泛型（Generics）

@M了个J

李明杰

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

小码哥教育 SEEMYGO

实力IT教育 www.520it.com

泛型（Generics）

■ 从 Java 5 开始，增加了泛型技术

■ 什么是泛型?

□ 将类型变为参数，提高代码复用率

■ 建议的类型参数名称

□ T ： Type

□ E ： Element

□ K ： Key

□ N ： Number

□ V ： Value

□ S、U、V ： 2nd, 3rd, 4th types

泛型类型（Generic Type）

- ■ 什么是泛型类型?
- ☐ 使用了泛型的类或者接口
- ☐ 比如
- ✓ java.util.Comparator
- ✓ java.util.Comparable

```java
public class Student<T> {
    private T score;
    public T getScore() {
        return score;
    }
    public void setScore(T score) {
        this.score = score;
    }
}
```

```java
// Java 7以前的写法
// Student<String> stu1 = new Student<String>();

// 从Java 7开始，可以省略右边<>中的类型
Student<String> stu1 = new Student<>();
stu1.setScore("A");
String score1 = stu1.getScore();

Student<Double> stu2 = new Student<>();
stu2.setScore(98.5);
Double score2 = stu2.getScore();
```

多个类型参数

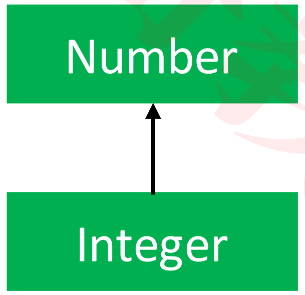
```java
public class Student<N, S> {
    private N no;
    private S score;
    public Student(N no, S score) {
        this.no = no;
        this.score = score;
    }
}
```

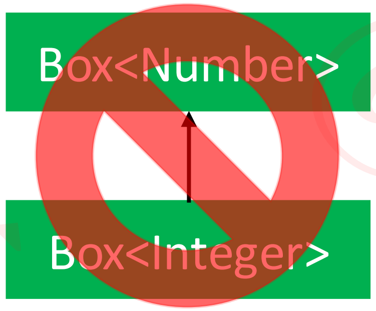
```java
Student<String, String> s1 = new Student<>("E9527", "A++");

Student<Integer, Double> s2 = new Student<>(18, 96.5);
```

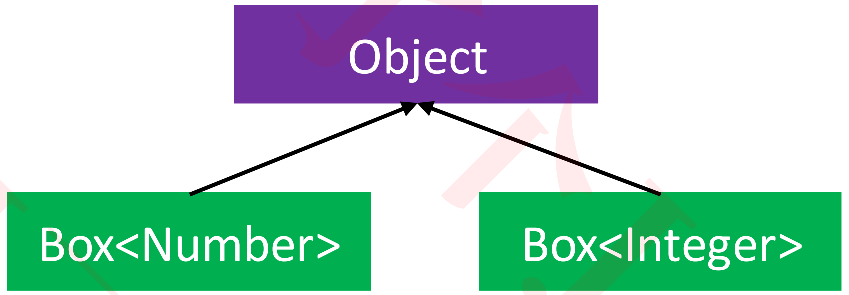
泛型类型的继承

```java
public class Box<E> {
    private E element;
    public E getElement() {
        return element;
    }
    public void setElement(E element) {
        this.element = element;
    }
}
```

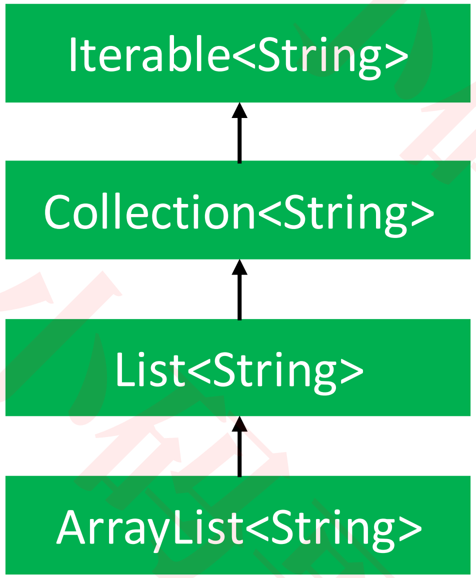
```java
Box<String> strBox = new Box<>();
Box<Integer> intBox = new Box<>();
Box<Object> objBox = new Box<>();
strBox = intBox; // error
objBox = strBox; // error

// 如果上面代码正确的话，请思考下面代码
objBox.setElement(new Object());
// 将Object直接转成String?
String str = strBox.getElement();
```





