toMap(keyMapper, valueMapper, mergeFunction, HashMap::new);
}
```

源码注释给出了非常详细的介绍，如果映射的 key 重复，则会使用合并方法对值进行合并。而且代码的注释中给出了典型的用法。

因此印证了我们的猜想，而且再次印证了读源码注释是学习的一个非常好的途径。

2.4 动手实践

俗话说：“实践出真知”。

大家完全可以动手写一个 DEMO 来验证书中的观点是否正确，并且加深自己的印象。

```
import org.junit.Test;

import java.util.Map;
import java.util.stream.Stream;

import static java.util.stream.Collectors.toMap;

public class MapDemo {

    @Test(expected = IllegalStateException.class)
    public void test() {
        // 构造姓名重复的数据
        Stream<Person> people = Stream
                .of(new Person("张三", "杭州")
                        , new Person("张三", "深圳")
                        , new Person("李四", "南京"));
        Map<String, String> phoneBook = people.collect(toMap(Person::getName, Person::getAddress));
    }
}
```

可以在 toMap 方法中打断点进行调试，也可以使用调试技巧小节讲到的根据异常来断点，查看调用栈来理解出错的原因等。

3. 总结

本节主要介绍了集合相关新增建议的启示，主要是“先猜想后验证”、多看经典书目、看源码和动手实践。

希望大家在后续的学习过程中能够灵活运用专栏所介绍的各种方法。

4. 思考和练习

1、编码过程中多打开常用类的源码，查看函数列表，常用函数的源码和注释。

2、请下载 commons-collections 和 guava 源码，了解集合相关工具类和基本用法。

欢迎在下方留言评论。

}

38 日期相关新增规约的启示

40 错误码相关新增规约的启示