概览

容器，就是可以容纳其他Java对象的对象。*Java Collections Framework（JCF）*为Java开发者提供了通用的容器，其始于JDK 1.2，优点是：

- 降低编程难度
- 提高程序性能
- 提高API间的互操作性
- 降低学习难度
- 降低设计和实现相关API的难度
- 增加程序的重用性

Java容器里只能放对象，对于基本类型（int, long, float, double等），需要将其包装成对象类型后（Integer, Long, Float, Double等）才能放到容器里。很多时候拆包装和解包装能够自动完成。这虽然会导致额外的性能和空间开销，但简化了设计和编程。

泛型（Generics）

Java容器能够容纳任何类型的对象，这一点表面上是通过泛型机制完成，Java泛型不是什么神奇的东西，只是编译器为我们提供的一个"语法糖"，泛型本身并不需要Java虚拟机的支持，只需要在编译阶段做一下简单的字符串替换即可。实质上Java的单继承机制才是保证这一特性的根本，因为所有的对象都是Object的子类，容器里只要能够存放Object对象就行了。 事实上，所有容器的内部存放的都是Object对象，泛型机制只是简化了编程，由编译器自动帮我们完成了强制类型转换而已。JDK 1.4以及之前版本不支持泛型，类型转换需要程序员显式完成。

```
//JDK 1.4 or before
ArrayList list = new ArrayList();
list.add(new String("Monday"));
list.add(new String("Tuesday"));
list.add(new String("Wensday"));
for(int i = 0; i < list.size(); i++){
    String weekday = (String)list.get(i);//显式类型转换
    System.out.println(weekday.toUpperCase());
}
```

```
//JDK 1.5 or latter
ArrayList<String> list = new ArrayList<String>();//参数化类型
list.add(new String("Monday"));
list.add(new String("Tuesday"));
list.add(new String("Wensday"));
for(int i = 0; i < list.size(); i++){
    String weekday = list.get(i);//隐式类型转换，编译器自动完成
    System.out.println(weekday.toUpperCase());
}
```

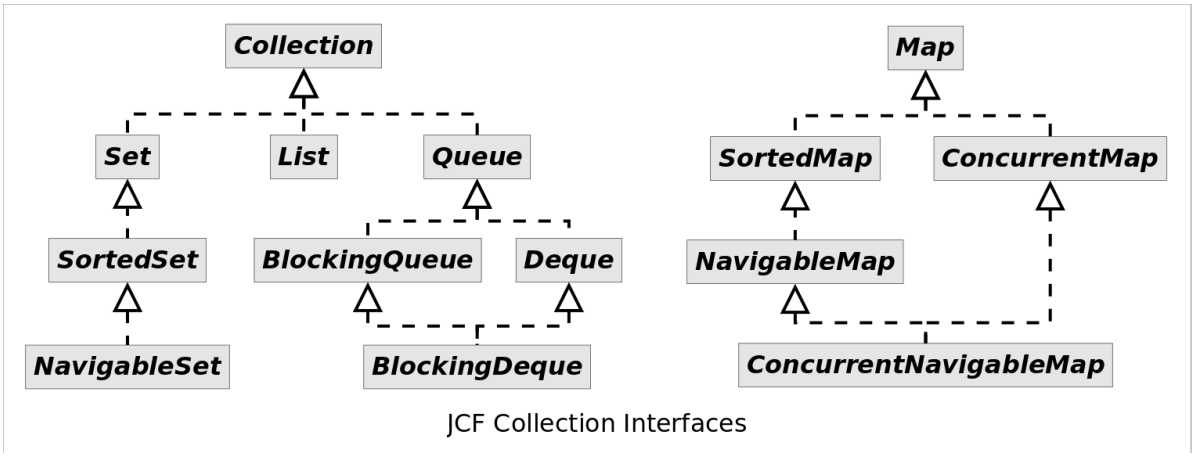
内存管理

跟C++复杂的内存管理机制不同，Java GC自动包揽了一切，Java程序并不需要处理令人头疼的内存问题，因此JCF并不像C++ STL那样需要专门的空间适配器（alloctor）。 另外，由于Java里对象都在堆上，且对象只能通过引用（reference，跟C++中的引用不是同一个概念，可以理解成经过包装后的指针）访问，容器里放的其实是对象的引用而不是对象本身，也就不存在C++容器的复制拷贝问题。

接口和实现（Interfaces and Implementations）

接口

为了规范容器的行为，统一设计，JCF定义了14种容器接口（collection interfaces），它们的关系如下图所示：



JCF Collection Interfaces

*Map*接口没有继承自*Collection*接口，因为*Map*表示的是关联式容器而不是集合。但Java为我们提供了从*Map*转换到*Collection*的方法，可以方便的将*Map*切换到集合视图。 上图中提供了*Queue*接口，却没有*Stack*，这是因为*Stack*的功能已被JDK 1.6引入的*Deque*取代。

实现

上述接口的通用实现见下表：

		Implementations				
		Hash Table	**Resizable Array**	**Balanced Tree**	**Linked List**	**Hash Table + Linked List**
Interfaces	**Set**	`HashSet`		`TreeSet`		`LinkedHashSet`
	List		`ArrayList`		`LinkedList`	
	Deque		`ArrayDeque`		`LinkedList`	
	Map	`HashMap`		`TreeMap`		`LinkedHashMap`

接下来的篇幅，会逐个介绍上表中容器的数据结构以及用到的算法。

迭代器（Iterator）

跟C++ STL一样，JCF的迭代器（Iterator）为我们提供了遍历容器中元素的方法。只有容器本身清楚容器里元素的组织方式，因此迭代器只能通过容器本身得到。每个容器都会通过内部类的形式实现自己的迭代器。相比STL的迭代器，JCF的迭代器更容易使用。

```
//visit a list with iterator
ArrayList<String> list = new ArrayList<String>();
list.add(new String("Monday"));
list.add(new String("Tuesday"));
list.add(new String("Wensday"));
Iterator<String> it = list.iterator();//得到迭代器
while(it.hasNext()){
    String weekday = it.next();//访问元素
    System.out.println(weekday.toUpperCase());
}
```

JDK 1.5 引入了增强的for循环，简化了迭代容器时的写法。

```
//使用增强for迭代
ArrayList<String> list = new ArrayList<String>();
list.add(new String("Monday"));
list.add(new String("Tuesday"));
list.add(new String("Wensday"));
for(String weekday : list){//enhanced for statement
    System.out.println(weekday.toUpperCase());
}
```

源代码

JDK安装目录下的src.zip包含了Java core API的源代码，本文采用的是JDK 1.7u79的源码，下载地址。这里复制了一份。

参考文献

- Collections Framework Overview
- The For-Each Loop