Guava使用手册

lobster

目录

[Guava使用手册 1](#_Toc381711273)

[1 Guava简介 3](#_Toc381711274)

[2 Guava使用手册 4](#_Toc381711275)

[2.1 基本的Guava工具。 4](#_Toc381711276)

[2.1.1 Joiner的使用 4](#_Toc381711277)

[2.1.2 Spliter的使用 5](#_Toc381711278)

[2.1.3 Charsets的使用 7](#_Toc381711279)

[2.1.4 Strings的使用 8](#_Toc381711280)

[2.1.5 CharMatcher的使用 9](#_Toc381711281)

[2.1.6 Preconditions的使用 10](#_Toc381711282)

[2.1.7 Objects工具的使用 10](#_Toc381711283)

[2.2 Guava Collections使用 12](#_Toc381711284)

[2.2.1 Collections的创建 12](#_Toc381711285)

[2.2.2 Lists的高级用法 13](#_Toc381711286)

[2.2.3 Sets的高级用法 14](#_Toc381711287)

[2.2.4 Maps的高级用法 16](#_Toc381711288)

[2.2.5 BiMap 17](#_Toc381711289)

[2.2.6 Multiset 18](#_Toc381711290)

[2.2.7 MultiMap 19](#_Toc381711291)

[2.2.8 不可变集合 20](#_Toc381711292)

[2.3 Guava的函数式编程 23](#_Toc381711293)

[2.3.1 Predicates 过滤 23](#_Toc381711294)

[2.3.2 Functions 转换 24](#_Toc381711295)

[2.4 Guava Concurrency 25](#_Toc381711296)

Guava简介

Guava是Google Java工程的核心基础类库，主要包括：集合，缓存，原始类型扩展，并发，注解，字符串处理，IO等等。这些高质量的 API 可以使你的JAVa代码更加优雅，更加简洁，让你工作更加轻松愉悦。下面我们就开始学习如何使用Guava。

**Guava包的简单说明：**

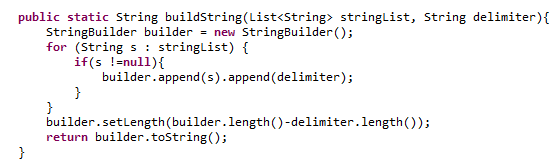
* com.google.common.annotations：普通注解类型。
* com.google.common.base：基本工具类库和接口。
* com.google.common.cache：缓存工具包，非常简单易用且功能强大的JVM内缓存。
* com.google.common.collect：带泛型的集合接口扩展和实现，以及工具类。
* com.google.common.eventbus：发布订阅风格的事件总线。
* com.google.common.hash： 哈希工具包。
* com.google.common.io：I/O工具包。
* com.google.common.math：原始算术类型和超大数的运算工具包。
* com.google.common.net：网络工具包。
* com.google.common.primitives：八种原始类型和无符号类型的静态工具包。
* com.google.common.reflect：反射工具包。
* com.google.common.util.concurrent：多线程工具包。

Guava使用手册

基本的Guava工具。

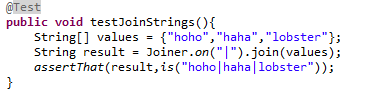
Joiner的使用

将一些字符串以指定的分割符拼接在一起，是我们经常做的事情。通常我们的代码会这样写：

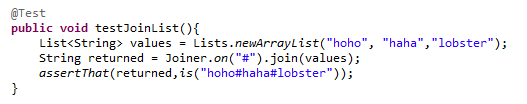


上述的例子只是拼接了一个List，但如果我们要拼接数组或其他集合呢？还要写若干个上述方法。上面的程序还有一段null的处理，如果为空就自动忽略，但我们现在的需求是如果为空则用其他字符串代替呢？是不是还要写好多方法，下面来看一下用Joiner如何简单的实现上述功能

