七、八、九

2019年1月15日　　9:37

- 异常分类：
  - Throwable：　异常对象都派生于Throwable类
    - Error：描述Java运行时系统的内部错误和资源耗尽错误
    - Exception：程序设计关注
      - RuntimeException:由程序错误导致的异常（如果出现RuntimeException异常，那么一定时你的问题）
        - 错误的类型转换
        - 数组访问越界
        - 访问null指针
      - 其他异常：程序本身没有问题，但由于向I/O错误这类问题导致的异常
        - 试图在文件尾部后面读取数据
        - 试图打开一个不存在的文件
        - 试图根据给定的字符串查找Class对象，而这个类并不存在
  - 此外，Java语言规范又把派生于Error类和RuntimeException类产生的异常成为**非受查**异常，所有其他的异常称为**受查**异常，编译器将会检是否为所有的受查异常提供了异常处理器
- 如果在子类覆盖了超类的一个方法，子类方法中声明的受查异常不能比超类方法声明中的异常更加通用
- 异常包装：（建议使用）捕获一个异常时，重新包装为自定义或其他异常，需要原始异常的错误信息，此时推荐使用异常的initCause(Throwable)方法，抛出的异常再使用getCause()即可获得原始异常。
- 带资源的try语句：
  - Java7以前代码如：
    ```
    open a resource
    try{
     work with the resource
    }finally{
     close the resource
    }
    ```
    这种情况，可能再finally里调用close方法时，又发生IOException，此时程序将抛出IOException，而覆盖了原始异常
  - Java7增加了一个AutoCloseable接口，实现这个接口的资源类可以使用带资源的try语句：
    ```
    try(Resource res = ...){
     work with res
    }
    ```
    此时，不论这个块正常退出或是存在一个异常，都会调用close()方法，并且抛出原始异常，而抑制close()方法产生的异常。可以通过抛出的异常e，调用 e.getSupppressed()方法获得抑制的异常。
- 异常机制技巧：早抛出，晚捕获
- 断言形式：
  - assert 条件；
  - assert 条件：表达式；
- 启用和禁用断言：断言默认禁用，可以在运行时用-enableassertions或-ea选项启用，用-disableassertions或-da禁用。禁用和启用断言不会重新编译，这是类加载器的功能。
- 断言失败是致命的、不可恢复的错误；断言检查只用于开放和测试阶段，不应该使用断言完成生产版本的逻辑操作。
- 默认情况，日志 管理器配置文件存在于jre/lib/logging.properties
- **泛型类**：将类型变量用尖括号<>括起来，并放在类名后面。一般用变量E表示集合元素类型，K和V表示表的关键字和值类型，T,U,S表示任意类型。使用时使用具体类型替换类型变量即可
- **泛型方法**：可以定义在普通类和泛型类中，将类型变量放在修饰符（public static)的后面，返回类型的前面。使用时在方法名前的尖括号中放入具体的类型即可，也可省略，大多时候都省略。
- **类型变量的限定**：将类型变量绑定到指定类型的子类型，如：<T extends Comparable &

Serizlizable>,这个限定类型列表中最多只能有一个类，且必须在限定列表中的第一个

- **类型擦除：**
  - 在虚拟机中没有泛型类型对象，所有对象都属于普通类。不论何时定义泛型类，都会自动提供一个**原始类型**，原始类型就是删去类型参数后的泛型类型名，擦除类型变量，并替换为限定类型列表的第一个类型变量（若果没有则使用Object）比如：
    Class Interval <T extends Comparable&Serializable>{T v1;T v2;}=>class Interval {Comparable v1,Comparable v2;}
  - 泛型方法也会按同样的方式进行类型擦除。
  - 在类型擦除之后，方法调用将在必要的时候使用强制类型转换。
- **桥方法：**在泛型子类覆盖父类方法，导致类型擦除与多态发生冲突时，编译器会自动生成一个桥方法。比如：

```
classPair<TextendsComparable>{
T first;
T second;

Public T getFirst(){
Return first;
}

Public void setSecond(Tsecond){
this.second=second;
}
}
Class DateInterval extends Pair<LocalDate>{
@Override
Public void setSecond(LocalDatesecond){
if(second.compareTo(getFirst())>=0){
super.setSecond(second);
}
}
}
```

此时，在DateInterval的原始类型中，将会有一个setSecond(LocalDate)方法，一个setSeconde(Comparable)方法，setSecond(Comparable)方法体内再调用setSecond(LocalDate)方法来实现多态调用，这个自动生成的setSecond(Comparable)方法叫做**桥方法。**（之前讲的具有协变的返回类型本质也是合成了一个桥方法）

