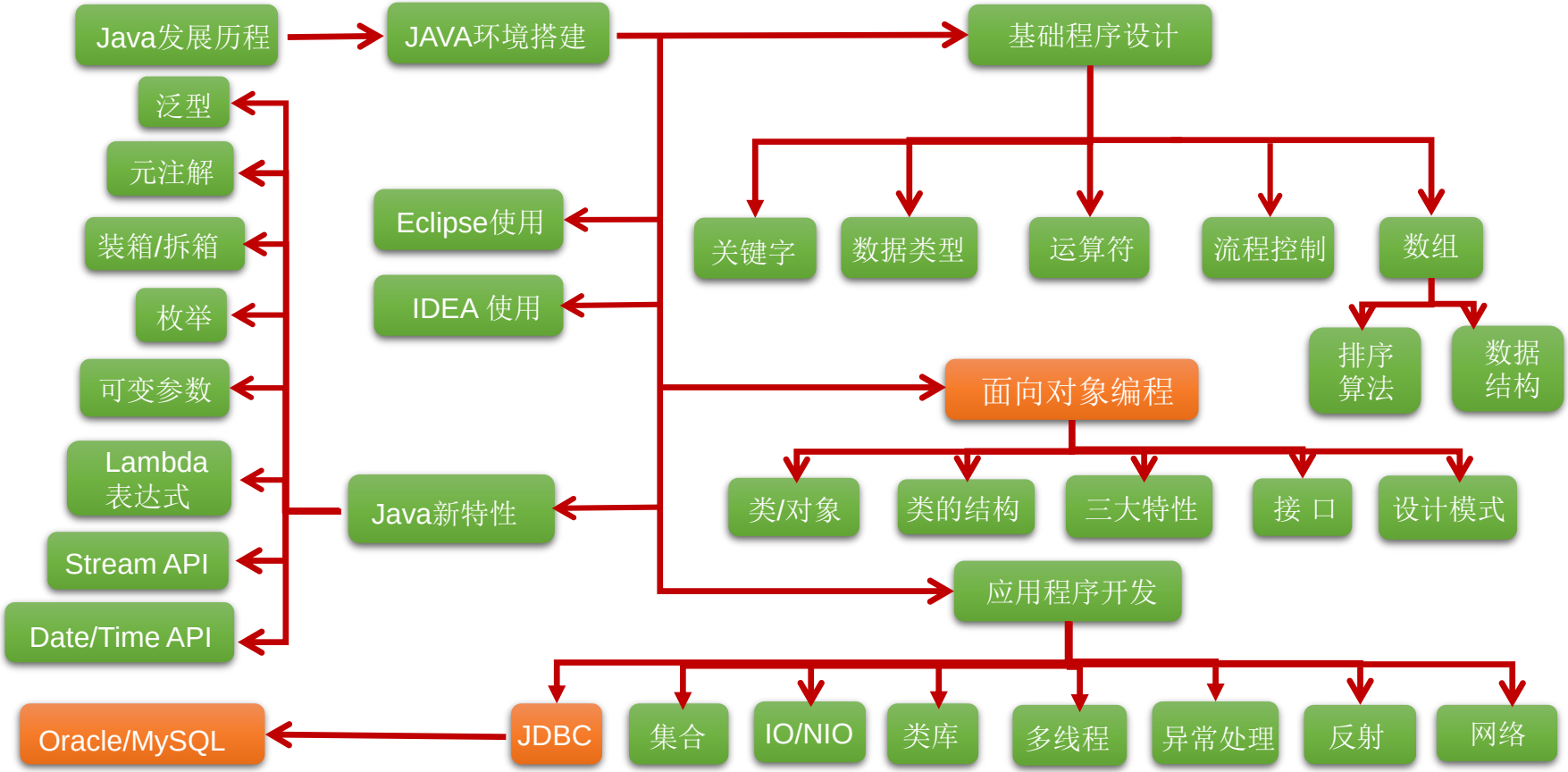
第12章 泛型

讲师：宋红康
新浪微博：尚硅谷-宋红康

Java基础知识图解



目录

尚硅谷

12-1 为什么要有泛型

12.1 为什么要有泛型(Generic)

