**[Java的Generic Programming简介](http://notesth.iteye.com/blog/1938486)**

**1. 定义**: 在定义class,interface和method的时候，generics可以把class或者是interface当成一个类型参数。

定义一个generic class:

1. **public** **class** Pair<T>  {
2. **public** Pair() { first = **null**; second = **null**; }
3. **public** Pair(T first, T second) { **this**.first = first; **this**.second = second; }
4. **public** T getFirst() { **return** first; }
5. **public** T getSecond() { **return** second; }
6. **public** **void** setFirst(T newValue) { first = newValue; }
7. **public** **void** setSecond(T newValue) { second = newValue; }
8. **private** T first;
9. **private** T second;
11. **public** **static** **void** main(String[] args) {
12. Pair<Integer> integerPair = **new** Pair(0, 1);
13. Pair<String> stringPair = **new** Pair("thinking in java", "java core");
14. ...
15. }
16. }

定义generic method:

1. **class** ArrayAlg  {
2. **public** **static**  T getMiddle(T[] a)  {
3. **return** a[a.length / 2];
4. }
5. }

“T”的使用限制范围（Bounds of Type Variable）:

1. **public** **static** <T **extends** Comparable>  Pair minmax(T[] a)...

其中Comparable就是T的bounding type，实际就是T的一个继承关系中的限制条件了。

**2. 原理**   
在虚拟机中，所有的类或者是接口都必须有一个类型，那“T”的类型是在定义了generic type之后，虚拟机会自动提供一个raw type。 raw type不是一个具体的类或者是接口，而是定义generics的所有的T的bounding type的总称，按语法来讲，他不是个名词，而是个代词。   
比如最开始的例子，Pair<T>，其的raw type就是Object，而...<T extends Comparable>...的raw type就是Comparable。   
在虚拟机中，第一个Pair<T>例子的代码就会变成：

1. **public** **class** Pair  {
2. **public** Pair(Object first, Object second)
3. {
4. **this**.first = first;
5. **this**.second = second;
6. }
7. **public** Object getFirst() { **return** first; }
8. **public** Object getSecond() { **return** second; }
9. **public** **void** setFirst(Object newValue) { first = newValue; }
10. **public** **void** setSecond(Object newValue) { second = newValue; }
11. **private** Object first;
12. **private** Object second;
13. ...
14. }

generics在虚拟机里的过程是：

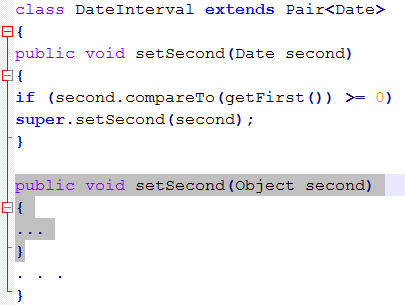
* 存，存成bounding type类型的
* 取，把bounding type类型的东西再还原回来

这个过程非常重要，很多地方都是拿这个过程解释的，当然，也有一部分冲突是由此引起的。   
下面就介绍一个由于此过程引起的冲突以及由此引出的一个概念，bridge method。   
看如下代码，注意这段代码是合法的：

1. **class** DateInterval **extends** Pair<Date>  {
2. **public** **void** setSecond(Date second)  {
3. **if** (second.compareTo(getFirst()) >= 0)
4. **super**.setSecond(second);
5. }
6. . . .
7. }

因为DateInterval是继承Pair<Date>，自然继承了Pair<Date>的setSecond方法，但是由于在虚拟机中Pair<Date>的setSecond方法被存为

1. **public** **void** setSecond(Object second)  {
2. ...
3. }

即相当于在DateInterval中有两个名字相同，参数不同（但是注意参数有继承关系）的方法，如果要调用setSecond就不知所措了，下图阴影部分为继承Pair<Date>所得，仅为示例之用。   


这时候编译器会提供一个bridge method，来解决两个方法冲突的问题。

1. **public** **void** setSecond(Object second) { setSecond((Date) second); }

这样问题就得到解决了，注意，这里是编译器的事儿，与写代码无关。   
在原理部分需要注意的几点是

* 实际上是没有generics存在的，也就是上面的“T”
* 所有的type parameters都会换成他们的bounding type
* bridge method会解决冲突的问题
* 在虚拟机里面会执行类型转换以确保安全

**3. 通配符（wildcard types）**   
也就是在使用generics的时候嫌麻烦，可以使得generic type中的type parameter可以在继承关系上使用更灵活点，或者是可以使用type parameter的父类，或者是可以使用其的子类。   
比如在方法：

1. **public** **static** **void** printBuddies(Pair<Employee> p)
2. {
3. Employee first = p.getFirst();
4. Employee second = p.getSecond();
5. System.out.println(first.getName() + " and " + second.getName() + " are buddies.";
6. }

其中要想给printBuddies传递一个Pair<Manager>类型的参数是不合法的，于是乎产生了一种新的定义方法：

1. **public** **static** **void** printBuddies(Pair<? **extends** Employee> p)

这样，就可以把Pair<Manager>类型的参数传入方法了。同理还可以定义代码：

1. **public** **static** **void** printBuddies(Pair<? **super** Manager> p)

这样的意思就变成所有Manager的父类都可以调用此段代码了。   
最后也可以就通配符“？”一个，什么继承关系都没有，比如：

1. **public** **static** **void** printBuddies(Pair<?> p)

具体通配符的用法可以参考<http://docs.oracle.com/javase/tutorial/extra/generics/wildcards.html>。