原 java 泛型详解-绝对是对泛型方法讲解最详细的，没有之一

2016年12月30日 11:44:29 VieLei 阅读数：235091

> 对java的泛型特性的了解仅限于表面的浅浅一层，直到在学习设计模式时发现有不了解的用法，才想起详细的记录一下。
> 本文参考java 泛型详解、Java中的泛型方法、 java泛型详解

1. 概述

泛型在java中有很重要的地位，在面向对象编程及各种设计模式中有非常广泛的应用。

什么是泛型？为什么要使用泛型?

> **泛型，即“参数化类型”。一提到参数，最熟悉的就是定义方法时有形参，然后调用此方法时传递实参。那么参数化类型怎么理解呢？将类型由原来的具体的类型参数化，类似于方法中的变量参数，此时类型也定义成参数形式（可以称之为类型形参），然后在使用时传入具体的类型（类型实参）。**
> **泛型的本质是为了参数化类型（在不创建新的类型的情况下，通过泛型指定的不同类型来控制形参具体限制的类型）**。也就是说在泛型使用过程中，操作的数据类型被指定为一个参数，这种参数类型可以用在类、接口和方法中，分别被称为泛型类、泛型接口、泛型方法。

2. 一个栗子

一个被举了无数次的例子：

```
1  List arrayList = new ArrayList();
2  arrayList.add("aaaa");
3  arrayList.add(100);
4
5  for(int i = 0; i< arrayList.size();i++){
6      String item = (String)arrayList.get(i);
7      Log.d("泛型测试","item = " + item);
8  }
```

毫无疑问，程序的运行结果会以崩溃结束：

```
1  java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String
```

ArrayList可以存放任意类型，例子中添加了一个String类型，添加了一个Integer类型，再使用时都以String的方式使用了。为了解决类似这样的问题（在编译阶段就可以解决），泛型应运而生。

我们将第一行声明初始化list的代码更改一下，编译器会在编译阶段就能够帮我们发现类似这样的问题。

```
1  List<String> arrayList = new ArrayList<String>();
2  ...
3  //arrayList.add(100); 在编译阶段，编译器就会报错
```

3. 特性

泛型只在编译阶段有效。看下面的代码：

```
1  List<String> stringArrayList = new ArrayList<String>();
```

```
List<Integer> integerArrayList = new ArrayList<Integer>();

Class classStringArrayList = stringArrayList.getClass();
Class classIntegerArrayList = integerArrayList.getClass();

if(classStringArrayList.equals(classIntegerArrayList)){
    Log.d("泛型测试","类型相同");
}
```

输出结果： D/泛型测试： 类型相同 。

通过上面的例子可以证明，在编译之后程序会采取去泛型化的措施。也就是说Java中的泛型，只在编译阶段有效。在编
验泛型结果后，会将泛型的相关信息擦出，并且在对象进入和离开方法的边界处添加类型检查和类型转换的方法。也就
会进入到运行时阶段。

对此总结成一句话：泛型类型在逻辑上看以看成是多个不同的类型，实际上都是相同的基本类型。

4. 泛型的使用

泛型有三种使用方式，分别为：泛型类、泛型接口、泛型方法

4.3 泛型类

泛型类型用于类的定义中，被称为泛型类。通过泛型可以完成对一组类的操作对外开放相同的接口。最典型的就是各种
List、Set、Map。

泛型类的最基本写法（这么看可能会有点晕，会在下面的例子中详解）：

```
class 类名称 <泛型标识：可以随便写任意标识号，标识指定的泛型的类型>{
  private 泛型标识 /*（成员变量类型）*/ var;
  .....

  }
}
```

一个最普通的泛型类：

```
//此处T可以随便写为任意标识，常见的如T、E、K、V等形式的参数常用于表示泛型
//在实例化泛型类时，必须指定T的具体类型
public class Generic<T>{
    //key这个成员变量的类型为T,T的类型由外部指定
    private T key;

    public Generic(T key) { //泛型构造方法形参key的类型也为T，T的类型由外部指定
        this.key = key;
    }

    public T getKey(){ //泛型方法getKey的返回值类型为T，T的类型由外部指定
        return key;
    }
}
```



```
//泛型的类型参数只能是类类型（包括自定义类），不能是简单类型
//传入的实参类型需与泛型的类型参数类型相同，即为Integer.
Generic<Integer> genericInteger = new Generic<Integer>(123456);

//传入的实参类型需与泛型的类型参数类型相同，即为String.
Generic<String> genericString = new Generic<String>("key_vlaue");
Log.d("泛型测试","key is " + genericInteger.getKey());
```

