回到首页

# 芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2019-12-11
JDK

# 精尽 JDK 源码解析 —— 集合（五）哈希集合 HashSet

## 1. 概述

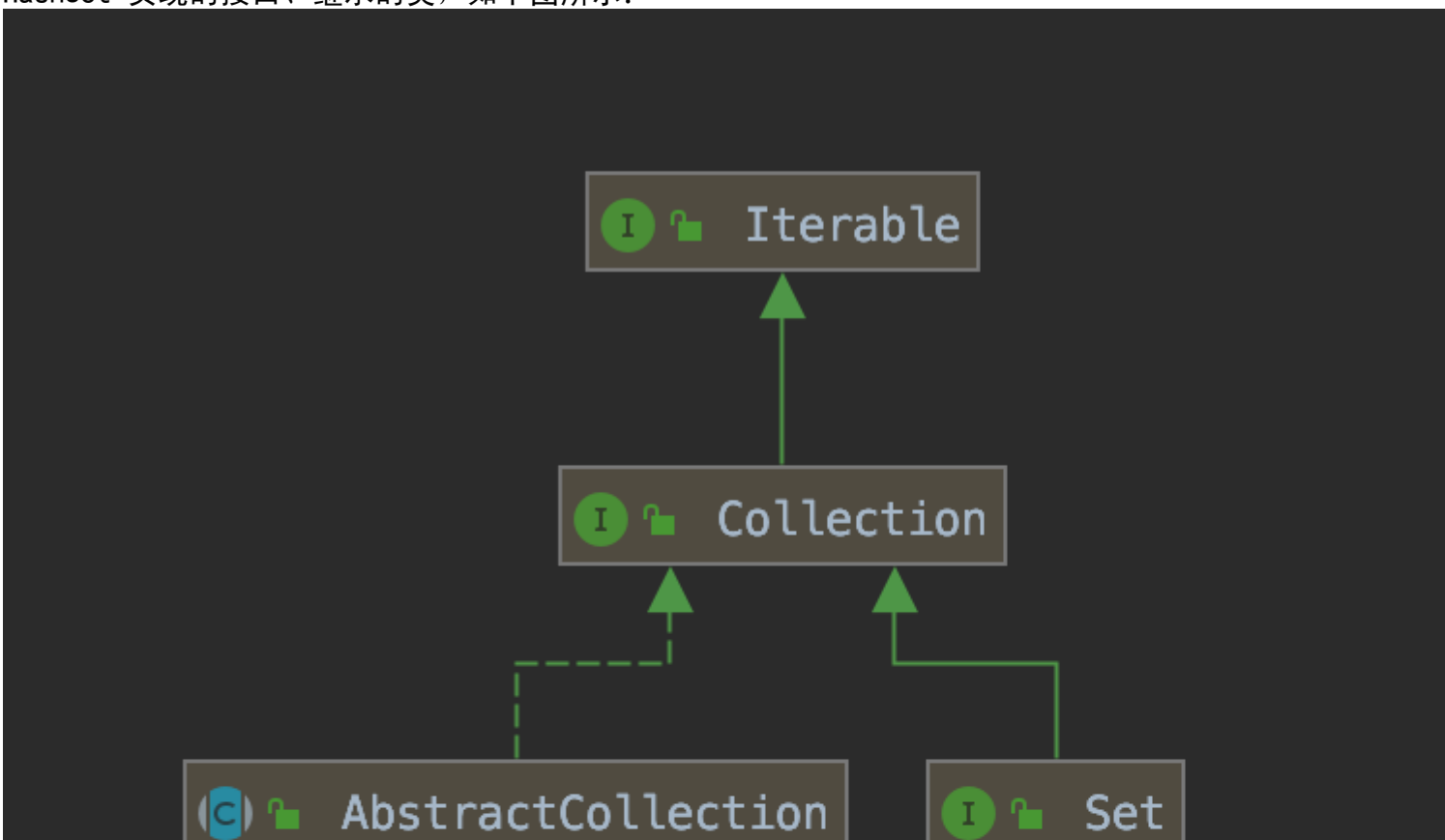
芳芳：突然有点词穷，不知道怎么介绍，嘿嘿。

HashSet ，基于 HashMap 的 Set 实现类。在业务中，如果我们有排重的需求，一般会考虑使用 HashSet 。

在 Redis 提供的 Set 数据结构，不考虑编码的情况下，它是基于 Redis 自身的 Hash 数据结构实现的。这点，JDK 和 Redis 是相同的。 感兴趣的胖友，看完本文后，可以自己去撸一撸。

