Java泛型概念

Java中与泛型相关的接口 之 术语定义

JSON_NULL 关注

0.096 2017.12.28 17:08:28 字数 846 阅读 573

在Java泛型编程中，很多单词从英文翻译中文后会变味，令人难以理解。在很多讲解Java泛型的中文作品中，对于同一英文单词的翻译也是各不相同，阅读时容易产生误解。在接下来的一段时间，我会针对Java中的泛型、注解等知识点进行学习，期间会出一些文章，作为学习的总结，也方便后来者参考。为了能够统一口径，避免产生误而浪费精力。在此对一些名称做统一说明，暂且称之为术语定义吧。

ParameterizedType

这是在 java.lang.reflect 包中一个接口的名称，很多文章中把它翻译为“参数化类型”，我通过参阅多方资料发现其实这个接口就是对“泛型实例”的说明。所以在之后的文章中，我就把这个接口称之为“泛型实例”。

其实这个接口就是在说明一个带参数的类型，如：Collection<String>，Map<String,Object>等。这看似大家平常所说的泛型，但其实不然。我们大家平时所说的泛型是Collection<E>和Map<K,V>等，而ParameterizedType表示的是泛型（Collection<E>和Map<K,V>等）的一种实例（Collection<String>和Map<String,Object>）。

ParameterizedType

TypeVariable

这是在 java.lang.reflect 包中一个接口的名称，其全名为：TypeVariable<D extends GenericDeclaration>。很多文章中翻译为“类型变量”，其实这种翻译也没有错。在我阅读了大量的有关TypeVariable的说明后发现其实称它为“泛型变量”更为合适。

其实这个接口是在说明“泛型”中的可变量，也就是Collection<E>和Map<K,V>中的E，K和V。

TypeVariable<D extends GenericDeclaration>

GenericDeclaration

这也是java.lang.reflect包中的一个接口，这个接口在很多文章中的翻译是“通用声明”，我看后直接是N脸蒙B的状态，完全不知道他在说什么。经常大量阅读资料后慢慢其解了，应该称它为“可以声明范型变量的实体”。

在他的定义中说的很明白：“只有实现了这个接口的‘实体’才能声明‘范型变量’”。实现了这个接口的“实体”有哪些呢？如下所示：Class，Constructor，Method。

GenericDeclaration

GenericArrayType

这个也是java.lang.reflect中的接口，如果你翻译成“通用数组类型”那就大错特错了。其实它是用来描述形如A<T>[]或T[]类型的。如此看来称之为“泛型数组”更为适合。

GenericArrayType

WildcardType

这个是java.lang.reflect中的接口，造成不要翻译作“通配符类型”，其实它是用来描述“泛型”中的通配符表达式（也可以叫泛型参数表达式）的。用于限定“泛型参数”的类型。形如：？extends classA、？ super classB。

在以后的文章中就称呼它为“泛型参数表达式”吧。

WildcardType

Type及其子接口的来历


JSON_NULL 关注
0.307 2017.12.28 17:09:48 字数 666 阅读 543

泛型出现之前

没有泛型的时候，只有所谓的原始类型。此时，所有的原始类型都通过字节码文件类Class类进行抽象。Class类的一个具体对象就代表一个指定的原始类型。

泛型出现之后

泛型出现之后，扩充了数据类型。从只有原始类型扩充了参数化类型（ParameterizedType）、类型变量类型（TypeVariable）、泛型限定的参数化类型 (含通配符+通配符限定表达式)（WildcardType）、泛型数组类型（GenericArrayType）。

与泛型有关的类型不能和原始类型统一到Class的原因

产生泛型擦除的原因

为了使用泛型的优势又不真正引入泛型，Java采用泛型擦除的机制来引入泛型。Java中的泛型仅仅是给编译器javac使用的，确保数据的安全性和免去强制类型转换的麻烦。但是，一旦编译完成，所有的和泛型有关的类型全部擦除。

Class不能表达与泛型有关的类型

因此，与泛型有关的泛型实例（ParameterizedType）、类型变量（TypeVariable）、泛型参数表达式 (含通配符+通配符限定表达式)（WildcardType）、泛型数组（GenericArrayType）这些类型全部被打回原形，在字节码文件中全部都是泛型被擦除后的原始类型，并不存在和自身类型一致的字节码文件。所以和泛型相关的新扩充进来的类型不能被统一到Class类中。

与泛型有关的类型在Java中的表示

为了通过反射操作这些类型以迎合实际开发的需要，Java就新增了ParameterizedType，GenericArrayType，TypeVariable 和WildcardType几种类型来代表不能被归一到Class类中的类型但是又和原始类型齐名的类型。

Type的引入：统一与泛型有关的类型和原始类型Class

为了程序的扩展性，最终引入了Type接口作为Class，ParameterizedType，GenericArrayType，ypeVariable和WildcardType这几种类型的总的父接口。这样实现了Type类型参数可以接受以上五种子类的实参，而以上五种类型的返回值可以用Type类型的变量来接收。