最新文章

Android自定义SeekBar，基础但是要比大

多数文章更综合一点，有一些常见的问题解决

一、Rxjava从头学：响应式编程

2、使用最广泛的模式-单例模式

1、写在最前：设计模式六大原则

java 枚举详解

个人分类

android 4篇

java基础 2篇

设计模式 2篇

归档

2018年6月 1篇

2017年10月 1篇

2016年12月 5篇

热门文章

java 泛型详解-绝对是对泛型方法讲解最详细的，没有之一
阅读数 235047

RabbitMQ 客户端（for andorid）使用详解
阅读数 5962

java 枚举详解
阅读数 1311

1、写在最前：设计模式六大原则
阅读数 1230

2、使用最广泛的模式-单例模式
阅读数 712

最新评论

java 泛型详解-绝对是对泛型方...
u014672466：有用，谢谢。

java 泛型详解-绝对是对泛型方...
qqHJQS：以转载到【哇哦窝】公众号，并注明出处！ 如有不便，请告知~

java 泛型详解-绝对是对泛型方...
ADKII：在4.3.2类中的所有方法中，定义Apple和Person实例的时候会报“无法从静态上下文引用 ...

java 泛型详解-绝对是对泛型方...
u013696874：4.6.1 里面66行编译错误 那为什么4.6.2的24行可以这样声明?

java 泛型详解-绝对是对泛型方...
cadi2011：写的牛逼啊

```
8  Log.d("泛型测试","key is " + genericString.getKey());
```

```
1  12-27 09:20:04.432 13063-13063/? D/泛型测试: key is 123456
2  12-27 09:20:04.432 13063-13063/? D/泛型测试: key is key_vlaue
```

.的泛型类，就一定要传入泛型类型实参么？并不是这样，在使用泛型的时候如果传入泛型实参，则会根据传入的
此时泛型才会起到本应起到的限制作用。如果不传入泛型类型实参的话，在泛型类中使用泛型的方法或成员变量
类型。

个例子：

```
1  Generic generic = new Generic("111111");
2  Generic generic1 = new Generic(4444);
3  Generic generic2 = new Generic(55.55);
4  Generic generic3 = new Generic(false);
5
6  Log.d("泛型测试","key is " + generic.getKey());
7  Log.d("泛型测试","key is " + generic1.getKey());
8  Log.d("泛型测试","key is " + generic2.getKey());
9  Log.d("泛型测试","key is " + generic3.getKey());
```

```
1  D/泛型测试: key is 111111
2  D/泛型测试: key is 4444
3  D/泛型测试: key is 55.55
4  D/泛型测试: key is false
```

注意：

- 1. 泛型的类型参数只能是类类型，不能是简单类型。
- 1. 不能对确切的泛型类型使用instanceof操作。如下面的操作是非法的，编译时会出错。

```
1  if(ex_num instanceof Generic<Number>){
2  }
```

泛型接口

接口与泛型类的定义及使用基本相同。泛型接口常被用在各种类的生产器中，可以看一个例子：

```
1  //定义一个泛型接口
2  public interface Generator<T> {
3      public T next();
4  }
```

现泛型接口的类，未传入泛型实参时：

```
1  /**
2   * 未传入泛型实参时，与泛型类的定义相同，在声明类的时候，需将泛型的声明也一起加到类中
3   * 即：class FruitGenerator<T> implements Generator<T>{
4   * 如果不声明泛型，如：class FruitGenerator implements Generator<T>，编译器会报错："Unknown class"
5   */
6  class FruitGenerator<T> implements Generator<T>{
7      @Override
8      public T next() {
9          return null;
0      }
```

```
}
```

现泛型接口的类，传入泛型实参时：

```
/**
 * 传入泛型实参时：
 * 定义一个生产器实现这个接口,虽然我们只创建了一个泛型接口Generator<T>
 * 但是我们可以为T传入无数个实参，形成无数种类型的Generator接口。
 * 在实现类实现泛型接口时，如已将泛型类型传入实参类型，则所有使用泛型的地方都要替换成传入的实参类型
 * 即：Generator<T>，public T next();中的的T都要替换成传入的String类型。
 */
public class FruitGenerator implements Generator<String> {

    private String[] fruits = new String[]{"Apple", "Banana", "Pear"};

    @Override
    public String next() {
        Random rand = new Random();
```


4.5 泛型通配符

我们知道 Ingeter 是 Number 的一个子类，同时在特性章节中我们也验证过 Generic<Ingeter> 与 Generic<Number> 种基本类型。那么问题来了，在使用 Generic<Number> 作为形参的方法中，能否使用 Generic<Ingeter> 的实例传入于 Generic<Number> 和 Generic<Ingeter> 是否可以看成具有父子关系的泛型类型呢?

