## 为什么要使用泛型和迭代器 + 面试题

### 泛型

#### 1）为什么要用泛型？

在泛型没有诞生之前，我们经常会遇到这样的问题，如以下代码所示：

ArrayList arrayList = new ArrayList();  
arrayList.add("Java");  
arrayList.add(24);  
for (int i = 0; i < arrayList.size(); i++) {  
 String str = (String) arrayList.get(i);  
 System.out.println(str);  
}

看起来好像没有什么大问题，也能正常编译，但真正运行起来就会报错：

Exception in thread “main” java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String

at xxx(xxx.java:12)

类型转换出错，当我们给 ArrayList 放入不同类型的数据，却使用一种类型进行接收的时候，就会出现很多类似的错误，可能更多的时候，是因为开发人员的不小心导致的。那有没有好的办法可以杜绝此类问题的发生呢？这个时候 Java 语言提供了一个很好的解决方案——“泛型”。

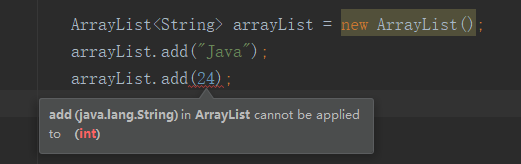
#### 2）泛型介绍

**泛型** ：泛型本质上是类型参数化，解决了不确定对象的类型问题。  
泛型的使用，请参考以下代码：

ArrayList<String> arrayList = new ArrayList();  
arrayList.add("Java");

这个时候如果给 arrayList 添加非 String 类型的元素，编译器就会报错，提醒开发人员插入相同类型的元素。

报错信息如下图所示：



enter image description here

这样就可以避免开头示例中，类型不一致导致程序运行过程中报错的问题了。

#### 3）泛型的优点

泛型的优点主要体现在以下三个方面。

* 安全：不用担心程序运行过程中出现类型转换的错误。
* 避免了类型转换：如果是非泛型，获取到的元素是 Object 类型的，需要强制类型转换。
* 可读性高：编码阶段就明确的知道集合中元素的类型。

### 迭代器（Iterator）

#### 1）为什么要用迭代器？

我们回想一下，在迭代器（Iterator）没有出现之前，如果要遍历数组和集合，需要使用方法。

数组遍历，代码如下：

String[] arr = new String[]{"Java", "Java虚拟机", "Java中文社群"};  
for (int i = 0; i < arr.length; i++) {  
 String item = arr[i];  
}

集合遍历，代码如下：

List<String> list = new ArrayList<String>() {{  
 add("Java");  
 add("Java虚拟机");  
 add("Java中文社群");  
}};  
for (int i = 0; i < list.size(); i++) {  
 String item = list.get(i);  
}

而迭代器的产生，就是为不同类型的容器遍历，提供标准统一的方法。

迭代器遍历，代码如下：

Iterator iterator = list.iterator();  
while (iterator.hasNext()) {  
 Object object = iterator.next();  
 // do something  
}

**总结** ：使用了迭代器就可以不用关注容器的内部细节，用同样的方式遍历不同类型的容器。

#### 2）迭代器介绍

迭代器是用来遍历容器内所有元素对象的，也是一种常见的设计模式。

迭代器包含以下四个方法。

* hasNext():boolean —— 容器内是否还有可以访问的元素。
* next():E —— 返回下一个元素。
* remove():void —— 删除当前元素。
* forEachRemaining(Consumer super E>):void —— JDK 8 中添加的，提供一个 lambda 表达式遍历容器元素。

迭代器使用如下：

List<String> list = new ArrayList<String>() {{  
 add("Java");  
 add("Java虚拟机");  
 add("Java中文社群");  
}};  
Iterator iterator = list.iterator();  
// 遍历  
while (iterator.hasNext()){  
 String str = (String) iterator.next();  
 if (str.equals("Java中文社群")){  
 iterator.remove();  
 }  
}  
System.out.println(list);

程序执行结果：

[Java, Java虚拟机]

forEachRemaining 使用如下：

List<String> list = new ArrayList<String>() {{  
 add("Java");  
 add("Java虚拟机");  
 add("Java中文社群");  
}};  
// forEachRemaining 使用  
list.iterator().forEachRemaining(item -> System.out.println(item));