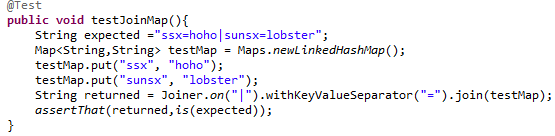
1. 拼接字符串数组：



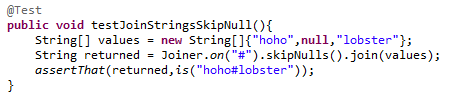
1. 拼接List



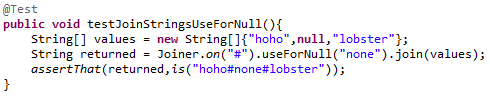
1. 拼接Map



1. 拼接字符串，并忽略空值



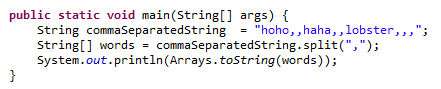
1. 拼接并用指定字符串替换空值。



怎么样，简单吧。

### Spliter的使用

我们有时还经常用一个字符分割字符串，最经常使用的方法是String.split方法，但是这个方法有个问题你知道么？看下面的例子：

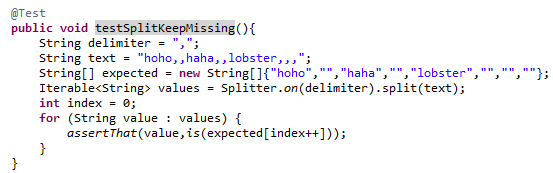


大家猜一下，最后的输出是下面的哪个：



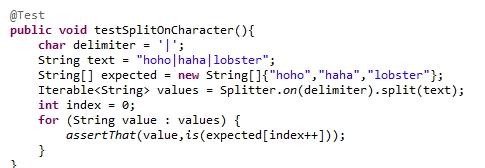
我们一般想要的结果肯定是第二个，但是用String.split分割后得到的数组却是第一个。

下面来看一下Guava Splitter是如何实现的

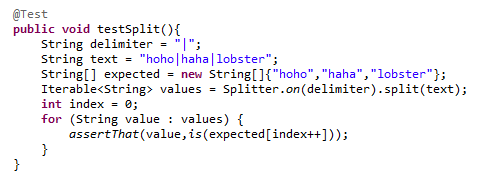


测试结果与我们希望的一致，那下面来看下Splitter有哪些用法：

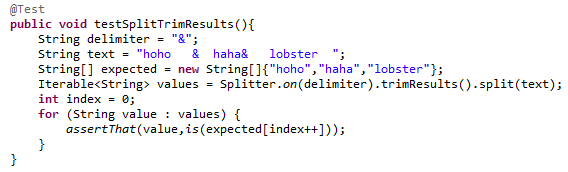
1. 以char做为分割符



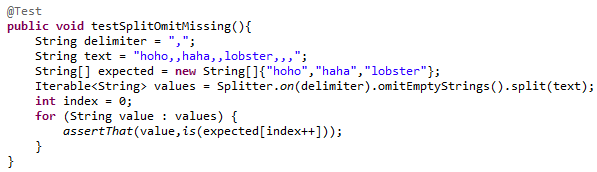
1. 以String做为分割符



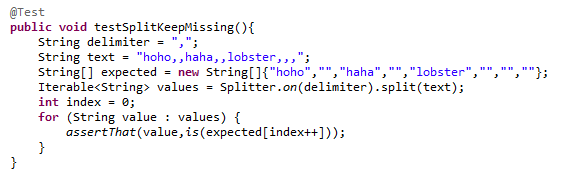
1. 对分割结果进行trim



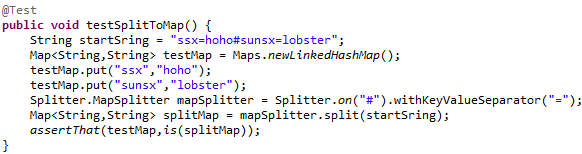
1. 空值忽略



1. 保留空值

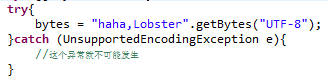


1. 分割字符串并转为Map



Charsets的使用

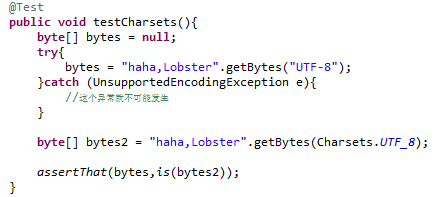
我们在使用String.getBytes(String charset)方法获得某字符集下Byte数组表示时，经常要对一个异常进行处理，代码如下