为了弄清楚这个问题，我们使用 Generic<T> 这个泛型类继续看下面的例子：

```
public void showKeyValue1(Generic<Number> obj){
    Log.d("泛型测试","key value is " + obj.getKey());
}
```

```
Generic<Integer> gInteger = new Generic<Integer>(123);
Generic<Number> gNumber = new Generic<Number>(456);

showKeyValue(gNumber);

// showKeyValue这个方法编译器会为我们报错：Generic<java.lang.Integer>
// cannot be applied to Generic<java.lang.Number>
// showKeyValue(gInteger);
```

通过提示信息我们可以看到 Generic<Integer> 不能被看作为 `Generic<Number> 的子类。由此可以看出:**同一种泛型（因为参数类型是不确定的），不同版本的泛型类实例是不兼容的。**

回到上面的例子，如何解决上面的问题？总不能为了定义一个新的方法来处理 Generic<Integer> 类型的类，这显然与相违背。因此我们需要一个在逻辑上可以表示**同时**是 Generic<Integer> 和 Generic<Number> 父类的引用类型。由此生。

我们可以将上面的方法改一下：

```
public void showKeyValue1(Generic<?> obj){
    Log.d("泛型测试","key value is " + obj.getKey());
}
```

类型通配符一般是使用？代替具体的类型实参，注意了，**此处'？'是类型实参，而不是类型形参** 。重要说三遍！**此处"**

不是类型形参！ 此处'? '是类型实参，而不是类型形参！再直白点的意思就是，此处的？和Number、String、Intege的类型，可以把？看成所有类型的父类。是一种真实的类型。

可以解决当具体类型不确定的时候，这个通配符就是 **？** ；当操作类型时，不需要使用类型的具体功能时，只使用Obje么可以用？通配符来表未知类型。

4.6 泛型方法

在java中,泛型类的定义非常简单，但是泛型方法就比较复杂了。

> 尤其是我们见到的大多数泛型类中的成员方法也都使用了泛型，有的甚至泛型类中也包含着泛型方法，这样在初学者中非常容易将了。

泛型类，是在实例化类的时候指明泛型的具体类型；泛型方法，是在调用方法的时候指明泛型的具体类型 。

```
/**
 * 泛型方法的基本介绍
 * @param tClass 传入的泛型实参
 * @return T 返回值为T类型
 * 说明：
 *    1）public 与 返回值中间<T>非常重要，可以理解为声明此方法为泛型方法。
 *    2）只有声明了<T>的方法才是泛型方法，泛型类中的使用了泛型的成员方法并不是泛型方法。
 *    3）<T>表明该方法将使用泛型类型T，此时才可以在方法中使用泛型类型T。
 *    4）与泛型类的定义一样，此处T可以随便写为任意标识，常见的如T、E、K、V等形式的参数常用于表示泛型。
 */
public <T> T genericMethod(Class<T> tClass)throws InstantiationException ,
  IllegalAccessException{
        T instance = tClass.newInstance();
        return instance;
}
```

```
Object obj = genericMethod(Class.forName("com.test.test"));
```