- 泛型的约束与局限性：
  - 不能用基本类型实例化类型参数，这是因为类型擦除之后Object不能代表基本类型
  - 运行时类型查询只适用于原始类型，即a instanceof Pair<String> 将报错，getClass总是返回原始类型。
  - 不能创建参数化类型的数组，因为在类型擦除后，可以往数组中插入不同类型变量的泛型类，导致ClassCastException.即：

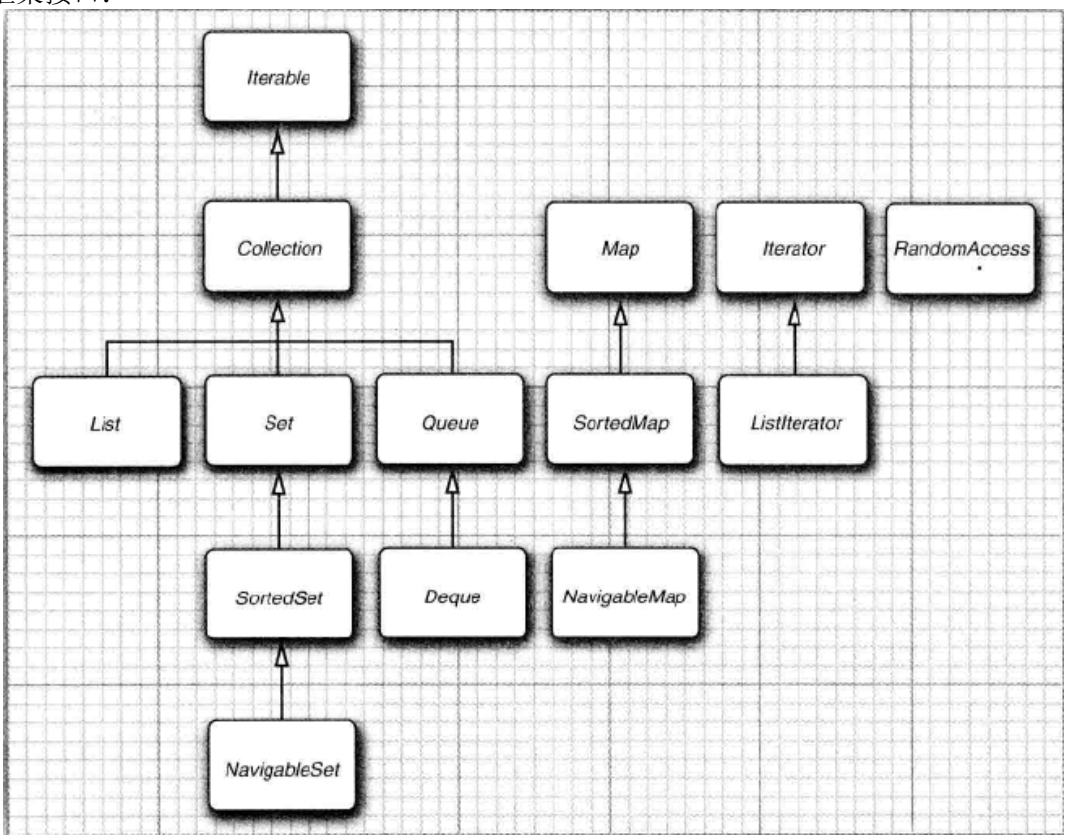
```
Pair<String>[] table =new Pair<String>[10] ;//所以不允许，会编译错误
Table[0]=new Pair<Employee>() ;
```

    可以使用ArrayList替代数组（不可创建，但可以声明Pair<String>[] table;
  - Varargs警告：参数个数可变时本质上会生成一个数组，当可变参数类型是泛型类型时，就会产生泛型数组，java虚拟机不会报错，但是会产生一个警告，可以使用@SafeVarargs抑制这个警告。对于秩序读取参数数组元素的方法，都可以使用这个注解抑制警告，但还是隐藏着危险，一旦使用数组元素时，很可能产生错误。
  - 不能实例化类型变量：不能new T(),T.Class等
  - 不能再静态域或方法中引用类型变量：
    static T var1//error
    static T getVar(){}//error 但可以这样：static <T> T getVar(){}
  - 不能抛出或捕获泛型类的实例
  - 可以消除对受查异常的检查：关键代码：

