Guava学习之SetMultimap

Guava学习之SetMultimap

SetMultimap及其子类的继承图如上所示。

SetMultimap是一个接口，继承自Multimap接口，同昨天说的ListMultimap接口类似，它也定义了所有继实现自SetMultimap的子类定义了一些共有的方法签名。SetMultimap接口并没有定义自己特有的方法签名，里面所有的方法都是重写了Multimap接口中的声明，只是将Multimap接口中返回Collection类型的函数修改成返回Set类型。从名字可以看出，SetMultimap接口继承自Multimap接口，它也是存储key和value键值对，只不过这里存储value是用Set实现的，而昨天说到的ListMultimap存储value是用List实现的，因为List和Set本身的一些特点，导致ListMultimap子类的实例中同一个key可以存储相同的value值；而SetMultimap子类的实例中同一个key不能存储相同的value。仔细对比SetMultimap接口和ListMultimap接口可以发现，SetMultimap接口比ListMultimap接口多重写了Multimap接口中的一个方法，那就是entries()，它被重写的函数原型如下：

```
@Override
Set<Map.Entry<K, V>> entries();
```

而entries()方法在Multimap接口中的原型如下：

```
Collection<Map.Entry<K, V>> entries();
```

为什么ListMultimap接口不需要重写Multimap接口中的entries方法呢？原因很简单，以Set实现的Multimap中同一个key不能存储相同的value，这是什么意思？你看看entries()函数返回类型中存储的是Map.Entry的实体，假如SetMultimap接口不重写Multimap接口中的entries()方法，那会发生什么事？存在两个相同的Entry这是不允许的，将Entry放在Set中就可以避免出现两个相同的Entry；而ListMultimap接口允许相同的key存在相同的value，也就是允许存在相同的Entry，所以用Collection来存储很恰当。来看下面的例子：

```
TreeMultimap treeMultimap = TreeMultimap.create();
treeMultimap.put("wyp", "hao");
```

```
treeMultimap.put("wyp", "hao");
treeMultimap.put("wyp1", "hao");
Set<Map.Entry> entries1 = treeMultimap.entries();
System.out.println(entries1);

ArrayListMultimap<String, String> arrayListMultimap =
    ArrayListMultimap.create();
arrayListMultimap.put("wyp", "hao");
arrayListMultimap.put("wyp", "hao");
arrayListMultimap.put("wyp1", "hao");
Collection<Map.Entry<String, String>> entries = arrayListMultimap.entries();
System.out.println(entries);
```

程序输出的结果如下：

```
[wyp=hao, wyp1=hao]
[wyp=hao, wyp=hao, wyp1=hao]
```

很合理！（未完，待续）