从上面看到，Type的出现仅仅起到了通过多态来达到程序扩展性提高的作用，没有其他的作用。因此Type接口的源码中没有任何方法。

Java中与泛型相关的接口 之 ParameterizedType


JSON_NULL 关注
0.472 2017.12.28 17:11:01 字数 623 阅读 2,103

在阅读本文之前可以先阅读以下三篇，以便对Java中的泛型有一个全局的认识：

1. Java中与泛型相关的接口 之 术语定义
2. Java中与泛型相关的接口 之 综述
3. Type及其子接口的来历

简介

ParameterizedType是Type的子接口，表示一个有参数的类型，例如Collection<T>，Map<K,V>等。但实现上 ParameterizedType并不直接表示Collection<T>和Map<K,V>等，而是表示Collection<String>和Map<String,String>等这种具体的类型。是不是看着眼熟，其实这就是我们常说的泛型。而ParameterizedType代表的是一个泛型的实例，我们就称ParameterizedType为“泛型实例”吧。

当创建泛型P（如：Collection<String>）时，将解析P实例化的泛型类型声明（如：Collection<T>），并且递归地创建P的所有泛型参数（如：String）。

实现这个接口的“类”必须实现一个equals()方法，该方法将任何“泛型类型”（如：Collection<T>）声明相同且“类型参数”（如：String）也相同的两个“类”等同起来。

Type[] getActualTypeArguments()

获取“泛型实例”中<>里面的“泛型变量”（也叫类型参数）的值，这个值是一个类型。因为可能有多个“泛型变量”（如：Map<K,V>）,所以返回的是一个Type[]。

注意：无论<>中有几层<>嵌套，这个方法仅仅脱去最外层的<>，之后剩下的内容就作为这个方法的返回值，所以其返回值类型是不确定的。

煮个栗子：

1. List<ArrayList> a1;//返回ArrayList，Class类型
2. List<ArrayList<String>> a2;//返回ArrayList<String>，ParameterizedType类型
3. List<T> a3;//返回T，TypeVariable类型
4. List<? extends Number> a4; //返回`? extends Number`，WildcardType类型
5. List<ArrayList<String>[]> a5;//返回ArrayList<String>[]，GenericArrayType 类型

Type getRawType()

返回最外层<>前面那个类型，即Map<K ,V>的Map。

Type getOwnerType()

获得这个类型的所有者的类型。这主要是对嵌套定义的内部类而言的，例如于对java.util.Map.Entry<K,V>来说，调用getOwnerType方法返回的就是interface java.util.Map。

如果当前类不是内部类，而是一个顶层类，那么getOwnerType方法将返回null。

Java中与泛型相关的接口 之 TypeVariable<D extends GenericDeclaration>

JSON_NULL 关注
2017.12.28 14:55:13 字数 606 阅读 632

在阅读本文之前可以先阅读以下三篇，以便对Java中的泛型有一个全局的认识：

1. Java中与泛型相关的接口 之 术语定义
2. Java中与泛型相关的接口 之 综述
3. Type及其子接口的来历

简介

TypeVariable是"类型变量"（或者叫"泛型变量"更准确些）的通用的顶级接口。在泛型编程中会用到"泛型变量"来描述类型，或者说是用来表示泛型。一般用大写字母作为类型变量，比如K、V、E等。

说到TypeVariable<D extends GenericDeclaration>就不得不提起java泛型中另一个比较重要的接口对象，GenericDeclaration接口对象。该接口用来定义哪些对象上是可以声明（定义）"范型变量"，所谓"范型变量"就是<E extends List>或者<E>, 也就是TypeVariable<D extends GenericDeclaration>这个接口的对应的对象，TypeVariable<D extends GenericDeclaration>中的D是extends GenericDeclaration的，用来通过"范型变量"反向获取拥有这个变量的GenericDeclaration。

【注意】类型变量声明（定义）的时候不能有下限（既不能有super），否则编译报错。为什么？T extends classA表示泛型有上限classA，当然可以，因为这样，每一个传进来的类型必定是classA（具有classA的一切属性和方法），但若是T super classA，传进来的类型不一定具有classA的属性和方法，当然就不适用于泛型。

Type[] getBounds()

获得该"范型变量"的上限（上边界），若无显式定义（extends），默认为Object。类型变量的上限可能不止一个，因为可以用&符号限定多个（这其中有且只能有一个为类或抽象类，且必须放在extends后的第一个，即若有多个上边界，则第一个&后必为接口）。

下面是一个例子：

```
package com.ibestcode.wfso.web.blog.Controller;

import java.util.Map;
import java.lang.reflect.*;
public class Test<K extends Integer & Map, V>{
  K key;
  V value;
  public static void main(String[] args) throws Exception
  {
    Type[] types = Test.class.getTypeParameters();
    for(Type type : types){
      TypeVariable t = (TypeVariable)type;
      System.out.println(t.getGenericDeclaration());
      int size = t.getBounds().length;
      System.out.println(t.getBounds()[size - 1]);
      System.out.println(t.getName() + "\n-------------分割线-------------");
    }
  }
}

// 下面是运行结果
class com.ibestcode.wfso.web.blog.Controller.Test
interface java.util.Map
K
```

```
-------------分割线-------------
class com.ibestcode.wfso.web.blog.Controller.Test
class java.lang.Object
V
-------------分割线-------------

Process finished with exit code 0
```

