Java泛型

参考资料: http://blog.csdn.net/ysjian\_pingcx/article/details/40019183

## Java泛型

泛型的应用可以提高的代码的复用性，同时泛型提供了类型检查，减少了数据的类型转换，同时保证了类型安全。java泛型的本质是参数化类型（Parameterized Type）的应用，也就是说所操作的数据类型被指定为一个参数，在用到的时候在指定具体的类型。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口和泛型方法。

Java语言中的泛型只在程序源码中存在，在编译后的字节码文件中，就已经被替换为原来的原始类型（Raw Type）了，并且在相应的地方插入了强制转型代码，因此对于运行期的Java语言来说，ArrayList<int>与ArrayList<String>就是同一个类。所以说泛型技术实际上是Java语言的一颗语法糖，Java语言中的泛型实现方法称为类型擦除，基于这种方法实现的泛型被称为伪泛型。

## 泛型擦除

1. 对于泛型类型，编译器自动提供一个相应的原始类型，原始类型的名字就是删去类型参数后的泛型类型名字，然后擦除类型变量并替换为限定类型（无限定类型用Object替换），如果有多个限定，默认为取第一个限定进行替换；
2. 对于泛型表达式的翻译，编译器自动插入强制类型转换；
3. 泛型方法的翻译可能引起类型擦除与多态的冲突。

## 约束与局限性

Java不能实现真正的泛型，只能使用类型擦除来实现伪泛型，这样虽然不会有类型膨胀的问题，但是也引起了许多新的问题。所以，Sun对这些问题做出了许多限制：

1. 不能用基本类型实例化类型参数。由于primitive type不是Object的子类。
2. 运行时类型查询只适用于原始类型（raw type）。a instanceof Pair<T>与a instanceof Pair<String>有相同结果，T被忽略了。
3. 不能抛出也不能捕获泛型类实例。
4. 参数化类型的数组不合法。Pair<String>[]  myTest = new Pair<String>[10]; //错误
5. 不能实例化类型变量。new T(...);//错误
6. 泛型类的静态上下文中类型变量无效。不能在静态域或方法中引用类型变量。
7. 注意擦除后的冲突。

## 命名规范

为了与java关键字区别开来，java泛型参数只是使用一个大写字母来定义。各种常用泛型参数的意义如下：

* E — Element，常用在java Collection里，如：List<E>, Iterator<E>, Set<E>
* K, V — Key，Value，代表Map的键值对
* N — Number，数字
* T — Type，类型，如String，Integer等等
* S,U,V etc. - 2nd, 3rd, 4th 类型，和T的用法一样

## 通配符与上下界

**无限定通配符**      无限定通配符适用于一些非null判断等简单操作。

Java泛型现在支持两种带有通配符的上下界的表达方式：

1. ? extends T - 这里的?表示类型T的任意子类型，包括类型T本身。
2. ? super T - 这里的?表示类型T的任意父类型，包括类型T本身。

**使用规则：**

当你需要从一个数据结构中获取数据时(get), 那么就使用? extends T；如果你需要存储数据(put)到一个数据结构时，那么就使用? super T；如果你又想存储数据，又想获取数据，那么就不要使用通配符?，即直接使用具体泛型T。

## 类型系统

引入泛型之后的类型系统增加了两个维度：一个是类型参数自身的继承体系结构，另外一个是泛型类或接口自身的继承体系结构。第一个指的是对于 List<String>和List<Object>这样的情况，类型参数String是继承自Object的，但前者不是后者的子类型，两者没有什么联系。而第二种指的是 List接口继承自Collection接口。对于这个类型系统，有如下的一些规则：

* 相同类型参数的泛型类的关系取决于泛型类自身的继承体系结构。即List<String>是Collection<String> 的子类型，List<String>可以替换Collection<String>。这种情况也适用于带有上下界的类型声明。
* 当泛型类的类型声明中使用了通配符的时候， 其子类型可以在两个维度上分别展开。如对Collection<? extends Number>来说，其子类型可以在Collection这个维度上展开，即List<? extends Number>和Set<? extends Number>等；也可以在Number这个层次上展开，即Collection<Double>和 Collection<Integer>等。如此循环下去，ArrayList<Long>和 HashSet<Double>等也都算是Collection<? extends Number>的子类型。
* 如果泛型类中包含多个类型参数，则对于每个类型参数分别应用上面的规则。