通配符有三种：

1、无限定通配符   形式<?>

2、上边界限定通配符 形式< ? extends Number>    //用Number举例

3、下边界限定通配符    形式< ? super Number>    //用Number举例

**1、泛型中的？通配符**

如果定义一个方法，该方法用于打印出任意参数化类型的集合中的所有数据，如果这样写

**[java]** [view plain](http://blog.csdn.net/LonelyRoamer/article/details/7927212) [copy](http://blog.csdn.net/LonelyRoamer/article/details/7927212)

1. **import** java.util.ArrayList;
2. **import** java.util.Collection;
3. **import** java.util.List;
4. publicclass GernericTest {
5. publicstaticvoid main(String[] args) **throws** Exception{
6. List<Integer> listInteger =**new** ArrayList<Integer>();
7. List<String> listString =**new** ArrayList<String>();
8. printCollection(listInteger);
9. printCollection(listString);
10. }
11. publicstaticvoid printCollection(Collection<Object> collection){
12. **for**(Object obj:collection){
13. System.out.println(obj);
14. }
15. }
16. }

语句printCollection(listInteger);报错

The method printCollection(Collection<Object>) in the type GernericTest is not applicable for the arguments (List<Integer>)

这是因为泛型的参数是不考虑继承关系就直接报错。

这就得用？通配符

**[java]** [view plain](http://blog.csdn.net/LonelyRoamer/article/details/7927212) [copy](http://blog.csdn.net/LonelyRoamer/article/details/7927212)

1. **import** java.util.ArrayList;
2. **import** java.util.Collection;
3. **import** java.util.List;
5. publicclass GernericTest {
6. publicstaticvoid main(String[] args) **throws** Exception{
7. List<Integer> listInteger =**new** ArrayList<Integer>();
8. List<String> listString =**new** ArrayList<String>();
9. printCollection(listInteger);
10. printCollection(listString);
11. }
12. publicstaticvoid printCollection(Collection<?> collection){
13. **for**(Object obj:collection){
14. System.out.println(obj);
15. }
16. }
17. }

在方法public static void printCollection(Collection<?> collection){}中不能出现与参数类型有关的方法比如collection.add();因为程序调用这个方法的时候传入的参数不知道是什么类型的，但是可以调用与参数类型无关的方法比如collection.size();

总结：使用?通配符可以引用其他各种参数化的类型,?通配符定义的变量的主要用作引用，可以调用与参数化无关的方法，不能调用与参数化有关的方法。

**2、泛型中的?通配符的扩展**

1:界定通配符的上边界

Vector<? extends 类型1> x = new Vector<类型2>();

类型1指定一个数据类型，那么类型2就只能是类型1或者是类型1的子类

Vector<? extends Number> x = new Vector<Integer>();//这是正确的

Vector<? extends Number> x = new Vector<String>();//这是错误的

2:界定通配符的下边界

Vector<? super 类型1> x = new Vector<类型2>();

类型1指定一个数据类型，那么类型2就只能是类型1或者是类型1的父类

Vector<? super Integer> x = new Vector<Number>();//这是正确的

Vector<? super Integer> x = new Vector<Byte>();//这是错误的

提示：限定通配符总是包括自己