Guava学习之Lists

Lists类主要提供了对List类的子类构造以及操作的静态方法。在Lists类中支持构造ArrayList、LinkedList以及newCopyOnWriteArrayList对象的方法。
其中提供了以下构造ArrayList的函数：下面四个构造一个ArrayList对象，但是不显式的给出申请空间的大小：

```
newArrayList()
newArrayList(E... elements)
newArrayList(Iterable<? extends E> elements)
newArrayList(Iterator<? extends E> elements)
```

以下两个函数在构造ArrayList对象的时候给出了需要分配空间的大小:

```
newArrayListWithCapacity(int initialArraySize)
newArrayListWithExpectedSize(int estimatedSize)
```

如果你事先知道元素的个数，可以用newArrayListWithCapacity函数；如果你不能确定元素的个数，可以用newArrayListWithExpectedSize函数，在newArrayListWithExpectedSize函数里面调用了computeArrayListCapacity(int arraySize)函数，其实现如下：

```
@VisibleForTesting static int computeArrayListCapacity(int arraySize) {
  checkArgument(arraySize >= 0);

  // TODO(kevinb): Figure out the right behavior, and document it
  return Ints.saturatedCast(5L + arraySize + (arraySize / 10));
 }
```

返回的容量大小为5L + arraySize + (arraySize / 10)，当arraySize比较大的时候，给定大小和真正分配的容量之比为10/11。
Lists类还支持构造LinkedList、newCopyOnWriteArrayList对象，其函数接口为：

```
newLinkedList()
newLinkedList(Iterable<? extends E> elements)
```

```
newCopyOnWriteArrayList()
newCopyOnWriteArrayList(Iterable<? extends E> elements)
```

我们还可以将两个（或三个）类型相同的数据存放在一个list中，这样可以传入到只有一个参数的函数或者需要减少参数的函数中，这些函数如下：

```
asList(@Nullable E first, E[] rest)
asList(@Nullable E first, @Nullable E second, E[] rest)
```

Lists类中transform函数可以根据传进来的function对fromList进行相应的处理，并将处理得到的结果存入到新的list对象中，这样有利于我们进行分析，函数接口如下：

```
public static <F, T> List<T> transform(
    List<F> fromList, Function<? super F, ? extends T> function)
```

使用例子：

```
Function<String, Integer> strlen = new Function<String, Integer>() {
  public Integer apply(String from) {
    Preconditions.checkNotNull(from);
    return from.length();
  }
};
List<String> from = Lists.newArrayList("abc", "defg", "hijkl");
List<Integer> to = Lists.transform(from, strlen);
for (int i = 0; i < from.size(); i++) {
  System.out.printf("%s has length %d₩n", from.get(i), to.get(i));
}
```

```
Function<String, Boolean> isPalindrome = new Function<String, Boolean>() {
  public Boolean apply(String from) {
    Preconditions.checkNotNull(from);
    return new StringBuilder(from).reverse().toString().equals(from);
  }
```

```
};
from = Lists.newArrayList("rotor", "radar", "hannah", "level", "botox");
List<Boolean> to1 = Lists.transform(from, isPalindrome);
for (int i = 0; i < from.size(); i++) {
   System.out.printf("%s is%sa palindrome₩n", from.get(i), to1.get(i) ? " " : " NOT ");
}
// changes in the "from" list are reflected in the "to" list
System.out.printf("₩nnow replace hannah with megan...₩n₩n");
from.set(2, "megan");
for (int i = 0; i < from.size(); i++) {
   System.out.printf("%s is%sa palindrome₩n", from.get(i), to1.get(i) ? " " : " NOT ");
}
```

Lists还可以将传进来的String或者CharSequence分割为单个的字符，并存入到一个新的List对象中返回，如下：

```
ImmutableList<Character> wyp = Lists.charactersOf("wyp");
System.out.println(wyp);
```

将List对象里面的数据顺序反转可以用reverse函数实现，取得List对象里面的子序列可以用subList函数实现。更多的实现可以参看其源码。（完）