如上面代码所示，在java 平台中有6种charset是必须支持的，其中一种是UTF-8，所以我们上述异常的处理经常是多余的代码。Guava的Charsets帮我们封装了这6种字符集，使用Charsets，我们的代码将变成下面这样：

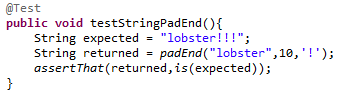


异常处理就不需要了。完整测试代码如下：

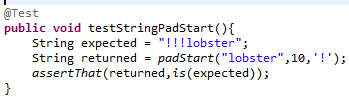


Strings的使用

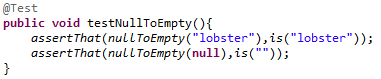
1. 向后补齐



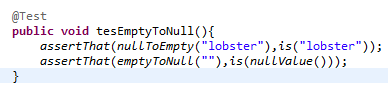
1. 从前补齐



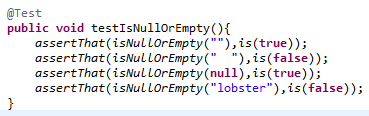
1. 将null变为空字符串



1. 空字符串变为null



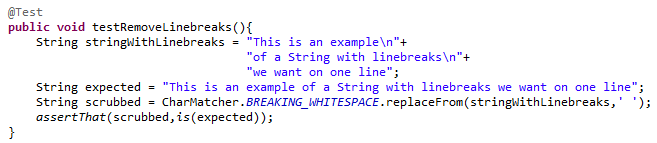
1. 判断是否为null或空字符



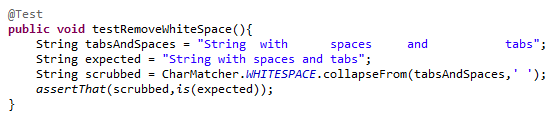
小结，上述的方法中，nullToEmpty和isNullOrEmpty是我们会经常使用的。