4.6.1 泛型方法的基本用法

光看上面的例子有的同学可能依然会非常迷糊，我们再通过一个例子，把我泛型方法再总结一下。

```
public class GenericTest {
    //这个类是个泛型类，在上面已经介绍过
    public class Generic<T>{
        private T key;

        public Generic(T key) {
            this.key = key;
        }

        //我想说的其实是这个，虽然在方法中使用了泛型，但是这并不是一个泛型方法。
        //这只是类中一个普通的成员方法，只不过他的返回值是在声明泛型类已经声明过的泛型。
        //所以在这个方法中才可以继续使用 T 这个泛型。
        public T getKey(){
            return key;
        }

        /**
     * 这个方法显然是有问题的，在编译器会给我们提示这样的错误信息"cannot reslove symbol E"
     * 因为在类的声明中并未声明泛型E，所以在使用E做形参和返回值类型时，编译器会无法识别。
     public E setKey(E key){
         this.key = keu
```

```
22      }
23      */
24      }
25
26      /**
27     * 这才是一个真正的泛型方法。
28     * 首先在public与返回值之间的<T>必不可少，这表明这是一个泛型方法，并且声明了一个泛型T
29     * 这个T可以出现在这个泛型方法的任意位置.
30     * 泛型的数量也可以为任意多个
31     *   如：public <T,K> K showKeyName(Generic<T> container){
32     *      ...
33     *      }
34     */
35      public <T> T showKeyName(Generic<T> container){
36          System.out.println("container key :" + container.getKey());
37          //当然这个例子举的不太合适，只是为了说明泛型方法的特性。
38          T test = container.getKey();
39          return test;
40      }
41
42      //这也不是一个泛型方法，这就是一个普通的方法，只是使用了Generic<Number>这个泛型类做形参而已。
43      public void showKeyValue1(Generic<Number> obj){
44          Log.d("泛型测试","key value is " + obj.getKey());
45      }
46
47      //这也不是一个泛型方法，这也是一个普通的方法，只不过使用了泛型通配符?
48      //同时这也印证了泛型通配符章节所描述的，?是一种类型实参，可以看做为Number等所有类的父类
49      public void showKeyValue2(Generic<?> obj){
50          Log.d("泛型测试","key value is " + obj.getKey());
51      }
52
53       /**
54     * 这个方法是有问题的，编译器会为我们提示错误信息："UnKnown class 'E' "
55     * 虽然我们声明了<T>,也表明了这是一个可以处理泛型的类型的泛型方法。
56     * 但是只声明了泛型类型T，并未声明泛型类型E，因此编译器并不知道该如何处理E这个类型。
57     public <T> T showKeyName(Generic<E> container){
58        ...
59     }
60     */
61
62      /**
63     * 这个方法也是有问题的，编译器会为我们提示错误信息："UnKnown class 'T' "
64     * 对于编译器来说T这个类型并未项目中声明过，因此编译也不知道该如何编译这个类。
65     * 所以这也不是一个正确的泛型方法声明。
66     public void showkey(T genericObj){
67
68     }
69     */
70
71      public static void main(String[] args) {
72
73
74      }
75  }
```

4.6.2 类中的泛型方法

当然这并不是泛型方法的全部，泛型方法可以出现杂任何地方和任何场景中使用。但是有一种情况是非常特殊的，当类中时，我们再通过一个例子看一下

```
1   public class GenericFruit {
```

```
    class Fruit{
        @Override
        public String toString() {
            return "fruit";
        }
    }

    class Apple extends Fruit{
        @Override
        public String toString() {
            return "apple";
        }
    }

    class Person{
        @Override
        public String toString() {
            return "Person";
        }
    }

    class GenerateTest<T>{
        public void show_1(T t){
            System.out.println(t.toString());
        }

        //在泛型类中声明了一个泛型方法，使用泛型E，这种泛型E可以为任意类型。可以类型与T相同，也可以不同。
        //由于泛型方法在声明的时候会声明泛型<E>，因此即使在泛型类中并未声明泛型，编译器也能够正确识别泛型方
        public <E> void show_3(E t){
            System.out.println(t.toString());
        }

        //在泛型类中声明了一个泛型方法，使用泛型T，注意这个T是一种全新的类型，可以与泛型类中声明的T不是同一
        public <T> void show_2(T t){
            System.out.println(t.toString());
        }
    }

    public static void main(String[] args) {
        Apple apple = new Apple();
        Person person = new Person();

        GenerateTest<Fruit> generateTest = new GenerateTest<Fruit>();
        //apple是Fruit的子类，所以这里可以
        generateTest.show_1(apple);
        //编译器会报错，因为泛型类型实参指定的是Fruit，而传入的实参类是Person
        //generateTest.show_1(person);

        //使用这两个方法都可以成功
        generateTest.show_2(apple);
        generateTest.show_2(person);

        //使用这两个方法也都可以成功
        generateTest.show_3(apple);
        generateTest.show_3(person);
    }
}
```

4.6.3 泛型方法与可变参数

再看一个泛型方法和可变参数的例子：

```
public <T> void printMsg( T... args){
    for(T t : args){
        Log.d("泛型测试","t is " + t);
    }
}
```

```
printMsg("111",222,"aaaa","2323.4",55.55);
```