### 相关面试题

#### 1.为什么迭代器的 next() 返回的是 Object 类型？

答：因为迭代器不需要关注容器的内部细节，所以 next() 返回 Object 类型就可以接收任何类型的对象。

#### 2.HashMap 的遍历方式都有几种？

答：HashMap 的遍历分为以下四种方式。

* 方式一：entrySet 遍历
* 方式二：iterator 遍历
* 方式三：遍历所有的 key 和 value
* 方式四：通过 key 值遍历

以上方式的代码实现如下：

Map<String, String> hashMap = new HashMap();  
hashMap.put("name", "老王");  
hashMap.put("sex", "你猜");  
// 方式一：entrySet 遍历  
for (Map.Entry item : hashMap.entrySet()) {  
 System.out.println(item.getKey() + ":" + item.getValue());  
}  
// 方式二：iterator 遍历  
Iterator<Map.Entry<String, String>> iterator = hashMap.entrySet().iterator();  
while (iterator.hasNext()) {  
 Map.Entry<String, String> entry = iterator.next();  
 System.out.println(entry.getKey() + ":" + entry.getValue());  
}  
// 方式三：遍历所有的 key 和 value  
for (Object k : hashMap.keySet()) {  
 // 循环所有的 key  
 System.out.println(k);  
}  
for (Object v : hashMap.values()) {  
 // 循环所有的值  
 System.out.println(v);  
}  
// 方式四：通过 key 值遍历  
for (Object k : hashMap.keySet()) {  
 System.out.println(k + ":" + hashMap.get(k));  
}

#### 3.以下关于泛型说法错误的是？

A：泛型可以修饰类  
B：泛型可以修饰方法  
C：泛型不可以修饰接口  
D：以上说法全错

答：选 C，泛型可以修饰类、方法、接口、变量。  
例如：

public interface Iterable<T> {  
}

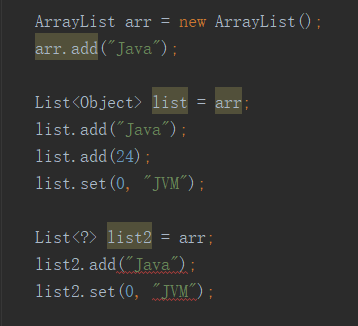
#### 4.以下程序执行的结果是什么？

List<String> list = new ArrayList<>();  
List<Integer> list2 = new ArrayList<>();  
System.out.println(list.getClass() == list2.getClass());

答：程序的执行结果是 true。  
题目解析：Java 中泛型在编译时会进行类型擦除，因此 List<String> list 和 List<Integer> list2 类型擦除后的结果都是 java.util.ArrayLis ，进而 list.getClass() == list2.getClass() 的结果也一定是 true。

#### 5. List<Object> 和 List<?> 有什么区别？

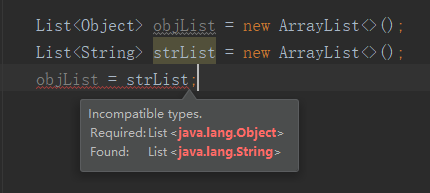
答：List<?> 可以容纳任意类型，只不过 List<?> 被赋值之后，就不允许添加和修改操作了；而 List<Object> 和 List<?> 不同的是它在赋值之后，可以进行添加和修改操作，如下图所示：



enter image description here

#### 6.可以把 List<String> 赋值给 List<Object> 吗？

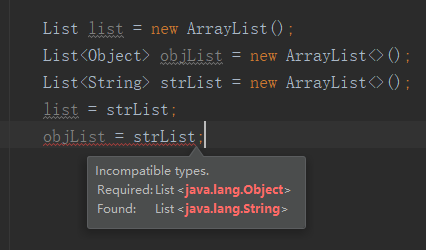
答：不可以，编译器会报错，如下图所示：



enter image description here

#### 7. List 和 List<Object> 的区别是什么？

答： List 和 List<Object> 都能存储任意类型的数据，但 List 和 List<Object> 的唯一区别就是，List 不会触发编译器的类型安全检查，比如把 List<String> 赋值给 List 是没有任何问题的，但赋值给 List<Object> 就不行，如下图所示：



enter image description here

#### 8.以下程序执行的结果是？

List<String> list = new ArrayList<>();  
list.add("Java");  
list.add("Java虚拟机");  
list.add("Java中文社群");  
Iterator iterator = list.iterator();  
while (iterator.hasNext()) {  
 String str = (String) iterator.next();  
 if (str.equals("Java中文社群")) {  
 iterator.remove();  
 }  
}  
while (iterator.hasNext()) {  
 System.out.println(iterator.next());  
}  
System.out.println("Over");

答：程序打印结果是 Over。  
题目解析：因为第一个 while 循环之后，iterator.hasNext() 返回值就为 false 了，所以不会进入第二个循环，之后打印最后的 Over。

#### 9.泛型的工作原理是什么？为什么要有类型擦除？

答：泛型是通过类型擦除来实现的，类型擦除指的是编译器在编译时，会擦除了所有类型相关的信息，比如 List<String> 在编译后就会变成 List 类型，这样做的目的就是确保能和 Java 5 之前的版本（二进制类库）进行兼容。

### 总结

通过本文知道了泛型的优点：安全性、避免类型转换、提高了代码的可读性。泛型的本质是类型参数化，但编译之后会执行类型擦除，这样就可以和 Java 5 之前的二进制类库进行兼容。本文也介绍了迭代器（Iterator）的使用，使用迭代器的好处是不用关注容器的内部细节，用同样的方式遍历不同类型的容器。

[点击此处下载本文源码](https://github.com/vipstone/java-%20interview/tree/master/interview-code/src/main/java/com/interview)

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持