第五章 泛型

2018年7月12日

17:48

# 第23条：请不要在原生代码使用原生类型

如果使用原生类型，就失掉了泛型在安全性和表达性方面的所有优势。

List<Object>这样的参数化类型比List这样的原生类型更具类型安全性。

泛型信息信息在**运行时**会被被擦除。

## 

## 不在新代码中使用原生类型的例外：

* 在类文字中必须使用原生态类型。

List.calss，String[].class和int.class都合法，但是List<String.class>和List<?>.class则不合法。

* 在参数化类型而非**无限制通配符**上使用instanceof操作符是非法。

o instanceof Set<T>或o instanceof Set<Object>是不合法的，但是 o instanceof Set<?> ，但是没必要。

if( o instanceof Set) { Set<?> m = Set<?> o } 一旦确定这个o是Set, 必须将它转换成通配符类型，而不是原生类型。

Set<Object>是个参数类型，表示可以包含任何对象类型的集合；

Set<?>则是一个人通配符类型，表示只能包含某种未知对象类型的一个集合。

Set则是原生态类型，它脱离了泛型系统，不安全。

# 第24条：消除非受检警告

泛型编程时会遇到许多编译器警告：

* 非受检强制转化警告
* 非受检方法调用警告
* 非受检普通函数数组创建警告
* 非受检转换警告

如无法消除警告，同时可以证明引起警告的代码是类型安全的，可以用一个@SuppresWarnings("unchecked")注解来禁止这条警告。

SuppresWarnings注解应该始终在尽可能小的范围中使用SuppresWarnings注解。

每当使用SuppresWarnings注解时，都要用注释把禁止该警告的原因记录下来。

# 第25条：列表优先于数组。

## 数组是协变的，泛型则是不可变的。

如果Sub为Super的子类型，那么数组类型Sub[]就是Super[]的子类

List<Object>与List<Long>之间没有父子类关系。

## 数组是具体化，数组运行时才知道并检查他们的元素而类型约束。

## 泛型则只在编译时强化它们的类型信息，在运行时擦除。

创建泛型，参数化的或者类型参数的数组是非法的。

# 第26条：优先考虑泛型

使用泛型比使用需要在客户端代码中进行转换的类型转化的类型来得更加安全，也更加容易。

# 第27条：优先考虑泛型方法

泛型方法比要求客户端转换输入参数并返回值的方法更加安全和容易。

通过将方法泛型化来确保方法可以不用转换类型。

# 第28条：利用优先通配符来提升API的灵活性。

为了获得最大限度的灵活性，要在表示生产者或者消费者的输入参数上使用通配符类型。

producer-extends, consumer-super(PECS)

如果类的用户必须考虑通配符类型，类的API或许就会出错。

# 第29条：优先考虑类型安全的异构容器。

将键进行参数化而不是将容器参数化，把参数化的键提交给容器来插入或者获取值的容器是异构容器。

异构容器的限性：

恶意的客户可以通过Class对象轻松破坏异构容器的类型安全。

异构容器不可用在具体化的类型中，即不可以保存List<String>等类型。