CharMatcher的使用

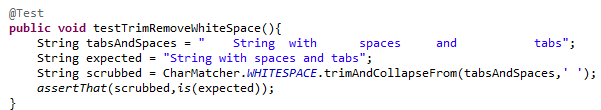
1. 去换行



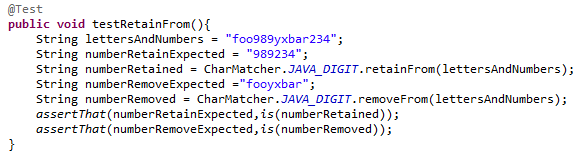
1. 去字符串中间的多余空格



1. 去字符串中间空格并将结果trim



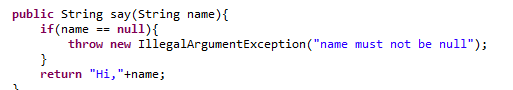
1. 保留数字去掉数字



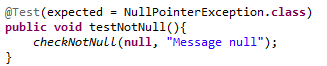
Preconditions的使用

1. 非空校验

我们经常写类似于下面这样的判断条件

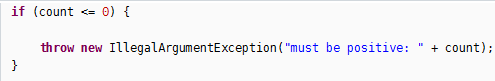


但如果使用Guava的Preconditions的话，我们检查一个空参数的代码将非常简洁

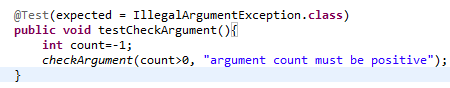


1. 条件检查

我们有时还要做一些验证和条件检查：

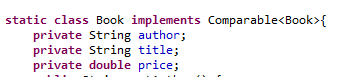


Guava的写法：



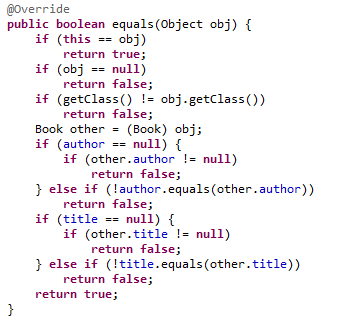
Objects工具的使用

首先我们创建一个Book类

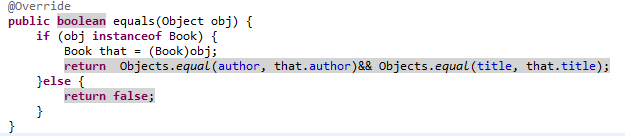


1. Equal方法

正常重写equal方法 的代码如下：

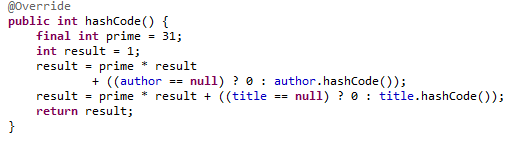


用Guava Objects来写：

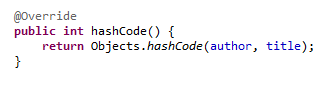


1. hashCode方法

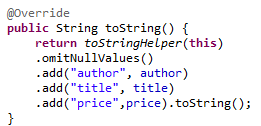
用java正常的写法：



用Guava的Objects：



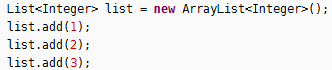
1. toStringHelper方法



Guava Collections使用

在我准备开始介绍Guava Collections前，问大家个问题，多少次你写了像下面这样的代码：





答案留到下面来回答，开始介绍Guava Collections

Collections的创建

1. List的创建

在JAVA5引进了范型后，如果new一个ArrayList的话，代码通常是这样的：



如果用Guava来创建，怎么做呢？



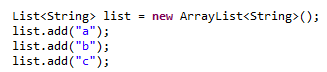
也可以静态引入(import static)



