JCF API的设计

2021-02-25 李力

为什么分享JCF?

- 1 研究API的框架，值得投入大量时间研究API.(Guava 为例子)
- 2 理解接口，继承，多态，抽象，复用的框架
- 3 理解数据结构和算法结合的框架
- 4 理解一个框架的设计点
- 5 学习如何用JCF设计可扩展性的代码

- 问：你能写出NIO代码吗?

我们回到1997年

- 简单时代
  - Java只有Vector, Hashtable& Enumeration
  - 因为平台增长，需要更多
- 野蛮人在敲门
  - JGL是STL的翻译
  - 拥有130个类和接口
  - JGL的设计者非常想把它放到JDK中
- Josh Bloch重新设计

回到1998年

- JavaOne 1998讨论Collection
- 没人知道一个Collection框架是什么
  - 不知道为什么需要Collection
- 会议讨论
  - 解释概念
  - 教学

集合历史

Release, Year	Changes
JDK 1.0, 1996	Java Released: `Vector`, `Hashtable`, `Enumeration`
JDK 1.1, 1996	(No API changes)
J2SE 1.2, 1998	Collections framework added
J2SE 1.3, 2000	(No API changes)
J2SE 1.4, 2002	`LinkedHash{Map,Set}`, `IdentityHashSet`, 6 new algorithms
J2SE 5.0, 2004	Generics, for-each, enums: generified everything, `Iterable` `Queue`, `Enum{Set,Map}`, concurrent collections
Java 6, 2006	`Deque`, `Navigable{Set,Map}`, `newSetFromMap`, `asLifoQueue`
Java 7, 2011	No API changes. Improved sorts & defensive hashing
Java 8, 2014	Lambdas (+ streams and internal iterators)

什么是Collection

- 对元素进行分组的对象
- 主要作用
  - 数据存取
  - 数据传输
- 相关例子
  - java.util.Vector
  - java.util.Hashtable
  - array

什么是Collection Freamwork

- 统一架构
  - 接口-实现独立
  - 实现-复用数据结构
  - 算法-复用函数
- 经典实例
  - STL
  - Smalltalk collections

好处

- 减轻编程成本
- 提高程序速度和质量
- 互不相关api之间的互操作性
- 减少学习新API的成本
- 减少重新设计API成本
- 软件复用

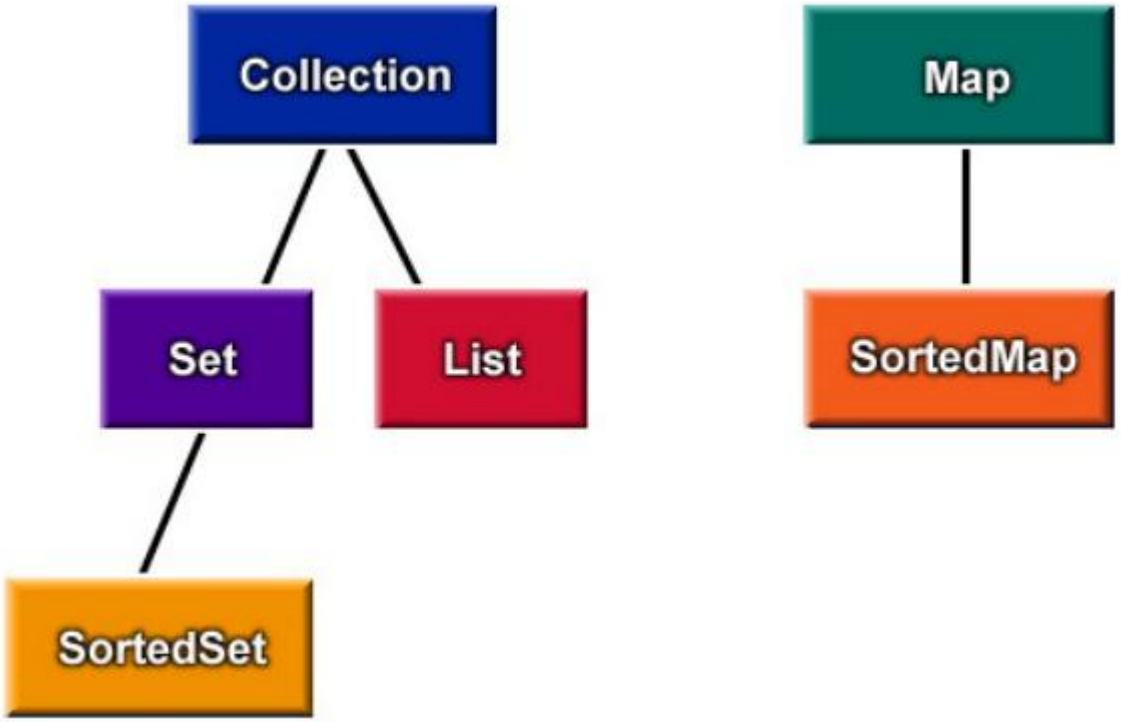
设计目标

- 小巧和简单
- 功能强大
- 方便扩展
- 和已经存在collection兼容
- API学习成本低

架构概览

- 1 核心接口
- 2 通用实现
- 3 包装实现
- 4 抽象实现
- 5 算法

核心接口



为什么设计这些接口？

- 每个接口区别是什么？源码导读
- 为什么需要投入大量时间在API研究上？

Collection接口

```
public interface Collection {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(Object element);    // Optional
    boolean remove(Object element); // Optional
    Iterator iterator();

    Object[] toArray();
    Object[] toArray(Object a[]);

    // Bulk Operations
    boolean containsAll(Collection c);
    boolean addAll(Collection c);     // Optional
    boolean removeAll(Collection c);  // Optional
    boolean retainAll(Collection c);  // Optional
    void clear();                     // Optional
}
```