4.6.4 静态方法与泛型

静态方法有一种情况需要注意一下，那就是在类中的静态方法使用泛型：**静态方法无法访问类上定义的泛型；如果静**
据类型不确定的时候，必须要将泛型定义在方法上。

即：**如果静态方法要使用泛型的话，必须将静态方法也定义成泛型方法** 。

```
public class StaticGenerator<T> {
    ....
    ....
    /**
   * 如果在类中定义使用泛型的静态方法，需要添加额外的泛型声明（将这个方法定义成泛型方法）
   * 即使静态方法要使用泛型类中已经声明过的泛型也不可以。
   * 如：public static void show(T t){..},此时编译器会提示错误信息：
     "StaticGenerator cannot be refrenced from static context"
   */
    public static <T> void show(T t){

    }
}
```

4.6.5 泛型方法总结

泛型方法能使方法独立于类而产生变化，以下是一个基本的指导原则：

> 无论何时，如果你能做到，你就该尽量使用泛型方法。也就是说，如果使用泛型方法将整个类泛型化，那么就应该使用泛型方法。
> 的方法而已，无法访问泛型类型的参数。所以如果static方法要使用泛型能力，就必须使其成为泛型方法。

4.6 泛型上下边界

在使用泛型的时候，我们还可以为传入的泛型类型实参进行上下边界的限制，如：类型实参只准传入某种类型的父类

- 为泛型添加上边界，即传入的类型实参必须是指定类型的子类型

```
public void showKeyValue1(Generic<? extends Number> obj){
    Log.d("泛型测试","key value is " + obj.getKey());
}
```

```
Generic<String> generic1 = new Generic<String>("11111");
Generic<Integer> generic2 = new Generic<Integer>(2222);
Generic<Float> generic3 = new Generic<Float>(2.4f);
Generic<Double> generic4 = new Generic<Double>(2.56);

//这一行代码编译器会提示错误，因为String类型并不是Number类型的子类
//showKeyValue1(generic1);

```

```
showKeyValue1(generic2);
showKeyValue1(generic3);
showKeyValue1(generic4);
```

如果我们把泛型类的定义也改一下:

```
public class Generic<T extends Number>{
    private T key;

    public Generic(T key) {
        this.key = key;
    }

    public T getKey(){
        return key;
    }
}
```

```
//这一行代码也会报错，因为String不是Number的子类
Generic<String> generic1 = new Generic<String>("11111");
```

再来一个泛型方法的例子：

```
//在泛型方法中添加上下边界限制的时候，必须在权限声明与返回值之间的<T>上添加上下边界，即在泛型声明的时候添加
//public <T> T showKeyName(Generic<T extends Number> container)，编译器会报错："Unexpected bound"
public <T extends Number> T showKeyName(Generic<T> container){
    System.out.println("container key :" + container.getKey());
    T test = container.getKey();
    return test;
}
```

通过上面的两个例子可以看出：**泛型的上下边界添加，必须与泛型的声明在一起** 。

4.7 关于泛型数组要提一下

看到了很多文章中都会提起泛型数组，经过查看sun的说明文档，在java中是**"不能创建一个确切的泛型类型的数组"**的

也就是说下面的这个例子是不可以的：

```
List<String>[] ls = new ArrayList<String>[10];
```

而使用通配符创建泛型数组是可以的，如下面这个例子：

```
List<?>[] ls = new ArrayList<?>[10];
```

这样也是可以的：

```
List<String>[] ls = new ArrayList[10];
```

下面使用Sun的一篇文档的一个例子来说明这个问题：

```
List<String>[] lsa = new List<String>[10]; // Not really allowed.
Object o = lsa;
Object[] oa = (Object[]) o;
List<Integer> li = new ArrayList<Integer>();
```

```
5  li.add(new Integer(3));
6  oa[1] = li; // Unsound, but passes run time store check
7  String s = lsa[1].get(0); // Run-time error: ClassCastException
```

> 这种情况下，由于JVM泛型的擦除机制，在运行时JVM是不知道泛型信息的，所以可以给oa[1]赋上一个ArrayList而不会出现异常，时候却要做一次类型转换，所以就会出现ClassCastException，如果可以进行泛型数组的声明，上面说的这种情况在编译期将不会错误，只有在运行时才会出错。
> 而对泛型数组的声明进行限制，对于这样的情况，可以在编译期提示代码有类型安全问题，比没有任何提示要强很多。