当然了，不止可以创建ArrayList,也可以创建LinkedList



除了new之外，我们平时还想放点数据进去，直接用java的写法：



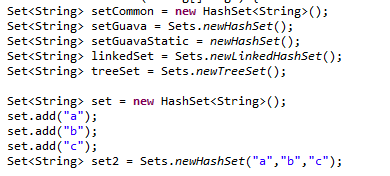
用Lists是这样：



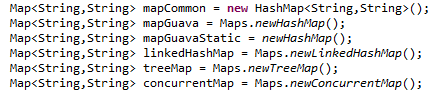
省事多了吧。

1. Set的创建

和List差不多，用法如下：



1. Map的创建



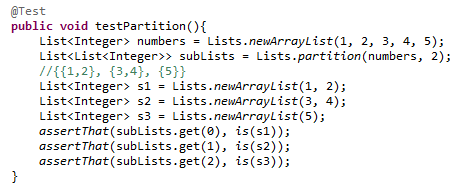
1. 小结

现在回过头来看本章开始前提出的问题，是不是都知道用Guava该怎么写了

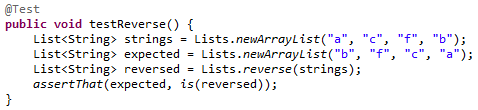
Lists的高级用法

1. partition 分割List

Lists.partition(List, int) 把List按指定大小分割

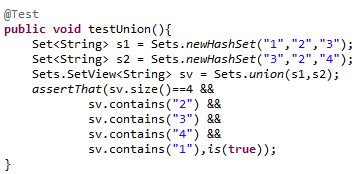


1. reverse反转List

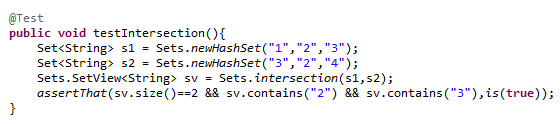


Sets的高级用法

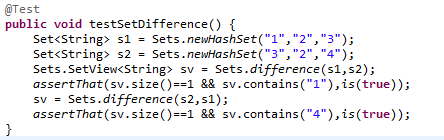
1. Sets.union取两个Set集合的并集。



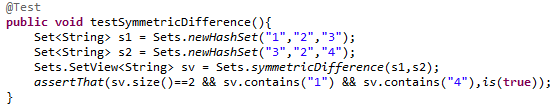
1. Sets.intersection 取两个集合Set的交集



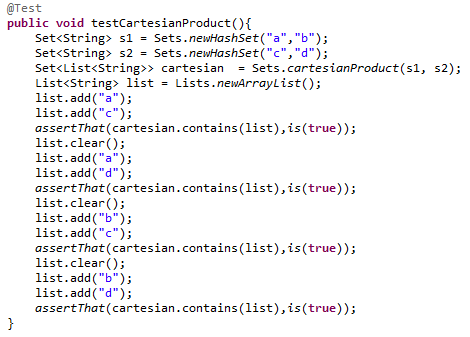
1. Sets.difference 取两个集合Set的差集



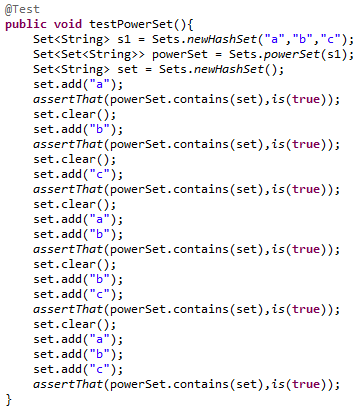
1. Sets.symmetricDifference取两个集合Set的对称差集



1. Sets.cartesianProduct(List<Set>)返回所有集合的笛卡儿积



1. Sets.powerSet(Set) 返回给定集合的所有子集

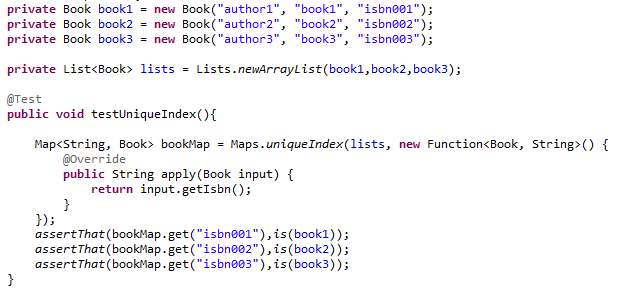


Maps的高级用法

1. uniqueIndex

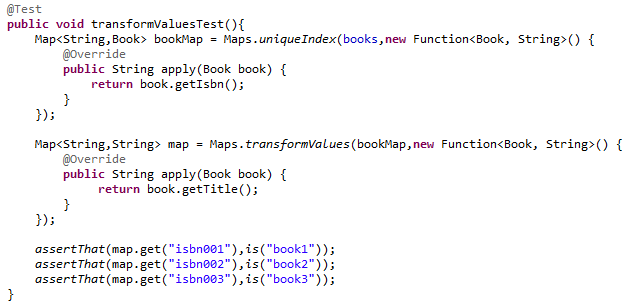
Maps.uniqueIndex(Iterable,Function)通常针对的场景是：有一组对象，它们在某个属性上分别有独一无二的值，而我们希望能够按照这个属性值查找对象。

比方说，我们有一些书，这些书的书号都是独一无二的，我们希望根据书号来查找书。



1. transformValues

按照uniqueIndex我们得到了一个关于book的Map，其中Key是书号，Value是书的实体Book，那么现在我们有个新需求，将这个Map中的Value转为书的title。通常的做法创建一个新的newMap<String,String>，遍历整个Map，然后获得每个book，将ISBN和title分别做为key,value 放入newMap中，好麻烦。。。来看下transformValues怎么做的。



