瞎扯淡：Java 泛型编程 - 泛型方法

作者：flhuoshan
原文链接：https://ld246.com/article/1489209528390
来源网站：链滴

概论

泛型方法可以定义在泛型类中，也可以定义在非泛型的普通类中，即泛型方法不依赖于泛型类的存在存在，而是可以单独定义和使用。有一个原则：能使用泛型方法完成的功能，就不要使用泛型类。泛方法表达一种：deal with T的概念。

泛型方法的一般格式：

```
public <T> void methodName(T t){
}
```

例如：

```
public class GenericMethodTest {
    public static <T> void dealWith( T t){
        System.out.println(t.getClass().getName());
    }
    public static void main(String[] args) {
        GenericMethodTest.dealWith("String");
        GenericMethodTest.dealWith(new Object());
        GenericMethodTest.dealWith(new BigDecimal(123));
    }
}
```

可变参数泛型方法

泛型方法和可变参数可以结合使用例如：

```
public class MultArgsTest {
    @SafeVarargs
    public static <T> List<T> makeList(T... args){
        List<T> result = new ArrayList<T>();
        for(T item : args){
            result.add(item);
        }
        return result;
    }
    public static void main(String[] args) {
        List<String> listOfStr = MultArgsTest.makeList("one","two","three","four");
        List<String> listOfStr2 = MultArgsTest.makeList("one","two","three","four","five","six");
        System.out.println(listOfStr);
        System.out.println(listOfStr2);

    }

}
```

具体实例

● 可以利用Java的泛型特性，编写一些实用的代码，比如生成器，它与反射技术结合，可以用于批量成对象(对象需要声明为public,确保BasicGenerator类具有访问权限，同时必须具备默认的构造器[一类都有])：

```
public class BasicGenerator<T> implements Generator<T> {

    private Class<T> type;

    public BasicGenerator(Class<T> type)     {
        this.type = type;
    }

    public T next() {
        try {
            return type.newInstance();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public static <T> Generator<T> create(Class<T> type) {
        return new BasicGenerator<T>(type);
    }

    public static void main(String[] args) {
        // 其他类型也可以
        String[] thousandStrs = new String[1000];
        Generator<String> gen = BasicGenerator.create(String.class);
        for (int i = 0; i < 1000; i++) {
            thousandStrs[i] = gen.next();
        }
    }

}

interface Generator<T> {
    public T next();

}
```

● 除了批量生成类之外，还可以结合集合类的特性，编写具有广泛用途的代码，比如，下面这个代码合了Set类，用来求得元素集合的交集并集等：

```
public class Sets {
    /**
     * 返回集合a与集合b的并集
     *
     * @param a
     * @param b
     * @return
     */
    public static <T> Set<T> union(Set<T> a, Set<T> b) {
        Set<T> result = new HashSet<T>(a);
        result.addAll(b);
        return result;
    }

    /**
```

```
     * 求集合a,b的交集并返回
     *
     * @param a
     * @param b
     * @return
     */
    public static <T> Set<T> intersection(Set<T> a, Set<T> b) {
        Set<T> result = new HashSet<T>(a);

        result.retainAll(b);
        return result;
    }

    /**
     * 移除superset中subse包含的元素并返回
     *
     * @param superset
     * @param subset
     * @return
     */
    public static <T> Set<T> difference(Set<T> superset, Set<T> subset) {
        Set<T> result = new HashSet<>(superset);
        result.removeAll(subset);
        return result;
    }

    /**
     * 返回a,b集合交集之外的元素
     *
     * @param a
     * @param b
     * @return
     */
    public static <T> Set<T> complement(Set<T> a, Set<T> b) {
        return difference(union(a, b), intersection(a, b));
    }

}
```

上述的代码可以结合任何实现了Set接口的类来进行上述的数学运算。不仅限于此，Java的泛型方法够大幅度的提高代码的通用性。