第七章 泛型在反射中的应用

反射是一个可以让编程者可以对程序定义进行修改的特性，反射在java中扮演的角色是class访问，对象监测，调试，编译，服务如JavaBean以及对象序列化，以及任何要创建，监测，随意的操控java对象的。

反射从java语言开始之初这一特性就被延用到现在。但是其中有两个大的对于反射的改变就是。泛型在反射中的引用，以及反射对于泛型。

泛型在反射中的应用，我是说的之前的一些类型，现在都用泛型代替了。特别是，之前的类Class现在变成Class<T> 了。这个看起来可能会觉得有点奇怪，但是一旦理解了，可以让我们使用反射更加的清晰。字面量Class和方法getClass使用了特别的技巧来返回更确切的类型。泛型在反射注解的效果特别的好。我们知道，类型参数T在Class<T>总是应该绑定具体的类型。我们提供了短小的库用于帮助我们使用一些未检查的类型转化。

反射对于泛型指的是，反射现在返回信息是关于泛型类型的。有一些新的接口用于代表泛型类型。包括类型变量，参数类型，通配符类型，有一些新的方法来获取泛型类型的属性，构造器以及方法。

7.1泛型在反射中的应用

java支持反射特性从1.0开始，类字面量支持从1.1开始。这些的核心就是Class，代表着运行时候一个对象的所有信息。你可以通过.class方法字面量的方式获取类类型，也可以通过getClass方法获取。

Class ki = Integer.class;

Number n = new Integer(12);

assert n.getClass() == ki

对于同一个类构造器来说，同一个类型总是有且仅有一个Class类型。为了强调这一点，我们通过 ＝＝ 比较符号进行比较。然而，在大多数情况下，使用equals方法更为妥当。

Java5之后的一大改变是Class现在可以放入一个类型变量的泛型，让Class代表着中类型的Class如Class<T> 标示T类型的class现在上面的代码可以写成下面的形式。

Class<Integer> ki = Integer.class;

Number n = new Integer(12);

Class<? extends Number> kn = n.getClass();

assert ki == kn

Class代表以及getClass方法被特殊对待被编译器。通常来说，如果T是一个类型没有类型参数，那么T.class的类型就是Class<T> 如果e是一个T表达式那么我们可以通过Class<? extends T> 来表示。通配符是需要的因为可能是子类型。

大多数使用反射的例子，你是不知道类的具体类型的否则你也不会用反射了。在这些粒子中你可以通过Class<?> 来写那个类型。使用未绑定的通配符。

7.2 反射类型是具体类型

反射能够让具体类型信息通过变成可获取。必要时候，因此，每个类都对一个具体类型做出反应。如果你想熬反射一个参数类型你就能获取具体信息通过相应的返回具体类型。

List<integer> in = new ArrayList<Integer>();

List<String> str = new ArrayList<String>();

assert in == str;

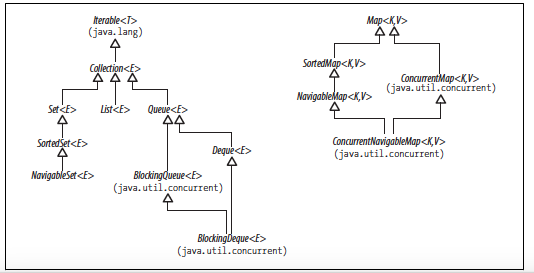
assert in == ArrayList.class;

10 java中主要的接口框架结构

这个章节主要介绍java的集合框架还有另外一个Iterable 这个在集合框架之外，但是在描述集合的时候也是不可或缺需要进行介绍的。他的作用如下：

* Iterable定义了一个类需要完成的合约。因为这个类的实例必然是可以被用于foreach语句的。

集合框架拥有下面的这些作用

* Collection定义了很多的功能，相比于map。他并没有直接的具体实现。但是具体的集合类都是Collection的子接口的实现。也是极好的。
* Set 是一个集合。元素不能够重复。并且是无序的。有序的set 可以自动排序并且返回他们按照放入的顺序。比如NavigableSet 就实现了这个，可以通过添加方法找到最接近匹配的目标元素。
* Queue 队列是一个集合设计用于接受元素从他的尾部用于处理。并且从头部出去，遵守先进先出原则。他的子接口Deque可以允许从头部和尾部添加或者删除数据。Queue和Deque有子接口，BlockingQueue以及BlockingDeque支持并发允许对线程阻塞。通过定义时间或者一致阻塞。知道要求的操作被完成。
* List是一个集合用于按照顺序处理元素并且允许重复元素。
* Map是一个集合使用键值对用于存储元素。他的继承ConcurrentMap支持并发处理。SortedMap支持通过key排序。保证返回他的值通过升序。通过NavigableMap 继承子SortedMap可以找到最符合要求的远足。ConcurrentNavigableMap用于继承并发Map以及NavigableMap。
* 