- 泛型：标签
- 举例：
  - 中药店，每个抽屉外面贴着标签
  - 超市购物架上很多瓶子，每个瓶子装的是什么，有标签
- 泛型的设计背景

集合容器类在设计阶段/声明阶段不能确定这个容器到底实际存的是什么类型的对象，所以在JDK1.5之前只能把元素类型设计为Object，JDK1.5之后使用泛型来解决。因为这个时候除了元素的类型不确定，其他的部分是确定的，例如关于这个元素如何保存，如何管理等是确定的，因此此时把元素的类型设计成一个参数，这个类型参数叫做泛型。Collection<E>，List<E>，ArrayList<E> 这个<E>就是类型参数，即泛型。

泛型的概念

- 所谓泛型，就是允许在定义类、接口时通过一个标识表示类中某个属性的类型或者是某个方法的返回值及参数类型。这个类型参数将在使用时（例如，继承或实现这个接口，用这个类型声明变量、创建对象时）确定（即传入实际的类型参数，也称为类型实参）。
- 从JDK1.5以后，Java引入了“参数化类型（Parameterized type）”的概念，允许我们在创建集合时再指定集合元素的类型，正如：List<String>，这表明该List只能保存字符串类型的对象。
- JDK1.5改写了集合框架中的全部接口和类，为这些接口、类增加了泛型支持，从而可以在声明集合变量、创建集合对象时传入类型实参。

泛型的类型必须是类，不能是基础数据类型，可以使用包装类

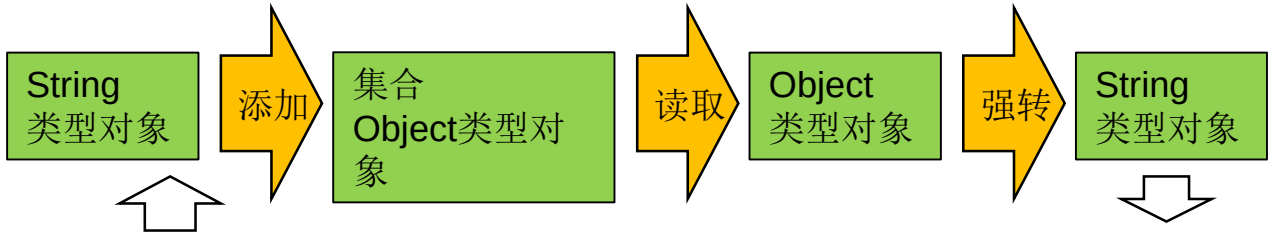
12.1 为什么要有泛型(Generic)

那么为什么要有泛型呢，直接Object不是也可以存储数据吗？

1. 解决元素存储的安全性问题，好比商品、药品标签，不会弄错。
2. 解决获取数据元素时，需要类型强制转换的问题，好比不用每回拿商品、药品都要辨别。

使用泛型后，可以用在compare()，compareTo()，equals()等需要强制转换类型的方法中。

在集合中没有泛型时



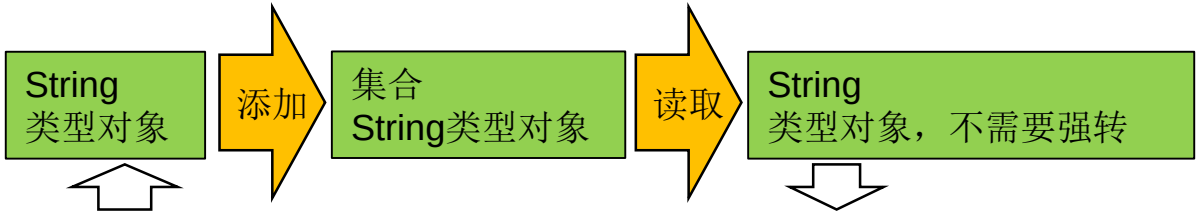
任何类型都可以添加到集合中：类型不安全

读取出来的对象需要强转：繁琐
可能有ClassCastException

若调用时没有指定泛型类型，则默认为Object

12.1 为什么要有泛型(Generic)