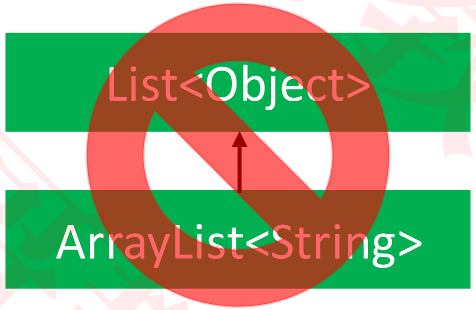


泛型类型的继承



```java
public interface Collection<E> extends Iterable<E>
public interface List<E> extends Collection<E>
public class ArrayList<E> implements List<E>
```

```java
Iterable<String> it = null;
Collection<String> col = null;
List<String> li = null;
ArrayList<String> al = null;

it = col;
col = li;
li = al;
```

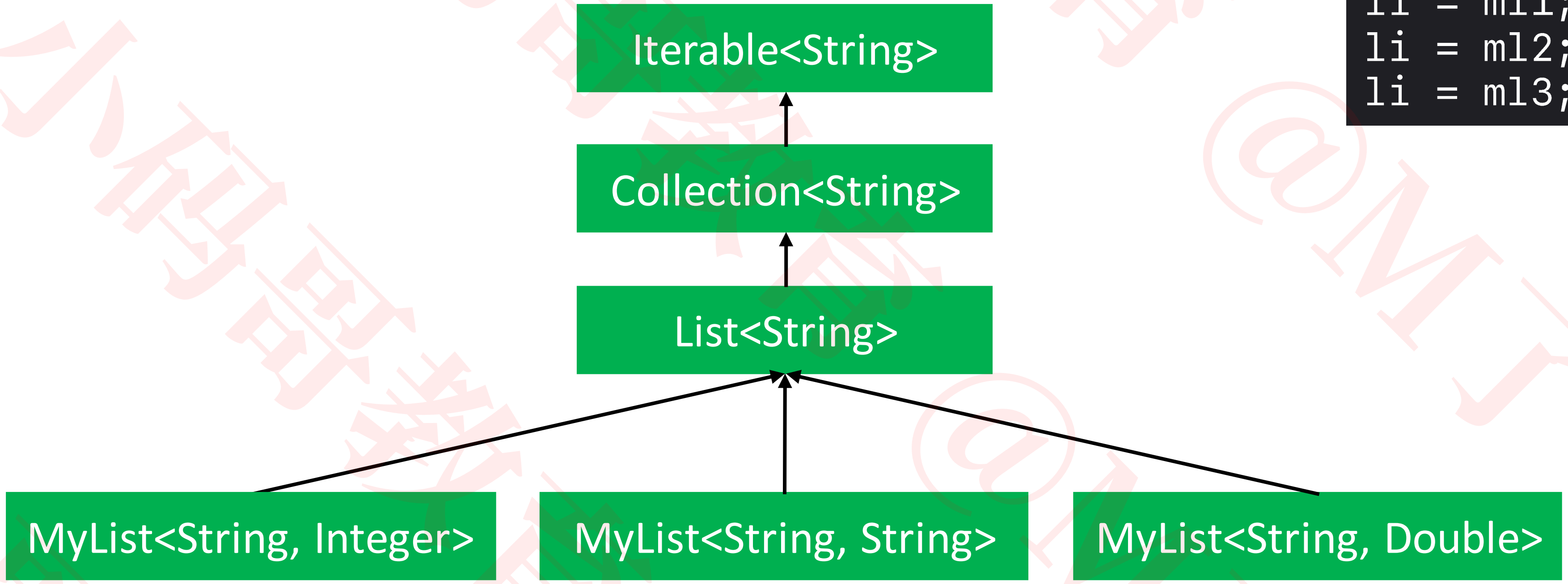


```java
List<Object> list = null;
ArrayList<String> al = null;

list = al; // error
```

泛型类型的继承

```java
public interface MyList<E, T> extends List<E> {
    void setNo(T no);
}
```

```java
List<String> li = null;
MyList<String, Integer> ml1 = null;
MyList<String, Double> ml2 = null;
MyList<String, String> ml3 = null;

li = ml1;
li = ml2;
li = ml3;
```



原始类型（Raw Type）

■ 什么是原始类型?

□ 没有传递具体的类型给泛型的类型参数

```
// Box称为是Box<E>的原始类型（Raw Type）
Box rawBox = new Box(); // warning: rawtypes
Box<String> strBox = new Box<>();
Box<Object> objBox = new Box<>();

rawBox = strBox; // ok
rawBox = objBox; // ok
strBox = rawBox; // warning: unchecked
objBox = rawBox; // warning: unchecked
```

■ 当使用了原始类型时，编译器会给出 rawtypes 警告（可以用 @SuppressWarnings 消除）

■ 将非原始类型赋值给原始类型时，编译器没有任何警告和错误

■ 将原始类型赋值给非原始类型时，编译器会给出 unchecked 警告（可以用 @SuppressWarnings 消除）

■ Box 是原始类型，Box<Object> 是非原始类型

泛型方法（Generic Method）

■ 什么是泛型方法?