下面采用通配符的方式是被允许的:**数组的类型不可以是类型变量，除非是采用通配符的方式**，因为对于通配符的方式要做显式的类型转换的。

```
1  List<?>[] lsa = new List<?>[10]; // OK, array of unbounded wildcard type.
2  Object o = lsa;
3  Object[] oa = (Object[]) o;
4  List<Integer> li = new ArrayList<Integer>();
5  li.add(new Integer(3));
6  oa[1] = li; // Correct.
7  Integer i = (Integer) lsa[1].get(0); // OK
```

5. 最后

本文中的例子主要是为了阐述泛型中的一些思想而简单举出的，并不一定有着实际的可用性。另外，一提到泛型，相信是在集合中，其实，在实际的编程过程中，自己可以使用泛型去简化开发，且能很好的保证代码质量。

泛型是什么？为什么要使用泛型?

我们在编写程序时，经常遇到两个模块的功能非常相似，只是一个是处理int型数据，另一个是处理String类型数据，或者其... 博文

Java泛型深入理解

Java泛型，包括Java泛型的实现，泛型擦除以及相关面试题，通配符理解 博文

泛型编程

1.C++两种抽象方法（1）面向对象编程封装（Encapsulation） 继承（Inheritance） 多态（Polymorphism）（2）泛型编程... 博文

泛型

集合可以存储任何类型的对象，但是当把一个对象存入集合后，集合会忘记这个对象的类型，将该对象从集合中取出时，这... 博文

什么是泛型?

高能预警！！！以下内容为一个学习java两个月的菜菜菜菜鸡的一点点见解，请谨慎阅读！！ 今天一个朋友突然问我泛... 博文

泛型中占位符T和?有什么区别?

泛型中占位符T和?有什么区别？这是一个好问题，有的人可能弄不清楚，所以我们这里简单的演示一下，相信大家一定能弄... 博文

Class T泛型和通配符泛型的区别

转载:https://www.cnblogs.com/skyislimit/p/5665853.html平时看java源代码的时候，如果碰到泛型的话，我想?TKVE这些是... 博文

【Java基础】泛型初级

引言泛型实现了参数化类型的概念，使代码可以应用于多种类型，通过解耦类或方法与所使用的类型之间的约束来实现。jav... 博文

Linux——linux学习全攻略 (转)

Linux——linux学习全攻略Linux——linux学习全攻略Linux——linux学习全攻略 博文

Linux 基础学习(笔记)

Linux发展史与安装一、Linux发展史1、Linux前身-Unix1968年 Multics项目MIT、Bell实验室、美国通用电气有限公司走到了... 博文

什么是Linux怎么学习?

Linux是什么?学了可以装逼?Linux简介Linux为何物Linux就是一个操作系统，就像你多少已经了解的Windows（xp，7，8）... 博文

Linux设备驱动开发入门

-

非常详细的linux学习资料-国内最全

为了让大家更好的更加方便的学习Linux，特意将积攒多年的linux学习资料奉上。内容包括linux学习的方方面面，Linux 学习路线

连续特征离散化和归一化

连续特征进行离散化处理。 博文

Java的组合（持有对象）与继承的区别

组合继承持有对象Adapter设计模式组合（持有对象）与继承的区别 博文

《C++0x漫谈》系列之：多线程内存模型

《C++0x漫谈》系列之：多线程内存模型 By刘未鹏(pongba)刘言|C++的罗浮宫(http://blog.csdn.net/pongba) 《C++0x漫谈… 博文

java中的泛型和反射的一些总结

什么叫反射？反射是框架设计的灵魂（使用的前提条件：必须先得到代表的字节码的Class，Class类用于表示.class文件（字… 博文

java中的方法返回值使用泛型，实现灵活的返回值类型

痛点： 使用Mybatis框架的时候，想封装一个底层JDBC控制器，用于提供和Mybatis交互的增删改查接口（公用的接口），… 博文

Java泛型类详细解读

什么是泛型？泛型（Genericstype或者generics）是对 Java 语言的类型系统的一种扩展，以支持创建可以按类型进行参数化… 博文

泛型方法的使用

一泛型方法的介绍如果定义类、接口是没有使用类型形参，但定义方法时想自己定义类型形参，这也是可以的，JDK1.5还提… 博文

java泛型（一）、泛型的基本介绍和使用

转载地址 http://m.blog.csdn.net/article/details?id=7864531现在开始深入学习java的泛型了，以前一直只是在集合中简单的使… 博文