在集合中有泛型时



只有指定类型才可以添加到集合中：类型安全　　读取出来的对象不需要强转：便捷

Java泛型可以保证如果程序在编译时没有发出警告，运行时就不会产生ClassCastException异常。同时，代码更加简洁、健壮。

12-2 在集合中使用泛型

12.2 在集合中使用泛型


```
ArrayList<Integer> list = new ArrayList<>();//类型推断

list.add(78);
list.add(88);
list.add(77);
list.add(66);

//遍历方式一：
//for(Integer i : list){
//不需要强转
    //System.out.println(i);
//}

//遍历方式二：
Iterator<Integer> iterator = list.iterator();
while(iterator.hasNext()){
    System.out.println(iterator.next());
}
```

```
Map<String,Integer> map = new HashMap<String,Integer>();

map.put("Tom1",34);
map.put("Tom2",44);
map.put("Tom3",33);
map.put("Tom4",32);
//添加失败
//map.put(33, "Tom");

Set<Entry<String,Integer>> entrySet = map.entrySet();

Iterator<Entry<String,Integer>> iterator =  entrySet.iterator();

while(iterator.hasNext()){
     Entry<String,Integer> entry = iterator.next();
     System.out.println(entry.getKey() + "--->" + entry.getValue());
}
```

12-3 自定义泛型结构

- 自定义泛型类
- 自定义泛型接口
- 自定义泛型方法

12.3 自定义泛型结构

1.泛型的声明

interface List<T> 和 class GenTest<K,V>

其中，T,K,V不代表值，而是表示类型。这里使用任意字母都可以。

常用T表示，是Type的缩写。

泛型参数用于设定一些不确定的参数，如果我们自定义一个类，可以传入多种参数，那么就使用泛型来作为不确定的参数

2.泛型的实例化：

一定要在类名后面指定类型参数的值（类型）。如：

```
List<String> strList = new ArrayList<String>();
Iterator<Customer> iterator = customers.iterator();
```

- T只能是类，不能用基本数据类型填充。但可以使用包装类填充
- 把一个集合中的内容限制为一个特定的数据类型，这就是generics背后的核心思想

```
Comparable c = new Date();
System.out.println(c.compareTo("red"));
```

JDK 1.5 之前

```
Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));
```

JDK 1.5

体会：使用泛型的主要优点是能够在编译时而不是在运行时检测错误。

12.3 自定义泛型结构：泛型类、泛型接口

1. 泛型类可能有多个参数，此时应将多个参数一起放在尖括号内。比如：<E1,E2,E3>
2. 泛型类的构造器如下：public GenericClass(){}。而下面是错误的：public GenericClass<E>(){}

构造器中不需要加泛型，但是在实例化对象时需要添加泛型

3. 实例化后，操作原来泛型位置的结构必须与指定的泛型类型一致。
4. 泛型不同的引用不能相互赋值。

>尽管在编译时ArrayList<String>和ArrayList<Integer>是两种类型，但是，在运行时只有一个ArrayList被加载到JVM中。

5. 泛型如果不指定，将被擦除，泛型对应的类型均按照Object处理，但不等价于Object。**经验：**泛型要使用一路都用。要不用，一路都不要用。
6. 如果泛型结构是一个接口或抽象类，则不可创建泛型类的对象。
7. jdk1.7，泛型的简化操作：ArrayList<Fruit> flist = new ArrayList<>();
8. 泛型的指定中不能使用基本数据类型，可以使用包装类替换。

12.3 自定义泛型结构：泛型类、泛型接口


```
class GenericTest {
    public static void main(String[] args) {
        // 1、使用时：类似于Object，不等同于Object
        ArrayList list = new ArrayList();
        // list.add(new Date());//有风险
        list.add("hello");

        test(list);// 泛型擦除，编译不会类型检查

        // ArrayList<Object> list2 = new ArrayList<Object>();
        // test(list2);//一旦指定Object，编译会类型检查，必须按照Object处理
    }

    public static void test(ArrayList<String> list) {
        String str = "";
        for (String s : list) {
            str += s + ",";
        }
        System.out.println("元素:" + str);
    }
}
```