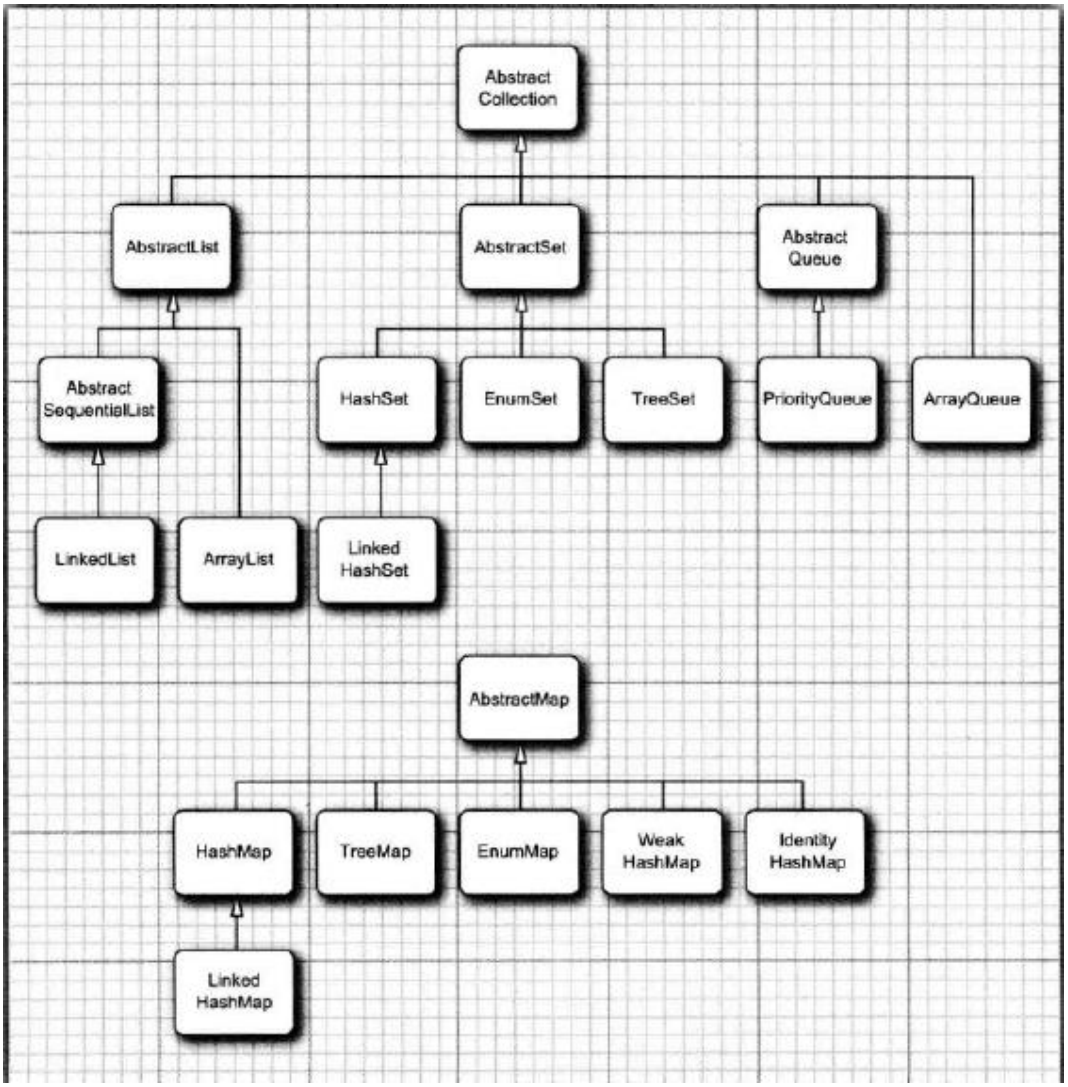
```
@SuppressWarning("unchecked")
public static <T extends Throwable> void throwAs(Throwable e) throws T{
    throw (T)e;
```

```
        }
```

- Pair<Manager>和Pair<Employee>没有任何关系，不能转换
- **通配符(?)限定(? extends Employee)**与**类型变量限定(T extends Employee)**类似，有一个附加能力，可以指定一个**超类型限定(? super Manager)**。其中：G代码某种泛型类
  - 子类型限定G(? extends Empoyee)，Employee类型或Employee的子类型。受此限定的变量，调用set方法时，只能接受null，因为无法确认接受的具体时Employee的哪种子类型。但可以调用get方法返回值,即安全地访问数据，返回类型为Employee。
  - 超类型限定G(? super Manager),Manager或Manager的超类型，直至Object类。受此限定的变量，调用set方法时，可以接受null和Manager即其子类。调用get方法只能赋给Object。
  - 无限定G(?)：受此限定的变量，调用set方法时，只能传递null，作为方法返回只能赋值给Object。
  - 参考https://blog.csdn.net/Baple/article/details/25056169
- 集合框架接口：