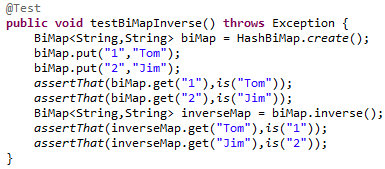
BiMap

BiMap提供了一种新的集合类型，它提供了key和value的双向关联的数据结构。

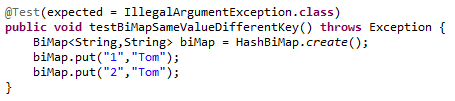
通常情况下，我们在使用Java的Map时，往往是通过key来查找value的，但是如果出现下面一种场景的情况，我们就需要额外编写一些代码了。首先我们来看一下一个序号和人名的Map结构。



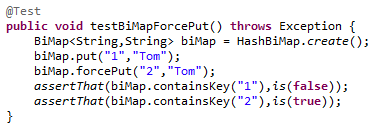
当我们希望通过序号查找人名的时候，很简单。但我们希望通过人名查找序号时，就不得不遍历Map了。这时我们就可以考虑使用Guava中的BiMap了。



在使用BiMap时，会要求Value的唯一性。如果value重复了则会抛出错误：java.lang.IllegalArgumentException，例如：



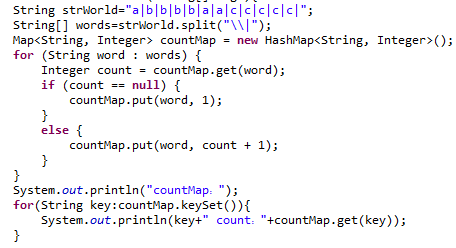
如果我们确实需要插入重复的value值，那可以选择forcePut方法。但是我们需要注意的是前面的key也会被覆盖了。



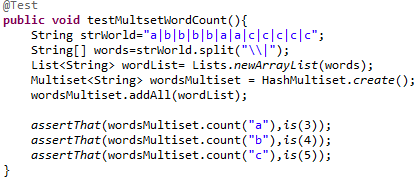
Multiset

Multiset是什么？顾名思义，Multiset和Set的区别就是可以保存多个相同的对象。在JDK中，List和Set有一个基本的区别，就是 List可以包含多个相同对象，且是有顺序的，而Set不能有重复，且不保证顺序（有些实现有顺序，例如LinkedHashSet和SortedSet 等）所以Multiset占据了List和Set之间的一个灰色地带：允许重复，但是不保证顺序。

常见使用场景：Multiset有一个有用的功能，就是跟踪每种对象的数量，所以你可以用来进行数字统计。 常见的普通实现方式如下：



上面的代码实现的功能非常简单，用于记录字符串在数组中出现的次数。这种场景在实际的开发过程还是容易经常出现的，如果使用实现Multiset接口的具体类就可以很容易实现以上的功能需求：



MultiMap

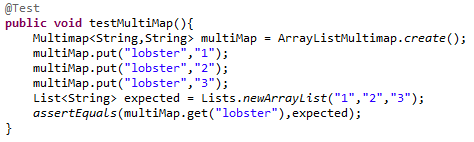
在日常的开发工作中，我们有的时候需要构造像Map<K, List<V>>或者Map<K, Set<V>>这样比较复杂的集合类型的数据结构，以便做相应的业务逻辑处理。例如：



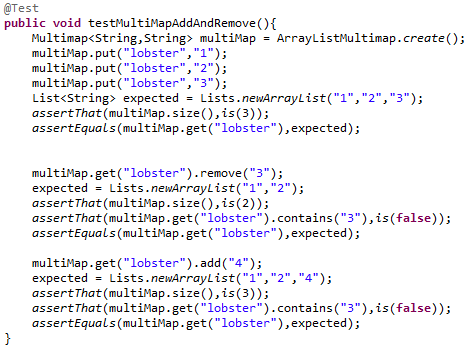
这样的数据结构，自己实现起来太麻烦，你需要检查key是否存在，不存在时则创建一个，存在时在List后面添加上一个。这个过程是比较痛苦的，如果你希望检查List中的对象是否存在，删除一个对象，或者遍历整个数据结构，那么则需要更多的代码来实现。