9. 在类/接口上声明的泛型，在本类或本接口中即代表某种类型，可以作为非静态属性的类型、非静态方法的参数类型、非静态方法的返回值类型。但在静态方法中不能使用类的泛型。

因为类的泛型是在对象实例化后指定的，而静态方法是在对象实例化之前就初始化的，因此它在类的泛型指定之前，无法使用泛型。

10. 异常类不能是泛型的

11. 不能使用new E[]。但是可以：E[] elements = (E[])new Object[capacity];

参考：ArrayList源码中声明：Object[] elementData，而非泛型参数类型数组。

12. 父类有泛型，子类可以选择保留泛型也可以选择指定泛型类型：

- 子类不保留父类的泛型：按需实现
  - 没有类型 擦除
  - 具体类型
- 子类保留父类的泛型：泛型子类

子类依然是泛型类

  - 全部保留
  - 部分保留

结论：子类必须是“富二代”，子类除了指定或保留父类的泛型，还可以增加自己的泛型

```
class Father<T1, T2> {
}
// 子类不保留父类的泛型
// 1)没有类型 擦除
class Son1 extends Father {// 等价于class Son extends Father<Object,Object>{
}
// 2)具体类型
class Son2 extends Father<Integer, String> {
}
// 子类保留父类的泛型
// 1)全部保留
class Son3<T1, T2> extends Father<T1, T2> {
}
// 2)部分保留
class Son4<T2> extends Father<Integer, T2> {
}
```

```
class Father<T1, T2> {
}
// 子类不保留父类的泛型
// 1)没有类型 擦除
class Son<A, B> extends Father{//等价于class Son extends Father<Object,Object>{
}

// 2)具体类型
class Son2<A, B> extends Father<Integer, String> {
}
// 子类保留父类的泛型
// 1)全部保留
class Son3<T1, T2, A, B> extends Father<T1, T2> {
}
// 2)部分保留
class Son4<T2, A, B> extends Father<Integer, T2> {
}
```

```java
class Person<T> {
    // 使用T类型定义变量
    private T info;
    // 使用T类型定义一般方法
    public T getInfo() {
        return info;
    }
    public void setInfo(T info) {
        this.info = info;
    }
    // 使用T类型定义构造器
    public Person() {
    }
    public Person(T info) {
        this.info = info;
    }
```

```java
// static的方法中不能声明泛型
//public static void show(T t) {
//
//}
// 不能在try-catch中使用泛型定义
//public void test() {
//try {
//
//} catch (MyException<T> ex) {
//
//}
//}
}
```

12.3 自定义泛型结构：泛型方法

- 方法，也可以被泛型化，不管此时定义在其中的类是不是泛型类。在泛型方法中可以定义泛型参数，此时，参数的类型就是传入数据的类型。
- 泛型方法的格式：

[访问权限] <泛型> 返回类型 方法名([泛型标识 参数名称]) 抛出的异常

- 泛型方法声明泛型时也可以指定上限(在12.5中讲)

```
public class DAO {

    public <E> E get(int id, E e) {

        E result = null;

        return result;
    }
}
```

泛型方法可以声明为static，因为它是在调用时才会传入参数，而不是在对象实例化时

泛型方法即可存在于普通方法中，也可存在于泛型方法中。

12.3 自定义泛型结构：泛型方法


```
public static <T> void fromArrayToCollection(T[] a, Collection<T> c) {
    for (T o : a) {
        c.add(o);
    }
}

public static void main(String[] args) {
    Object[] ao = new Object[100];
    Collection<Object> co = new ArrayList<Object>();
    fromArrayToCollection(ao, co);

    String[] sa = new String[20];
    Collection<String> cs = new ArrayList<>();
    fromArrayToCollection(sa, cs);

    Collection<Double> cd = new ArrayList<>();
    // 下面代码中T是Double类，但sa是String类型，编译错误。
    // fromArrayToCollection(sa, cd);
    // 下面代码中T是Object类型，sa是String类型，可以赋值成功。
    fromArrayToCollection(sa, co);
}
```