D getGenericDeclaration()

获得声明（定义）这个“范型变量”的类型及名称，即如：

```
//当声明这个“范型变量”的GenericDeclaration是一个类时
class com.xxx.xxx.classA

//当声明这个“范型变量”的GenericDeclaration是一个方法时
public void com.fcc.test.Main.test(java.util.List)

//当声明这个“范型变量”的GenericDeclaration是一个构造器时
public com.fcc.test.Main()
```

String getName()

获得这个“范型变量”在声明（定义）时候的名称。如：K、V、E等。

AnnotatedType[] getAnnotatedBounds()

还不知道什么用，肯定和注解相关。

Java中与泛型相关的接口 之 GenericDeclaration


JSON_NULL 关注
2017.12.28 17:10:13 字数 415 阅读 516

在阅读本文之前可以先阅读以下三篇，以便对Java中的泛型有一个全局的认识：

1. Java中与泛型相关的接口 之 术语定义
2. Java中与泛型相关的接口 之 综述
3. Type及其子接口的来历

简介

GenericDeclaration接口继承了AnnotatedElement接口，是所有“可以声明（定义）范型变量”的实体（如Class，Constructor，Method）的公共接口。也就是说只有实现了这个接口的才能在对应“实体”上声明“范型变量”。所谓范型变量就是<E extends List>或者<E>, 也就是TypeVariable<D extends GenericDeclaration>这个接口对应的对象，TypeVariable<D extends GenericDeclaration>中的D是继承自 GenericDeclaration的，用来通过范型变量（TypeVariable）反向获取拥有这个变量的GenericDeclaration。

目前实现GenericDeclaration接口的类包括Class（类）, Method（方法）, Constructor（构造器），也就是说只能在这几种对象上进行范型变量的声明（定义）。GenericDeclaration的直接实现子类没有Field类，所以属性上面不能定义类型变量。GenericDeclaration的接口方法getTypeParameters用来逐个获取该GenericDeclaration的“范型变量”声明。

TypeVariable<?>[] getTypeParameters()

获取当前“实体”上声明的“泛型变量"。

按被声明的顺序返回一个TypeVariable对象的数组，其中TypeVariable对象是由GenericDeclaration对象进行声明的。如果底层没有使用GenericDeclaration声明“范型变量”，那么返回一个长度为0的数组。

下面是一个例子：

```
public class Main<K extends classA & interfaceB, V> {
    classA<K>[][] key;
    V value;
    public static void main(String[] args) throws Exception{
        TypeVariable[] types = Main.class.getTypeParameters();
        for(TypeVariable type : types){
            System.out.println(type.getName());
        }
    }
}
//输出结果
K
V
```

Java中与泛型相关的接口 之 GenericArrayType

JSON_NULL 关注

2017.12.28 17:11:20 字数 206 阅读 671

在阅读本文之前可以先阅读以下三篇，以便对Java中的泛型有一个全局的认识：

1. Java中与泛型相关的接口 之 术语定义
2. Java中与泛型相关的接口 之 综述
3. Type及其子接口的来历

简介

GenericArrayType是Type的子接口，用于表示"泛型数组"，描述的是形如：A<T>[]或T[]的类型。其实也就是描述ParameterizedType类型以及TypeVariable类型的数组，即形如：classA<T>[][]、T[]等。

Type getGenericComponentType()

获取"泛型数组"中元素的类型，要注意的是：无论从左向右有几个[]并列，这个方法仅仅脱去最右边的[]之后剩下的内容就作为这个方法的返回值。

Java中与泛型相关的接口 之 WildcardType


JSON_NULL 关注
2017.12.28 17:11:38 字数 281 阅读 368

在阅读本文之前可以先阅读以下三篇，以便对Java中的泛型有一个全局的认识：

1. Java中与泛型相关的接口 之 术语定义
2. Java中与泛型相关的接口 之 综述
3. Type及其子接口的来历

简介

WildcardType是Type的子接口，用于描述形如“? extends classA” 或 “? super classB”的“泛型参数表达式”。

Type[] getUpperBounds()

获取泛型表达式上界.

根据API的注释提示：现阶段通配符表达式仅仅接受一个上边界或者下边界，这个和定义“范型变量”的时候可以指定多个上边界是不一样。但是API说了，为了保持扩展性，这里返回值类型写成了数组形式。实际上现在返回的数组的大小就是1，通配符?指定多个上边界或者下边界现在是会编译出错的（jdk1.7是这样的，至于7及以后就不知道了）。

Type[] getLowerBounds()

获取泛型表达式下界。