□ 使用了泛型的方法（实例方法、静态方法、构造方法），比如 Arrays.sort(T[], Comparator<T>)

```java
public static void main(String[] args) {
    Student<String, String> s1 = new Student<>();
    Main.<String, String>set(s1, "K99", "C++");

    Student<Integer, Double> s2 = new Student<>();
    // 编译器可以自动推断出类型参数的具体类型
    set(s2, 25, 99.5);
}
static <T1, T2> void set(Student<T1, T2> stu, T1 no, T2 score) {
    stu.setNo(no);
    stu.setScore(score);
}
```

小码哥教育 SEEMYGO

泛型方法 – 示例

```java
public class Box<E> {
    private E element;
    public Box() {}
    public Box(E element) {
        this.element = element;
    }
}
```

```java
<T> void addBox(T element, List<Box<T>> boxes) {
    Box<T> box = new Box<>(element);
    boxes.add(box);
}
```

```java
List<Box<Integer>> boxes = new ArrayList<>();
addBox(11, boxes);
addBox(22, boxes);
addBox(33, boxes);
```

泛型方法 – 类型推断

```java
public class Collections {
    @SuppressWarnings("unchecked")
    public static final <T> List<T> emptyList() {
        return (List<T>) EMPTY_LIST;
    }
}
```

```java
List<String> list1 = Collections.emptyList();
List<Integer> list2 = Collections.emptyList();
```

小码哥教育 SEEMYGO

泛型方法 – 构造方法

```java
public class Person<T> {
    private T age;
    public <E> Person(E name, T age) {

    }
}
```

```java
Person<Integer> p1 = new Person<>("Jack", 20);
Person<Double> p2 = new Person<>(666, 20.6);
Person<String> p3 = new Person<>(12.34, "80后");
```

限制类型参数

- 可以通过 extends 对类型参数增加一些限制条件，比如 <T extends A>
  - extends 后面可以跟上类名、接口名，代表 T 必须是 A 类型，或者继承、实现 A

```java
public class Person<T extends Number> {
    private T age;
    public Person(T age) {
        this.age = age;
    }
    public int getAge() {
        return (age == null) ? 0 : age.intValue();
    }
}
```

```java
Person<Double> p1 = new Person<>(18.7);
System.out.println(p1.getAge()); // 18
Person<Integer> p2; // OK
Person<String> p3; // Error
```

- 可以同时添加多个限制，比如 <T extends A & B & C>，代表 T 必须同时满足 A、B、C

小码哥教育 SEEMYGO

限制类型参数 – 示例

```java
<T extends Comparable<T>> T getMax(T[] array) {
    if (array == null || array.length == 0) return null;
    T max = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] == null) continue;
        if (array[i].compareTo(max) <= 0) continue;
        max = array[i];
    }
    return max;
}
```

```java
Double[] ds = { 5.6, 3.4, 8.8, 4.6 };
System.out.println(getMax(ds)); // 8.8

Integer[] is = { 4, 19, 3, 28, 56 };
System.out.println(getMax(is)); // 56
```

限制类型参数 – 示例

```java
public class Student<T extends Comparable<T>> implements Comparable<Student<T>> {
    private T score;
    public Student(T score) {
        this.score = score;
    }
    @Override
    public int compareTo(Student s) {
        if (s == null) return 1;
        if (score != null && s.score != null) return score.compareTo(s.score);
        if (score == null && s.score == null) return 0;
        return s.score == null ? 1 : -1;
    }
    @Override
    public String toString() {
        return "[score=" + score + "]";
    }
}
```

```java
Student<Integer>[] stus = new Student[3];
stus[0] = new Student<>(18);
stus[1] = new Student<>(38);
stus[2] = new Student<>(28);
// [score=38]
System.out.println(getMax(stus));
```

通配符（Wildcards）

- 在泛型中，问号（?）被称为是通配符

- 通常用作变量类型、返回值类型的类型参数

- 不能用作泛型方法调用、泛型类型实例化、泛型类型定义的类型参数

通配符 – 上界

■ 可以通过 extends 设置类型参数的上界

```
// 类型参数必须是Number类型或者是Number的子类型
void testUpper(Box<? extends Number> box) {}

Box<Integer> p1 = null;
Box<Number> p2 = null;
Box<? extends Number> p3 = null;
Box<? extends Integer> p4 = null;
testUpper(p1);
testUpper(p2);
testUpper(p3);
testUpper(p4);
```