## 2. 类图

HashSet 实现的接口、继承的类，如下图所示：



实现 java.util.Set 接口，并继承 java.util.AbstractSet 抽像类。
实现 java.io.Serializable 接口。
实现 java.lang.Cloneable 接口。

# 3. 属性

HashSet 只有一个属性，那就是 map 。代码如下：

```
// HashSet.java

private transient HashMap<E, Object> map;
```

map 的 key ，存储 HashSet 的每个 key 。

map 的 value ，因为 HashSet 没有 value 的需要，所以使用一个统一的 PRESENT 即可。代码如下：

```
// HashSet.java

// Dummy value to associate with an Object in the backing Map
private static final Object PRESENT = new Object();
```

# 4. 构造方法

HashSet 一共有 5 个构造方法，代码如下：

```
// HashSet.java

public HashSet() {
    map = new HashMap<>();
}

public HashSet(Collection<? extends E> c) {
    // 最小必须是 16 。
    // (int) (c.size()/.75f) + 1 避免扩容
    map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));
    // 批量添加
    addAll(c);
}

public HashSet(int initialCapacity, float loadFactor) {
    map = new HashMap<>(initialCapacity, loadFactor);
}

public HashSet(int initialCapacity) {
    map = new HashMap<>(initialCapacity);
}

HashSet(int initialCapacity, float loadFactor, boolean dummy) {
    map = new LinkedHashMap<>(initialCapacity, loadFactor); // 注意，这种情况下的构造方法，创建的是 LinkedHashMap 对象
}
```

在构造方法中，会创建 HashMap 或 LinkedHashMap 对象。

# 5. 添加单个元素

#add(E e) 方法，添加单个元素。代码如下：

```
// HashSet.java

public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
```

map 的 value 值，就是我们看到的 PRESENT 。

而添加多个元素，继承自 AbstractCollection 抽象类，通过 #addAll(Collection<? extends E> c) 方法，代码如下：

```
// AbstractCollection.java

public boolean addAll(Collection<? extends E> c) {
    boolean modified = false;
    // 遍历 c 集合，逐个添加
    for (E e : c)
        if (add(e))
            modified = true;
    return modified;
}
```

在方法内部，会逐个调用 #add(E e) 方法，逐个添加单个元素。

# 6. 移除单个元素

#remove(Object key) 方法，移除 key 对应的 value ，并返回该 value 。代码如下：

```
// HashSet.java

public boolean remove(Object o) {
    return map.remove(o) == PRESENT;
}
```

# 7. 查找单个元素

#contains(Object key) 方法，判断 key 是否存在。代码如下：

```
// HashSet.java

public boolean contains(Object o) {
```

```
        return map.containsKey(o);
    }
```

芮芮：后面的内容，快速看即可。

# 8. 转换成数组

#toArray(T[] a) 方法，转换出 key 数组返回。代码如下：

```
    // HashSet.java

    @Override
    public Object[] toArray() {
        return map.keysToArray(new Object[map.size()]);
    }

    @Override
    public <T> T[] toArray(T[] a) {
        return map.keysToArray(map.prepareArray(a));
    }
```

# 9. 清空

#clear() 方法，清空 HashSet 。代码如下：

```
    // HashSet.java

    public void clear() {
        map.clear();
    }
```

# 10. 序列化

#writeObject(ObjectOutputStream s) 方法，序列化 HashSet 对象。代码如下：

```
    // HashSet.java

    @java.io.Serial
    private void writeObject(java.io.ObjectOutputStream s)
        throws java.io.IOException {
        // Write out any hidden serialization magic
        // 写入非静态属性、非 transient 属性
        s.defaultWriteObject();

        // Write out HashMap capacity and load factor
        // 写入 map table 数组大小
        s.writeInt(map.capacity());
        // 写入 map 加载因子
```

```
        s.writeFloat(map.loadFactor());

        // Write out size
        // 写入 map 大小
        s.writeInt(map.size());

        // Write out all elements in the proper order.
        // 遍历 map ，逐个 key 序列化
        for (E e : map.keySet())
            s.writeObject(e);
    }
```