Guava的Multimap就提供了一个方便地把一个键对应到多个值的数据结构。让我们可以简单优雅的实现上面复杂的数据结构，让我们的精力和时间放在实现业务逻辑上，而不是在数据结构上，下面我们具体来看看Multimap的相关知识点。

上面数据结构用Multimap来实现，代码结构清晰就简单了很多，具体代码如下：



下面看一下如何操作MultiMap



不可变集合

不可变集合，顾名思义就是说集合是不可被修改的。集合的数据项是在创建的时候提供，并且在整个生命周期中都不可改变。

为什么要用immutable对象？immutable对象有以下的优点：

1.对不可靠的客户代码库来说，它使用安全，可以在未受信任的类库中安全的使用这些对象

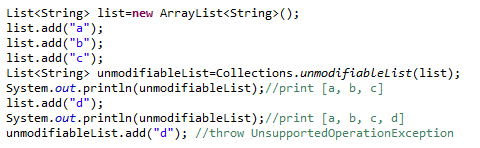
2.线程安全的：immutable对象在多线程下安全，没有竞态条件

3.不需要支持可变性, 可以尽量节省空间和时间的开销. 所有的不可变集合实现都比可变集合更加有效的利用内存 (analysis)

4.可以被使用为一个常量，并且期望在未来也是保持不变的

immutable对象可以很自然地用作常量，因为它们天生就是不可变的对于immutable对象的运用来说，它是一个很好的防御编程（defensive programming）的技术实践。

在JDK中提供了Collections.unmodifiableXXX系列方法来实现不可变集合, 但是存在一些问题，下面我们先看一个具体实例：

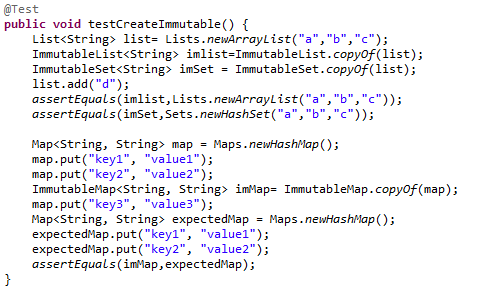


Collections.unmodifiableList实现的不是真正的不可变集合，当原始集合修改后，不可变集合也发生变化。不可变集合不可以修改集合数据，当强制修改时会报错，实例中的最后的add会直接抛出异常。

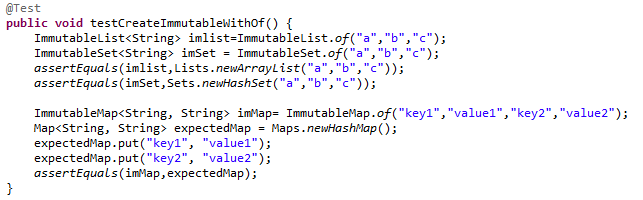
Guava提供了对JDK里标准集合类里的immutable版本的简单方便的实现，以及Guava自己的一些专门集合类的immutable实现。当你不希望修改一个集合类，或者想做一个常量集合类的时候，使用immutable集合类就是一个最佳的编程实践。**注意：每个Guava immutable集合类的实现都拒绝null值。**

来看一下Immutable集合的使用方法，一个immutable集合可以有以下几种方式来创建：

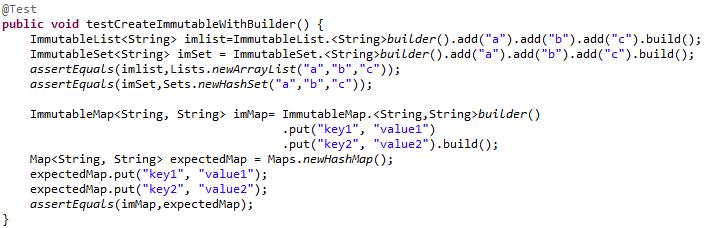
1. 用copyOf方法, 譬如, ImmutableSet.copyOf(set)