一文搞懂Java泛型到底是什么东东

对java的泛型特性的了解仅限于表面的浅浅一层，直到在学习设计模式时发现有不了解的用法，才想起详细的记录一下。本… 博文

什么是泛型？为什么要使用泛型？

我们在编写程序时，经常遇到两个模块的功能非常相似，只是一个是处理int类型数据，另一个处理String类型数据，或者其… 博文

定义泛型

Java的一个聚集对象可以存储多种类型的数据，比如Set中可以存整数也可以存字符串。没有泛型之前有以下两个问题：从聚… 博文

Java高级系列——如何使用、何时使用泛型（Generics）？

一、介绍泛型的概念代表了对类型的抽象（C++开发人员熟知的模板）。它是一个非常强大的概念，它允许开发抽象算法和… 博文

Java中声明泛型方法

泛型是什么意思在这就不多说了，而Java中泛型类的定义也比较简单，例如：publicclassTest{}。这样就定义了一个泛型类Te… 博文

think in java 第十五章 泛型 总结随笔

1)基本概念： 泛型(GenericType或Generics)是对Java语言的类型系统的一种扩展，以支持创建可以按类型进行参数化的… 博文

java 泛型详解-绝对是对泛型方法讲解最详细的

对Java的泛型特性的了解仅限于表面的浅浅一层，直到在学习设计模式时发现有不了解的用法，才想起详细的记录一下。本… 博文

[疯狂Java]泛型：泛型方法、泛型方法VS类型通配符（?）

1.定义泛型方法： 1)如果你定义了一个泛型（类、接口），那么Java规定，你不能在所有的静态方法、静态初块等所有静态... 博文

【Dubbo】通过dubbo源码理解重新认识JAVA泛型及泛型方法

在com.alibaba.dubbo.rpc.protocol.dubbo.DubboProtocol类中使用到泛型方法，即public<T>Exporter<It... 博文

Java泛型详解—绝对是对泛型方法讲解最详细的，没有之一！

1.概述泛型在java中有很重要的地位，在面向对象编程及各种设计模式中有非常广泛的应用。什么是泛型？为什么要使用泛... 博文

java 泛型详解-绝对是对泛型方法讲解最详细的，没有之一（重点理解泛型方法和泛型通配符）

http://blog.csdn.net/s10461/article/details/53941091对java的泛型特性的了解仅限于表面的浅浅一层，直到在学习设计模式... 博文

Java泛型详解，通俗易懂的写法

我们知道，使用变量之前要定义，定义一个变量时必须要指明它的数据类型，什么样的数据类型赋给什么样的值。假如我们... 博文

泛型常用特点

（1）Java泛型开发人员在使用泛型的时候，很容易根据自己的直觉而犯一些错误。比如一个方法如果接收List<Objec... 博文

泛型<T> <E> <K,V> 里面的 T E K V 分别是什么类型?

泛型<T> <E> <K,V> 里面的 T E K V 分别是什么类型? 这样写有什么好处? 目前，我知道 <String> <Student> 等 这样写...

泛型，为什么使用T而不使用Object

如题，使用泛型时，通常用T，但T继承于Object,为什么不直接使用object。两者间有啥却别?

泛型超详细解读（一）：泛型基础

一：什么是泛型泛型是jdk5才引进的，泛型其实指得就是参数化类型，使得代码可以适应多种类型。像容器，List，大量使用... 博文

泛型 T 和 通配符？

最近被两者搞得有点晕，特意复习一下基础泛型的应用：可以理解为将类型指定为抽象类型（通俗点就是模糊指定类型，或... 博文

Linux入门基础教程

1.1 Linux操作系统简介Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务... 博文

Linux学习笔记--rm命令(删除文件或目录)

rm：英文名remove，删除的意思。1.命令格式：rm[选项]文件或目录2.常用选项："rm-f"强行删除，忽略不存在的文件，不提... 博文

在Linux下学习MySQL

1、升降序排序命令英文全称：2、MySQL在Linux上的Yum安装过程：3、为什么需要格式化磁盘？4、MySQL数据类型：5... 博文

Linux shell 学习

·详细介绍Linuxshell脚本基础学习（一）·详细介绍Linuxshell脚本基础学习（二）·详细介绍Linuxshell脚本基础学习（三）·... 博文

Linux--每天学习一个命令（一）

Linux系统监控Top命令Processes:355total,3running,352sleeping,1540threads22:53:50LoadAvg:2.23,1.85,1.79CPUus... 博文