# 11. 反序列化

#readObject(ObjectInputStream s) 方法，反序列化成 HashSet 对象。代码如下：

```
    // HashSet.java

    @java.io.Serial
    private void readObject(java.io.ObjectInputStream s)
        throws java.io.IOException, ClassNotFoundException {
        // Read in any hidden serialization magic
        // 读取非静态属性、非 transient 属性
        s.defaultReadObject();

        // Read capacity and verify non-negative.
        // 读取 HashMap table 数组大小
        int capacity = s.readInt();
        // 校验 capacity 参数
        if (capacity < 0) {
            throw new InvalidObjectException("Illegal capacity: " +
                                             capacity);
        }

        // Read load factor and verify positive and non NaN.
        // 获得加载因子 loadFactor
        float loadFactor = s.readFloat();
        // 校验 loadFactor 参数
        if (loadFactor <= 0 || Float.isNaN(loadFactor)) {
            throw new InvalidObjectException("Illegal load factor: " +
                                             loadFactor);
        }

        // Read size and verify non-negative.
        // 读取 key-value 键值对数量 size
        int size = s.readInt();
        // 校验 size 参数
        if (size < 0) {
            throw new InvalidObjectException("Illegal size: " +
                                             size);
        }

        // Set the capacity according to the size and load factor ensuring that
        // the HashMap is at least 25% full but clamping to maximum capacity.
        // 计算容量
        capacity = (int) Math.min(size * Math.min(1 / loadFactor, 4.0f),
```

```
                HashMap.MAXIMUM_CAPACITY);

        // Constructing the backing map will lazily create an array when the first element is
        // added, so check it before construction. Call HashMap.tableSizeFor to compute the
        // actual allocation size. Check Map.Entry[].class since it's the nearest public type to
        // what is actually created.
        SharedSecrets.getJavaObjectInputStreamAccess()
                     .checkArray(s, Map.Entry[].class, HashMap.tableSizeFor(capacity)); // 不知道作甚，哈哈哈。

        // Create backing HashMap
        // 创建 LinkedHashMap 或 HashMap 对象
        map = (((HashSet<?>)this) instanceof LinkedHashSet ?
               new LinkedHashMap<>(capacity, loadFactor) :
               new HashMap<>(capacity, loadFactor));

        // Read in all elements in the proper order.
        // 遍历读取 key 键，添加到 map 中
        for (int i=0; i<size; i++) {
            @SuppressWarnings("unchecked")
                E e = (E) s.readObject();
            map.put(e, PRESENT);
        }
    }
```

# 12. 克隆

#clone() 方法，克隆 HashSet 对象。代码如下：

```
    // HashSet.java

    public Object clone() {
        try {
            // 调用父方法，克隆创建 newSet 对象
            HashSet<E> newSet = (HashSet<E>) super.clone();
            // 可控 mao 属性，赋值给 newSet
            newSet.map = (HashMap<E, Object>) map.clone();
            // 返回
            return newSet;
        } catch (CloneNotSupportedException e) {
            throw new InternalError(e);
        }
    }
```

# 13. 获得迭代器

#iterator() 方法，获得迭代器。代码如下：

```
    // HashSet.java

    public Iterator<E> iterator() {
        return map.keySet().iterator();
    }
```

# 666. 彩蛋

总的来说，比较简单，相信胖友一会会时间就已经看完了。

可能偶尔我们会使用到 LinkedHashSet ，它是 HashSet 的子类，胖友自己去看。更加简单~

关于 HashSet 的总结，只有一句话：HashSet 是基于 HashMap 的 Set 实现类。

文章目录


1. 1. 概述
2. 2. 类图
3. 3. 属性
4. 4. 构造方法
5. 5. 添加单个元素
6. 6. 移除单个元素
7. 7. 查找单个元素
8. 8. 转换成数组
9. 9. 清空
10. 10. 序列化
11. 11. 反序列化
12. 12. 克隆
13. 13. 获得迭代器
14. 14. 666. 彩蛋





回到首页