自定义泛型类举例：

```
class Creature{}
class Person extends Creature{}
class Man extends Person{}
class PersonTest {
    public static <T extends Person> void test(T t){
        System.out.println(t);
    }

    public static void main(String[] args) {
        test(new Person());
        test(new Man());
        //The method test(T) in the type PersonTest is not
        //applicable for the arguments (Creature)
        test(new Creature());
    }
}
```

12-4 泛型在继承上的体现

12.4 泛型在继承上的体现

请输出如下来两段代码有何不同

```
public void printCollection(Collection c) {
    Iterator i = c.iterator();
    for (int k = 0; k < c.size(); k++) {
        System.out.println(i.next());
    }
}
```

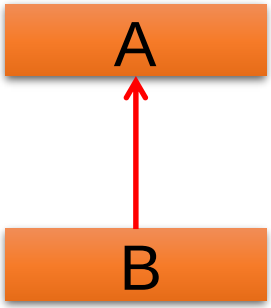
```
public void printCollection(Collection<Object> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

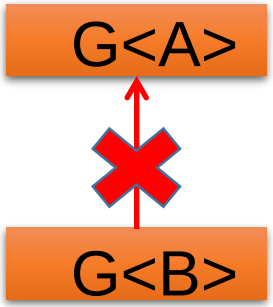
12.4 泛型在继承上的体现

但是，对于B<String>和A<String>这两个之间是继承关系，如List<String> 和 ArrayList<String>有继承关系

如果B是A的一个子类型（子类或者子接口），而G是具有泛型声明的类或接口，G<B>并不是G<A>的子类型！

比如：String是Object的子类，但是List<String >并不是List<Object>的子类。





```java
public void testGenericAndSubClass() {
    Person[] persons = null;
    Man[] mans = null;
    // 而 Person[] 是 Man[] 的父类.
    persons = mans;

    Person p = mans[0];

    // 在泛型的集合上
    List<Person> personList = null;
    List<Man> manList = null;
    // personList = manList;(报错)
}
```

12-5 通配符的使用

12.5 通配符的使用

1.使用类型通配符：？

比如：List<?> ，Map<?,?>

？用来表示任意一个类型，通常用来表示父类

List<?>是List<String>、List<Object>等各种泛型List的父类。

2.读取List<?>的对象list中的元素时，永远是安全的，因为不管list的真实类型是什么，它包含的都是Object。

3.写入list中的元素时，不行。因为我们不知道c的元素类型，我们不能向其中添加对象。

- 唯一的例外是null，它是所有类型的成员。

12.5 通配符的使用

- 将任意元素加入到其中不是类型安全的：

```
Collection<?> c = new ArrayList<String>();
c.add(new Object()); // 编译时错误
```

不能向有通配符的类中添加元素

因为我们不知道c的元素类型，我们不能向其中添加对象。add方法有类型参数E作为集合的元素类型。我们传给add的任何参数都必须是一个未知类型的子类。因为我们不知道那是什么类型，所以我们无法传任何东西进去。

- 唯一的例外的是null，它是所有类型的成员。

- 另一方面，我们可以调用get()方法并使用其返回值。返回值是一个未知的类型，但是我们知道，它总是一个Object。

12.5 通配符的使用

```
public static void main(String[] args) {
    List<?> list = null;
    list = new ArrayList<String>();
    list = new ArrayList<Double>();
    // list.add(3);//编译不通过
    list.add(null);

    List<String> l1 = new ArrayList<String>();
    List<Integer> l2 = new ArrayList<Integer>();
    l1.add("尚硅谷");
    l2.add(15);
    read(l1);
    read(l2);
}

public static void read(List<?> list) {
    for (Object o : list) {
        System.out.println(o);
    }
}
```

12.5 通配符的使用：注意点