转：java 泛型详解-绝对是对泛型方法讲解最详细的，没有之一

原文链接：https://www.cnblogs.com/coprince/p/8603492.html对java的泛型特性的了解仅限于表面的浅浅一层，直到在学习... 博文

【系列】重新认识Java——泛型（基础、使用和实现原理）

泛型是Java中重要知识点，是必须要深刻掌握的内容。由于泛型相关的内容比较多，基于单一知识原则，笔者打算将泛型部... 博文

CTF/CTF练习平台-flag在index里【php://filter的利用】

原题内容： http://120.24.86.145:8005/post/ Mark一下这道题，前前后后弄了两个多小时，翻了一下别的博主的wp感觉还是... 博文

关于树的几个ensemble模型的比较（GBDT、xgBoost、lightGBM、RF）

决策树的Boosting方法比较 原始的Boost算法是在算法开始的时候，为每一个样本赋上一个权重值，初始的时候，大家都是... 博文

【小程序】微信小程序开发实践

帐号相关流程注册范围 企业 政府 媒体 其他组织换句话讲就是不让个人开发者注册。 :)填写企业信息不能使用和之前的公众... 博文

DM368开发 -- 编码并实时播放

最近正好又用到 DM368 开发板，就将之前做的编解码的项目总结一下。话说一年多没碰，之前做的笔记全忘记是个什么鬼... 博文

【STM库应用】stm32 之 TIM （详解一 通用定时器）

STM32的TIM一般有高级定时器TIM1，(TIM8只有在互联性产品有)，普通定时器TIM2，TIM3，TIM4，(TIM5，TIM6，TIM7... 博文

servlet+jsp实现过滤器，防止用户未登录访问

我们可能经常会用到这一功能，比如有时，我们不希望用户没有进行登录访问后台的操作页面，而且这样的非法访问会让系... 博文

通俗理解条件熵

1 信息熵以及引出条件熵 我们首先知道信息熵是考虑该随机变量的所有可能取值，即所有可能发生事件所带来的信息量的期... 博文

将Excel文件导入数据库（POI+Excel+MySQL+jsp页面导入）第一次优化

本篇文章是根据我的上篇博客，给出的改进版，由于时间有限，仅做了一个简单的优化。相关文章：将excel导入数据库2018... 博文

jquery/js实现一个网页同时调用多个倒计时(最新的)

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本上都是千遍一律的不好用. ... 博文

ThreadLocal的设计理念与作用

Java中的ThreadLocal类允许我们创建只能被同一个线程读写的变量。因此，如果一段代码含有一个ThreadLocal变量的引用... 博文

配置简单功能强大的excel工具类搞定excel导入导出工具类(一)

对于J2EE项目导入导出Excel是最普通和实用功能,本工具类使用步骤简单,功能强大,只需要对实体类进行简单的注解就能实... 博文

【深入Java虚拟机】之五：多态性实现机制——静态分派与动态分派

Class文件的编译过程中不包含传统编译中的连接步骤，一切方法调用在Class文件里面存储的都只是符号引用，而不是方法... 博文

关于SpringBoot bean无法注入的问题（与文件包位置有关）

问题场景描述整个项目通过Maven构建，大致结构如下： 核心Spring框架一个module spring-boot-base service和dao一个m... 博文

非局部均值去噪（NL-means）

非局部均值（NL-means）是近年来提出的一项新型的去噪技术。该方法充分利用了图像中的冗余信息，在去噪的同时能最大… 博文

centos 查看命令源码

yum install yum-utils 设置源: [base-src] name=CentOS-5.4 - Base src - baseurl=http://vault.ce... 博文

expat介绍文档翻译

原文地址：http://www.xml.com/pub/a/1999/09/expat/index.html 因为需要用，所以才翻译了这个文档。但总归赖于英语水平… 博文

入门讲解 动态规划模型方法讲解 固定长度数组讲解 Solidity元组讲解 论文详解

server的安全控制模型是什么 sql ios获取idfa android title搜索 ios 动态修改约束 java 学习 泛型 java泛型学习