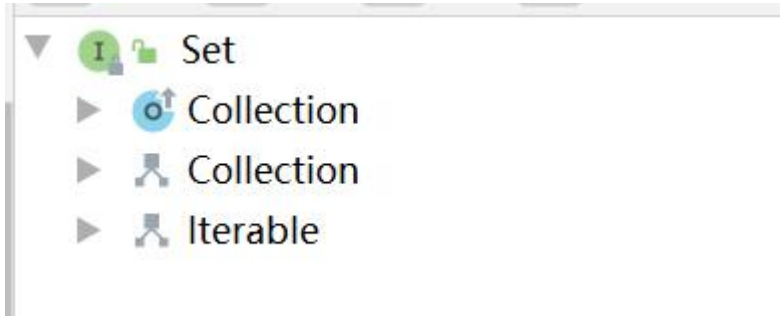
Iterator接口

- 替换Enumeration接口
  - 新加remove方法
  - 提升名字

- 源码对比
  - Iterator和Enumeration
  - Iterator和Spliterator
  - Iterator和Iterable

Set接口

- 没有新加任何方法Collection
- 新加限制：无重复元素
- 强制equals和hashCode计算



Set惯例

```
Set s1, s2;

boolean isSubset = s1.containsAll(s2);

Set union = new HashSet<>(s1);
union.addAll(s2);

Set intersection = new HashSet(s1);
intersection.retainAll(s2);

Set difference = new HashSet(s1);
difference.removeAll(s2);

Collection c;
Collection noDups = new HashSet(c);
```

List接口

```
public interface List extends Collection {
    Object get(int index);
    Object set(int index, Object element);   // Optional
    void add(int index, Object element);     // Optional
    Object remove(int index);                // Optional
    boolean addAll(int index, Collection c); // Optional
    int indexOf(Object o);
    int lastIndexOf(Object o);

    List subList(int from, int to);

    ListIterator listIterator();
    ListIterator listIterator(int index);
}
```

List例子

```
public static void swap(List a, int i, int j) {
    Object tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}

 private static Random r = new Random();

 public static void shuffle(List a) {
     for (int i = a.size(); i > 1; i--)
         swap(a, i - 1, r.nextInt(i));
 }
```

List惯例

```
List a, b;

// Concatenate two lists
a.addAll(b);

// Range-remove
a.subList(from, to).clear();

// Range-extract
List partView = a.subList(from, to);
List part = new ArrayList(partView);
partView.clear();
```

Map接口

```
public interface Map {
    int size();
    boolean isEmpty();
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    Object get(Object key);
    Object put(Object key, Object value);   // Optional
    Object remove(Object key);              // Optional
    void putAll(Map t);  // Optional
    void clear();        // Optional
    // Collection Views
    public Set keySet();
    public Collection values();
    public Set entrySet();
}
```

Map惯例

```
// Iterate over all keys in Map m
Map< m;
for (iterator i = m.keySet().iterator(); i.hasNext(); )
    System.out.println(i.next());

// "Map algebra"
Map a, b;
boolean isSubMap = a.entrySet().containsAll(b.entrySet());
Set commonKeys = new HashSet(a.keySet()).retainAll(b.keySet

//Remove keys from a that have mappings in b
a.keySet().removeAll(b.keySet());
```

通用实现

- 统一命名和行为

General-purpose Implementations

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

选择一个实现

- Set
  - HashSet -- O(1) 访问，无序
  - TreeSet -- O(log n) 访问，有序
- Map
  - – HashMap -- (HashSet)
  - – TreeMap -- (TreeSet)
- List
  - ArrayList -- O(1) random access, O(n) insert/remove
  - LinkedList -- O(n) random access, O(1) insert/remove;
    - Use for queues and deques(不在这么用了)

实现行为

- Fail-fast iterator
- Null元素，key，value允许
- 不是线程安全的

同步包装

- 匿名实现，每个核心接口一个
- 静态工厂
- 线程安全的新实现
- 必须手动同步迭代

同步包装例子

Synchronization Wrapper Example

```
Set s = Collections.synchronizedSet(new HashSet());

        ...

s.add("wombat");  // Thread-safe

        ...

synchronized(s) {
    Iterator i = s.iterator(); // In synch block!
    while (i.hasNext())
        System.out.println(i.next());
}
```

便利实现

- Arrays.asList(Object[] a)　（View）
- EMPTY_SET, EMPTY_LIST, EMPTY_MAP
- singleton(Object o) (不可变)
- nCopies(Object o) (不可变) Bag!

自定义实现其他想法

- 持久化
- 高并发
- 高性能，特定目的
- 节省空间
- 便利类

自定义实现例子

- 通过抽象实现非常简单

```
// List adapter for primitive int array
public static List intArrayList(int[] a) {
    return new AbstractList() {
        public Integer get(int i) {
            return new Integer(a[i]);
        }

        public int size() { return a.length; }

        public Object set(int i, Integer e) {
            int oldVal = a[i];
            a[i] = e.intValue();
            return new Integer(oldVal);
        }
    };
}
```

复用算法

- static void sort(List list);
- static int binarySearch(List list, Object key);
- static Object min(Collection coll);
- static Object max(Collection coll);
- static void fill(List list, Object e);
- static void copy(List dest, List src);
- static void reverse(List list);
- static void shuffle(List list);

源码结构导读

- HashMap
- LinkedList

强大扩展性

- Apache common Collection
- Guava Collection

参考

- https://docs.oracle.com/javase/tutorial/collections/
- https://docs.oracle.com/javase/8/docs/technotes/guides/collections/reference.html
- https://docs.oracle.com/javase/8/docs/technotes/guides/collections/designfaq.html