- 集合框架中的类：



集合类型	描　述
ArrayList	一种可以动态增长和缩减的索引序列
LinkedList	一种可以在任何位置进行高效地插入和删除操作的有序序列
ArrayDeque	一种用循环数组实现的双端队列
HashSet	一种没有重复元素的无序集合
TreeSet	一种有序集
EnumSet	一种包含枚举类型值的集
LinkedHashSet	一种可以记住元素插入次序的集
PriorityQueue	一种允许高效删除最小元素的集合
HashMap	一种存储键/值关联的数据结构
TreeMap	一种键值有序排列的映射表
EnumMap	一种键值属于枚举类型的映射表
LinkedHashMap	一种可以记住键/值项添加次序的映射表
WeakHashMap	一种其值无用武之地后可以被垃圾回收器回收的映射表
IdentityHashMap	一种用 == 而不是用 equals 比较键值的映射表

- ListIterator接口（LinkedList实现了该接口）
  - 增加了一个add()方法，表示再迭代器前添加一个元素。
  - 增加了一个previous()，同next()相反
  - 增加了一个hasPrevious(),同hasNext()相反
  - 可以调用LinkedList的listIterator()方法获得ListIterator实例
- 同一集合的多个迭代器，若其中一个对集合结构进行更改，即添加和删除元素，将导致其他迭代器抛出ConcurrentModeifcationException异常，但set方法不会。
- 推荐使用链表LinkedList，在表元素少或不需要对表元素进行随机访问的时候。需要经常使用get，set访问表元素时，推荐使用数组列表；但在需要并发同步的时候使用Vector。
- 更新映射项：
  - map.put(key,map.getOrDefault(key,0)+1)
  - map.putIfAbsent(key,0);map.put(key,map.get(key)+1)
  - Map.put(key,1,Integer::sum);//如果键原先不存在，先将key与1映射，否则使用Integer::sum函数组合原值和1

- LinkeHashMap默认使用插入顺序，初始化accessOrder为true，可以到达使用访问顺序
- IdentityHasMap键的散列值不是hashCode函数计算的，而是使用System.identityHashCode方法计算，这是使用对象内存地址计算，所以此时比较需要使用==
- 视图：例如映射类的keySet方法返回一个实现Set接口的类对象，这个类的方法堆原映射进行操作，这种方式产生的集合称为视图。如：
  a. 轻量级集合包装器：Arrays.asList(),Collections.nCopies()等（注意Arrays.asList返回的ArrayList并不是java.util.ArrayList,而是java.util.Arrays$ArrayList,一个视图类。它继承AbstractList,但其中修改数组大小的方法（比如与迭代器相关的add和remove方法）并未实现（其实是覆盖，因为AbstractList里面的默认实现是抛出UnsupportOperationException。））
  b. 子范围：subList(begin,end),有序集和映射除了按照索引位置以外，还可以按照排列顺序取子集如SortedSet的subSet(from,to)方法。
  c. 不可修改视图，Collections.unmodifiableCollection,Collections.unmodifiableList等，调用修改器方法将会抛出UnspportedOperationException
  d. 同步视图：多线程时需要同步map，可以使用Collections.synchronizedMap(map)，Collections.synchronizedList等方法，将map、list等转换称具有同步访问方法的map、list
  e. 受查视图，泛型集合很有可能发生错误类型元素混入，但只有在get元素，并进行类型cast的时候才能发现，而Collections.checkList(list,String.class)等方法可以产生受查对象，在插入时检查插入的是否时String类（但不能嵌套检查，比如ArrayList<Pair<String>>,无法检查插入Pair<Date>
  f. 排序与混排，Collections.sort(list)(元素实现Comparable接口），也可以用list.sort(comparator)（传入一个comparator对象），Collections.reverseOrder()(逆排序，先按元素类型的compareTo方法排序，然后使用比较器b.compareTo(a)排序。或list.sort(comparator.reversed())（先根据传入的comparator对象排序，然后再逆排）。混排Collections.shuffle(list)（随机混排序）
- 二分查找：Collections.binarySearch(list,element),如果element没有实现Comparable接口，则Collection.binarySearch(list,element,comparator)(list必须升序排列。如果返回值是正，则表示匹配对象的索引，如果是负值，则表示没有匹配对象，但可以将element插入到-i-1位置（此时返回的-i值表示第一个比element大的对象索引），保持有序）（如果list不是randomAccess的且list长，则会自动改为线性查找）
- 批操作:
  - collection1.removeAll(collection2);
  - Collection1.retainAll(collection2)；删除collection2中没出现过的元素，（可以用来做交集）
- 集合转数组，collection.toArray()方法返回的是object[]数组，且不可以进行类型转换，可以使用collection.toArray(new String[size])方法，将会返回String[]类型（若size不够大则将会新建一个数组）
- 位集：BitSet用于存放一个位序列（可以看作一个01串），bitSet.set(i)将第i位设置为开状态，bitSet.clear(i)清除，设置为关状态