Java泛型详解

向日花开 关注
2016.12.22 11:30:35 字数 1,782 阅读 1,229

泛型

泛型由来

泛型字面意思不知道是什么类型，但又好像什么类型都是。看前面用到的集合都有泛型的影子。

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, (
...
}
```

以ArrayList为例,它为什么要写成ArrayList<E>这样.我也不知道他为什么要写成这样，但是我知道如果它不用泛型，那代码就乱了，那也别写代码了。

- ArrayList运用泛型可以这么写

```
ArrayList<String> strings = new ArrayList<>();//可以存String
ArrayList<Integer> integers = new ArrayList<>();//可以存Integer类型
ArrayList<Object> objects = new ArrayList<>();//可以存对象
```

- ArrayList没用泛型之后:

  如果要存放各种各样的样类型，是不是意味着写各种各样对象的链表，那开发者可以不用活了...咦，或者你可以可不用死了，你发现所有类都继承自Object类，那只要这么写一个

  ，

  在取出元素的时候强转成对应的类型就可以了，是的，这样就可以不会被写代码累死了。但为什么源码没有这么写，因为它没泛型强大！让我们看下面代码了解泛型的由来。

假如我要写一个类存放一个int类型的模型，那简单

```
public class IntegerFun {
  private int data;

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }
}
```


向日花开 关注
总资产107 (约9.96元)

Flutter Hero动画案例
阅读 64

CustomPainter——微信拍视频按钮效果实现
阅读 251

Flutter 小说爬虫示例
阅读 318

推荐阅读

三面字节跳动被虐得“体无完肤”，15天读完这份pdf，终拿下美团研发
阅读 56,259

Android刘海屏、水滴屏全面屏适配方案
阅读 4,361

Android 组件化开源app -开眼短视频(OpenEyes)
阅读 7,995

这是一份面向Android开发者的复习指南
阅读 10,701

做了5年Android，靠着这份面试题跟答案，我从12K变成了30K
阅读 33,474

满足你的需求，但需求变了，我还要一个存放String类型的，那你也忍了，再来一个

```
    public class StringFun {

        private String data;

        public String getData() {
            return data;
        }

        public void setData(String data) {
            this.data = data;
        }
    }
```

需求又添加了一个，存放Long、Student、Math.....于是撕逼开始...结束之后，这次你聪明了，写了一个万能的，管它存放什么都行的类：

```
public class ObjectFun {
    private Object data;

    public Object getData() {
        return data;
    }

    public void setData(Object data) {
        this.data = data;
    }
}
```

这样总算解决了问题，看用法：

你总觉得你写的前无故人，后无来者了，可是经理还是过来找你了，因为你的程序跑不起来了，你认真的看了一下，发现代码第十五行，存放的是Integer 结果你转成了Float出错了，那你可能会抱怨编译器
没有立即告诉你这里存在问题，接下来我们来看看运用泛型会怎么样。

```
public class Fun<T> {
    private T data;

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }
}
```

用法:

这就是使用泛型的原因.

多泛型

上面写的还不够全，因为Fun<T>只能存放一种类型的元素，假如我要存放多种呢，我希望你已经会了，再来一个泛型。

```
/**
 * 泛型类
 *
 * @param <T>泛型T
 * @param <V>泛型V
 */
public class Fun<T, V> {
    private T data;
    private V data2;

    //泛型方法
    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }

    public V getData2() {
        return data2;
    }

    public void setData2(V data2) {
        this.data2 = data2;
    }
}
```

要存放无数个呢.....

```
Fun<T,T1,T2,T3,.,.>{
}
```

泛型规范

T1,T2,T3,.....泛型可以随便写吗，可以随便写，但我们追求规范。

- E — Element，常用在java Collection里，如：List<E>,Iterator<E>,Set<E>
- K,V — Key，Value，代表Map的键值对
- N — Number，数字
- T — Type，类型，如String，Integer等等

泛型接口，泛型类，泛型方法

- 泛型接口

```
/**
 * 格式:接口名后面跟 <T>
 *
 * @param <T>
 */
public interface IManager<T> {
    void add(T data);

    T remove(int index);

```

```
    void sop();
}
```

- 泛型类（之前的都是）
- 泛型类实现泛型接口(关于怎么更好的构建泛型类，就靠诸君在日后的生涯中寻找答案了)

```
/**
 * @param <T>
 */
public class Manager<T> implements IManager<T> {
    private List<T> datas;

    public Manager() {
        datas = new ArrayList<>();
    }

    @Override
    public void add(T data) {
        datas.add(data);
    }

    @Override
    public T get(int index) {
        return datas.get(index);
    }

    @Override
    public void sop() {
        for (T t : datas) {
            System.out.println(t);
        }
    }
}
```

- 泛型方法(前面的好多)

```
   @Override
    public T get(int index) {
         return datas.get(index);
    }

    //泛型方法
     public T getData() {
         return data;
     }
```

案例运行

```
public class Demo {

    public static void main(String[] args) {
        Manager<Student> manager = new Manager<Student>();
        manager.add(new Student("小鱼", 20));
        manager.add(new Student("小黑", 30));
        manager.add(new Student("SF", 21));

        System.out.println("get--->" + manager.get(1));

        manager.sop();
    }
}
```

泛型能代表的太多了，是否能给它一些限制呢，答案也是肯定的。下面来看泛型的上下限。

确定上限

什么叫确定上限，字面意思就是你的上限我已经给你定好了，你不可能再超出这个范围，那就有用到一个关键字 extends，我们让 T（泛型）extends 某一个类，那是不是这个泛型的上限就被你决定了。
下面我们看代码。

- 定义基类

