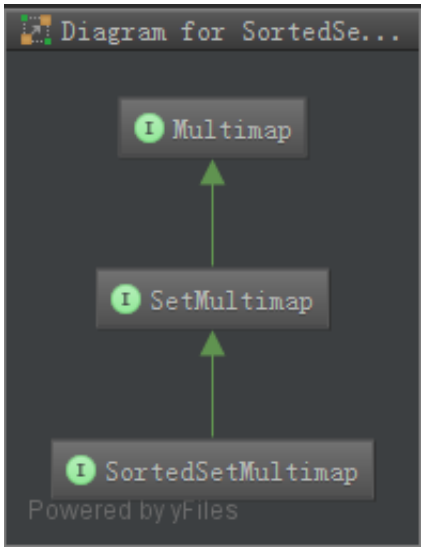
Gauva学习之SortedSetMultimap



Gauva学习之SortedSetMultimap

SortedSetMultimap是一个接口，它的继承关系如上所示。继承了SortedSetMultimap接口的类中key所对应的value是有序的。因为SortedSetMultimap的子类中key所对应的value是有序的，所以SortedSetMultimap重写了SetMultimap中的以下四个方法：

```
@Override
SortedSet<V> get(@Nullable K key);

@Override
SortedSet<V> removeAll(@Nullable Object key);

@Override
SortedSet<V> replaceValues(K key, Iterable<? extends V> values);

@Override Map<K, Collection<V>> asMap();
```

前三个函数都是将SetMultimap接口中相应的函数返回类型（Set）修改成了SortedSet类型。而SortedSet接口是继承自Set的，只不过SortedSet的子类可以保证其中的value是有序的，而SortedSetMultimap接口就是需要保证key中所对应的value是有序的，所以可以使用SortedSet。

但是既然asMap()也是重载了SetMultimap中相应的函数，为什么它就返回Map<k, Collection>，而不返回Map<k, SortedSet>呢？这是因为asMap()函数能够保证其返回的类型为一定是SortedSet的子类。看看代码就知道了，TreeMultimap类是SetMultimap的实现类，它的asMap()实现如下：

```
@Override
@GwtIncompatible("NavigableMap")
public NavigableMap<K, Collection<V>> asMap() {
        return (NavigableMap<K, Collection<V>>) super.asMap();
}
```

而super.asMap();最上层类（AbstractMultimap接口）的实现如下：

```
private transient Map<K, Collection<V>> asMap;

@Override
public Map<K, Collection<V>> asMap() {
   Map<K, Collection<V>> result = asMap;
   return (result == null) ? asMap = createAsMap() : result;
}
```

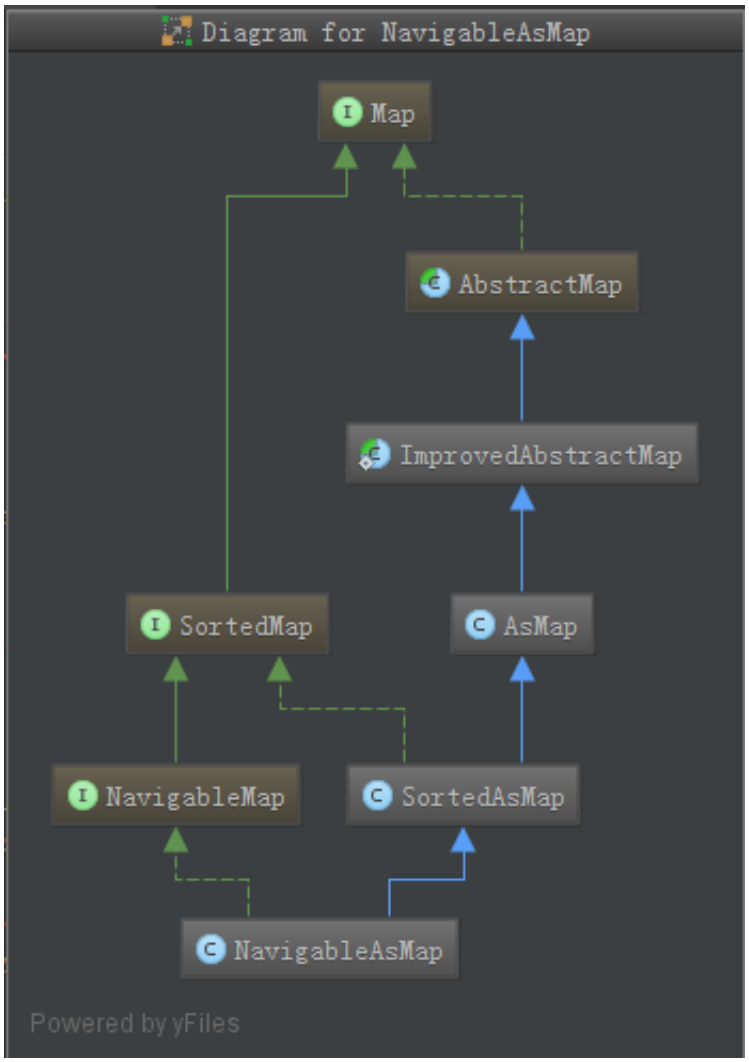
而createAsMap()函数在AbstractMultimap中的定义如下：

```
abstract Map<K, Collection<V>> createAsMap();
```

这是一个抽象的函数，需要其子类（这里是指TreeMultimap）实现，那么TreeMultimap中对createAsMap()函数是怎么实现的呢？看下它的实现代码：

```
@Override
@GwtIncompatible("NavigableMap")
NavigableMap<K, Collection<V>> createAsMap() {
   return new NavigableAsMap(backingMap());
}
```

我们不需要知道NavigableAsMap类的实现方式，只需要知道NavigableAsMap类的继承关系，如下所示：



Gauva学习之SortedSetMultimap

从上面NavigableAsMap类的继承关系图可以看出，NavigableAsMap类是SortedMsp的实现类，它能保证的Map中的key和value都是有序的。所以，SortedSetMultimap中asMap()函数返回类型为Map<k , Collection>。

SortedSetMultimap接口中还定义了valueComparator函数，其原型如下：

```
Comparator<? super V> valueComparator();
```

valueComparator函数返回对multimap中value排序的对象，如果返回为null，则说明multimap中value是按照自然排序的。