```
//注意点1：编译错误：不能用在泛型方法声明上，返回值类型前面<>不能使用?
public static <?> void test(ArrayList<?> list){

}
```

```
//注意点2：编译错误：不能用在泛型类的声明上
class GenericTypeClass<?>{

}
```

```
//注意点3：编译错误：不能用在创建对象上，右边属于创建集合对象
ArrayList<?> list2 = new ArrayList<?>();
```

12.5 通配符的使用：有限制的通配符

- <?>
  允许所有泛型的引用调用
- 通配符指定上限
  上限extends：使用时指定的类型必须是继承某个类，或者实现某个接口，即<=
- 通配符指定下限
  下限super：使用时指定的类型不能小于操作的类，即>=
- 举例：
  - **<? extends Number>　(无穷小 , Number]**
    只允许泛型为Number及Number子类的引用调用
  - **<? super Number>　[Number , 无穷大)**
    只允许泛型为Number及Number父类的引用调用
  - **<? extends Comparable>**
    只允许泛型为实现Comparable接口的实现类的引用调用

记住，获取元素时返回的是最大的类型，能够囊括所有类型；添加元素时，添加的是最小的类型，能够转换为所有类型。

在获取元素时，对于extends，返回的元素类型是最大的上限类型，如左边的Number；对于super，返回的类型是最大的Object类型

在向其中添加元素时，对于extends，由于添加的元素要能够转换为其中所有的类型，因此添加的元素一定是最小的，但是extends中无法确定最小的，因此不能添加；对于super来说，最小的就是Number以及它的子类，因此可以添加Number和它的子类

```java
public static void printCollection3(Collection<? extends Person> coll) {
    //Iterator只能用Iterator<?>或Iterator<? extends Person>.why?
    Iterator<?> iterator = coll.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next());
    }
}

public static void printCollection4(Collection<? super Person> coll) {
    //Iterator只能用Iterator<?>或Iterator<? super Person>.why?
    Iterator<?> iterator = coll.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next());
    }
}
```

12.5 通配符的使用：有限制的通配符

练习题：为什么编译如下的操作会报错？

```
59
60      public static void addStrings(List<? extends Object> list) {
61
62  //          list.add("aaa");
63  //          list.add("aaa");
64
65          list.add(new A());
66          list.add(new Circle());
67          list.add(new GeometricObject());
68          list.add(new Object());
69      }
70
71      public void addString(List<? super A> list) {
72
73          list.add(new A());
74          list.add(new Circle());
75          list.add(new GeometricObject());
76          list.add(new Object());
77      }
```

12-6 泛型应用举例

如操作数据库的DAO，一般都是泛型，如通用Mapper，我们要使用时直接实现接口Mapper<T t>，就可以调用其中的一些方法了。

12.6 泛型应用举例：泛型嵌套


```
public static void main(String[] args) {
    HashMap<String, ArrayList<Citizen>> map = new HashMap<String, ArrayList<Citizen>>();
    ArrayList<Citizen> list = new ArrayList<Citizen>();
    list.add(new Citizen("刘恺威"));
    list.add(new Citizen("杨幂"));
    list.add(new Citizen("小糯米"));
    map.put("刘恺威", list);

    Set<Entry<String, ArrayList<Citizen>>> entrySet = map.entrySet();
    Iterator<Entry<String, ArrayList<Citizen>>> iterator = entrySet.iterator();
    while (iterator.hasNext()) {
        Entry<String, ArrayList<Citizen>> entry = iterator.next();
        String key = entry.getKey();
        ArrayList<Citizen> value = entry.getValue();
        System.out.println("户主：" + key);
        System.out.println("家庭成员：" + value);
    }
}
```

12.6 泛型应用举例：实际案例

用户在设计类的时候往往会使用类的关联关系，例如，一个人中可以定义一个信息的属性，但是一个人可能有各种各样的信息（如联系方式、基本信息等），所以此信息属性的类型就可以通过泛型进行声明，然后只要设计相应的信息类即可。

GenericPerson.java


尚硅谷
让天下没有难学的的技术