```
    /**
     * 基类
     */
    public class Person {

    int age;
    String name;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

- 定义子类

```
public class Child extends Person {

    public Child(String name, int age) {
        super(name, age);
    }
}
```

- 还有一个不相关的类

```
    public class Dog {

        private String name;
        private int age;

        public Dog(String name, int age) {
            this.name = name;
            this.age = age;
        }
    }
```

- 定义泛型类

```
    public class Fun1<T extends Person> {//确定上限，(泛型类的建模很重要)
        private T datas;

        public T getDatas() {
            return datas;
        }

        public void setDatas(T datas) {
            this.datas = datas;
        }
    }
```

- 运行(接收的引用类型要么是Person类，要么是Person的子类： 确定上限)

确定下限

感觉用的不多，关键字 super

案例

```
    public class Demo {
        public static void main(String[] args) {
            Collection<Student> cs = new ArrayList<Student>();
            cs.add(new Student("李xx", 20));
            cs.add(new Student("xxx", 19));
            cs.add(new Student("hhahah", 20));
            sop2(cs);

        }

        //接收的引用类型要么是Student类，要么是Student的父类：   确定下限
        static void sop2(Collection<? super Student> cs) {
            Iterator<?> iterator = cs.iterator();
            while (iterator.hasNext()) {
                System.out.println(iterator.next());
            }
        }
    }
```

让我们带着泛型的目光回顾 TreeSet中涉及Collections、Comparator、Comparable

我们说过TreeSet存储的元素是要支持可排序的，那他有两种方式，一是实现Comparable接口，二是在构造TreeSet实例的时候传一个Comparator实例。
我们先看源码：

- Comparable

```
package java.lang;

public interface Comparable<T> {//一个泛型接口
    int compareTo(T var1);
}
```

这就是Comparable所有的代码，简单吧.

- Comparator代码巨多，我们也就只看一行

```
 public interface Comparator<T> {
     int compare(T var1, T var2);
     ......
  }
```

和Comparable很像;

- Collections集合工具类，代码巨多，我们也就只看几行

```
public static <T extends Comparable<? super T>> void sort(List<T> var0) {
        var0.sort((Comparator)null);
    }

public static <T> void sort(List<T> var0, Comparator<? super T> var1) {
     var0.sort(var1);
   }
```

当初也许你会很好奇，这个类凭什么帮你排序，现在你知道了吧，你所传的实例都被泛型限定好了，这里出现了一个以前没说过的"?"号，我们先忽略它。
两个sort方法，要么实现Comparable，要么是Comparator，但有一点他们是统一的，就是都是用确定下限的泛型方式。加深印象!

案例 Comparator泛型的确定下限

- Animal(基类)

```
public class Animal {
     int age;
     String name;

    public Animal(int age, String name) {
        this.age = age;
        this.name = name;
    }

    @Override
    public String toString() {
        return "[" + this.name + "\t" + this.age + "]";
    }
}
```

- Cat（子类）

```
public class Cat extends Animal {

    public Cat(int age, String name) {
        super(age, name);
    }

    @Override
    public String toString() {
        return super.age + "";
    }
}
```

- 运行

还有一个?号等着去解决...

? 通配符

我们在Collections 的源码中看到了好多Comparable<? super T>，那这个？和T有什么关系呢。

? 和T没有什么必然的联系。我们要使用T，则必须在定义类的时候申明T，像 class Fun<T>,然后在类里可以使用T这一类型，

而? 则表示通配（填充），表示通配，表示通配，而不是定义，因此我们使用之前不用定义它，表示通配！就如 Class<?> cls = Person.class.getClass();

```
Class<T>在实例化的时候，T要替换成具体类
Class<?>它是个通配泛型，?可以代表任何类型

<? extends T>受限统配，表示T的一个未知子类。
<? super T>下限统配，表示T的一个未知父类。
```

参考

夯实JAVA基本之一 —— 泛型详解(1)(2):基本使用
计算机思维逻辑-泛型 (上)(中)(下)

Android之路
2楼 2017.04.20 16:17

泛型擦除没有讲喔....

赞 回复

向日花开 作者
2017.04.20 16:29

@Android之路 确实没写,我之前就很少接触泛型擦除😰

回复

添加新评论

被以下专题收入，发现更多相似内容

Java学习笔记 技术干货

推荐阅读

更多精彩内容 >

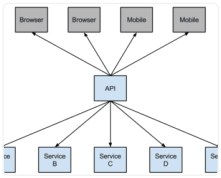
Java初级面试题

1. Java基础部分 基础部分的顺序：基本语法，类相关的语法，内部类的语法，继承相关的语法，异常的语法，线程的语...

会飞的鱼69 阅读 23,382 评论 18 赞 390

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...



卡卡罗2017 阅读 83,866 评论 14 赞 122

百战程序员V1.2——尚学堂旗下高端培训_ Java1573题

百战程序员_ Java1573题 QQ群：561832648489034603 掌握80%年薪20万掌握50%年薪...

Albert陈凯 阅读 11,797 评论 2 赞 32

Java泛型详解

一、引入泛型机制的原因 假如我们想要实现一个String数组，并且要求它可以动态改变大小，这时我们都会想到用Arr...

南南啦啦啦 阅读 149 评论 0 赞 1

想写封信给未知的你

【故事总有残缺】 广东的夏天总是比故乡的夏天要长一点。 于是我并没有像故乡的小伙伴一样，添上一件外套。相反我还是每...

一灯照我江湖静one 阅读 138 评论 1 赞 1