11 基础预习

这个章节，我们会花时间讨论一下基于这个框架下的一些底层的概念。

11.1 Iterable 和Iterators

迭代器指的是实现了Iterator接口的对象，迭代器的目的是提供一个统一的方式用于取出集合的元素。所以不论是那种集合你在使用，并且不管他们的实现是怎么样的。你都知道如何处理他们的元素按照顺序。这样通常需要写一些很冗余的代码带时通过心的for循环，则很容易处理了。

在java5的集合接口被制作成继承Iterable接口，因此任何的set，list，queue都能够被foreach形式的遍历。如果你写出你自己的Iterable的实现你也是可以食用foreach的。

Iterator的普遍用途在ArrayList和HashMap中使用的十分平凡。当我们在使用iterator变量集合的时候，任何对集合的改变都会爆出ConcurrentModifyException。

11.2实现

之前我们大只的看了下集合类的接口，这些接口定义了我们期待的集合的一些行为。但是正如我们介绍的一样，有很多种方式去实现者实现这些接口。为什么框架不就使用最好的一种实现方式呢。那当然会让生活变得简单。太简单实际上。根本不像生活。其中的一些实现可能对其他来说并不是什么好的操作。墨菲定律告诉我们这对其他人肯定是一个坏处。因为根本就没有最好的实现。所以我们必须要考虑减少一些，只是提供一些最为共用的程序。选择你的县实现去优化这些操作。

其中最常见的三种操作类型就是。通过位置插入和删除元素。同步内容取得元素。和遍历整个集合元素。下面有四种常见的数据类型用于实现这样的操作。

Arrays

其他所有的编程语言从Fortran开始就有了数组。因为数组是直接从硬件上实现的。他们有随机内存的属性。对于通过位置的元素获取。以及遍历元素十分的迅速但是插入一记移除元素在任意的位置很慢。通过数组实现的集合包括，ArrayList，CopyOnWriteArrayList,EnumSet,和EnumMap以及很多的Queue以及Deque的实现。他们也是实现hash table的很重要的一个实现部分。

Linked lists

正如名字暗示的那样，这些是一组嵌入的链式结构。每个cell包含一个指向下一个cell的索引。Linked lists 和数组的实现相当的不同。通常按照位置去取元素十分的缓慢。因为你不得不从开始的索引一个个的索引过去。但是插入荷删除操作挺快的。

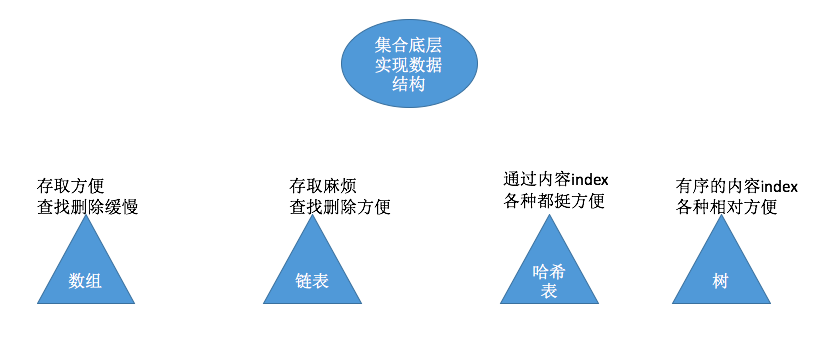
链表被用于例如ConcurrentLinkedQueue,LinkedBlockingQueue以及linkedList他们也被用于实现HashSet以及LinkedHashSet 等等。

Hash tables

他们提供一种存储元素的方式通过根据键值取建立一个索引而不是通过int类型的位置index。比如List。和数组和链表不同的是。哈希表不提供根据位置获取元素内容的方法。但是通过内容获取数据。这样的方式很快，不论是取元素还是插入还是删除元素。哈希表被使用的场景十分丰富。例如Set Map HashSet LinkedHashSet 以及 ＊hashMap

Trees

树形结构也是通过内容index 元素的。但是不同的是。他们能够存储以及获取通过一种有序的方式。他们相对来说还是挺快的对于插入以及删除操作。通过内容来获取他们以及遍历全部的内容。引用于属性结构的结构如TreeSet TreeMap PriorityHeap



11.5 集合以及线程安全

当一个程序在跑的时候，他可能与多个线程同时在执行这个java程序不同的线程可以同时获取同一块内存区域和系统资源。对于有多个cpu的实质性的并发能够通过将不同的线程分配给不同的cpu完成真正意义上的并发。当线程数量多于cpu数量的时候，我们通过时间切片来完成。使用并发编程的好处在于，在多核以及多cpu的机器上失分明显。第二个好处在于两个行为如果相互没有关系的话，你可能不想让其中一个行为需要等待另一个程序完成。这种需求特别在GUI中使用到。虽然并发对于效率的替身有实质性的好处，但是它是有代价的。不同的线程同时处理同一块内存区域可能导致意想不到的结果。除非你限制他们的访问顺序。