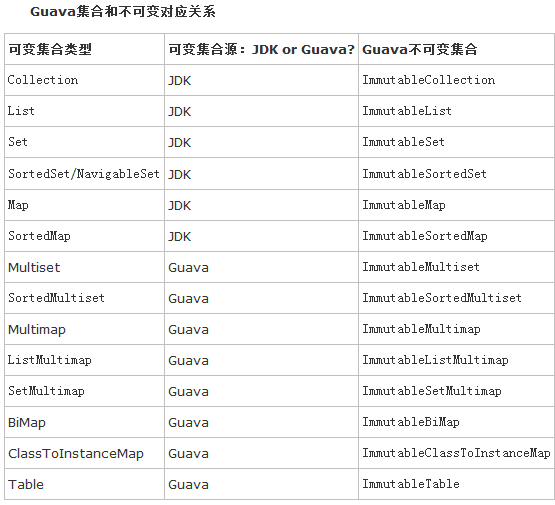
1. 使用of方法，譬如，ImmutableSet.of("a", "b", "c")或者ImmutableMap.of("a", 1, "b", 2)



1. 使用Builder类



1. 不可变集合不只上述的List,Set,Map，还有很多JDK的集合和Guava集合都有各自对应的不可变集合，下面一张图列出了这些对应关系

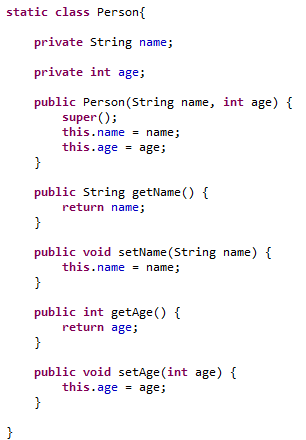


Guava的函数式编程

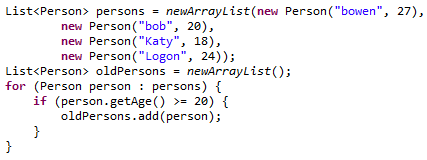
Google Collections给我们带来了一对非常优雅的东东，叫做:Functions and Predicates! 和你使用的scala一样有神奇的地方，现在你可以使用在no-functional 的java身上了。你可以在com.google.common.base包里找到这些（更多）。

Predicates 过滤

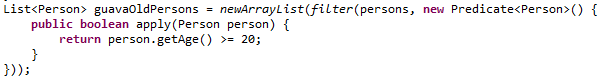
我们先创建一个简单的Person类



我们创建若干个Person对象并放到集合里，这里我们的需求是选取其中年龄大于20的人，那通常的写法是下面这样的



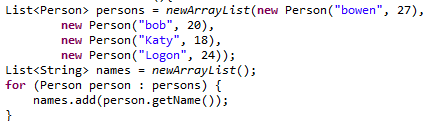
这就是典型的filter模式。filter即从一个集合中根据一个条件筛选元素。其中person.getAge() >=20就是这个条件。Guava为这种模式提供了一个filter的方法。所以我们可以这样写：



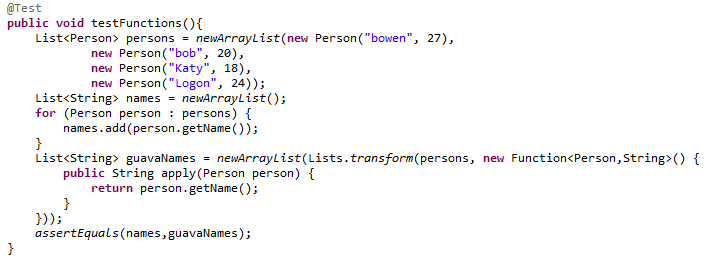
Functions 转换

列表操作还有另一个常见的模式，就是将数组中的所有元素映射为另一种元素的列表，这就是map pattern。在前面Maps的高级用法中已经举过这种用法的例子，这里再稍稍举个简单例子。

还是上节中的Person，还是先创建几个Person，放入一个集合中，这时我们的需求是得到这些Person的名子。我们十有八九都这么写：



Guava已经给我们提供了这种Pattern的结果办法，那就是使用transform方法：



Guava Concurrency