通配符 – 上界 – 示例

```
double sum(List<? extends Number> list) {
    double s = 0.0;
    for (Number n : list) {
        s += n.doubleValue();
    }
    return s;
}
```

```
List<Integer> is = Arrays.asList(1, 2, 3);
// 6.0
System.out.println(sum(is));

List<Double> ds = Arrays.asList(1.2, 2.3, 3.5);
// 7.0
System.out.println(sum(ds));
```

通配符 – 下界

- 可以通过 super 设置类型参数的下界

```
// 类型参数必须是Integer类型或者是Integer的父类型
void testLower(Box<? super Integer> box) {}

Box<Integer> p1 = null;
Box<Number> p2 = null;
Box<? super Integer> p3 = null;
Box<? super Number> p4 = null;
testLower(p1);
testLower(p2);
testLower(p3);
testLower(p4);
```

小码哥教育 SEEMYGO

通配符 – 下界 – 示例

```java
void addNumbers(List<? super Integer> list) {
    for (int i = 1; i <= 10; i++) {
        list.add(i);
    }
}
```

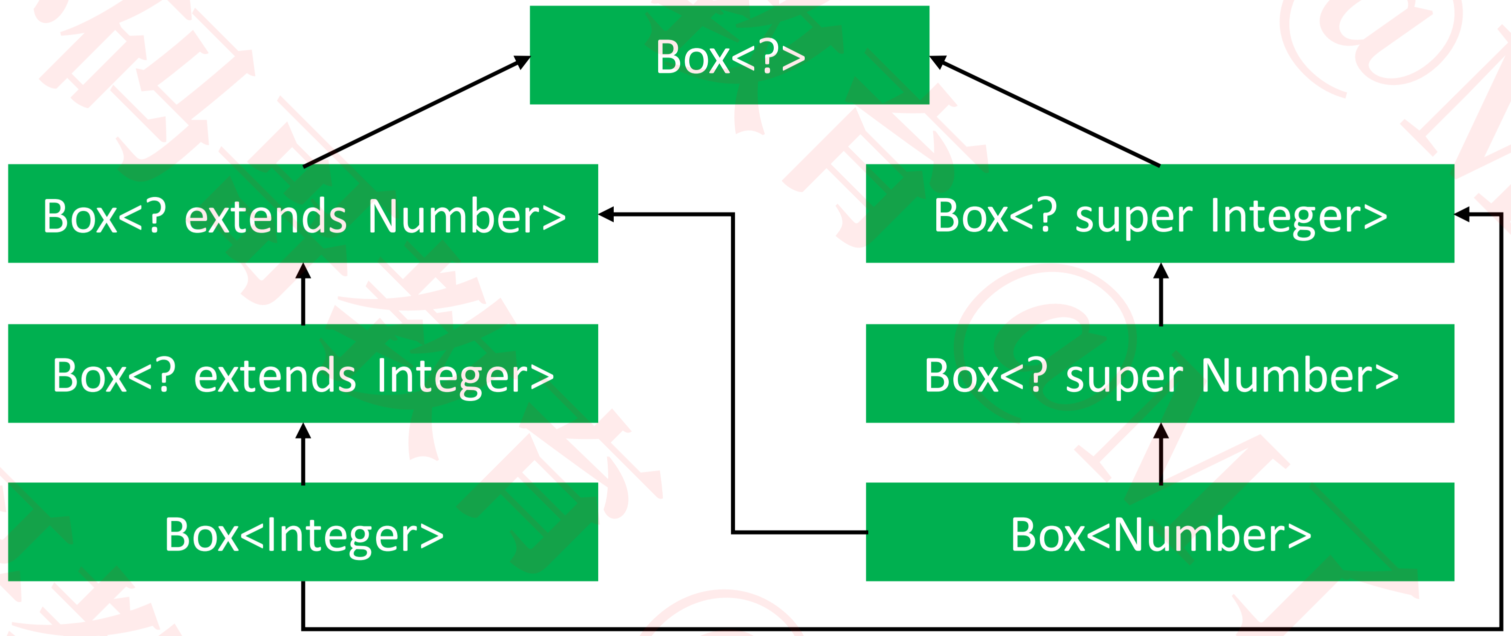
```java
List<Integer> is = new ArrayList<>();
addNumbers(is);
// [1, 2, 3, 4, 5]
System.out.println(is);

List<Number> ns = new ArrayList<>();
addNumbers(ns);
// [1, 2, 3, 4, 5]
System.out.println(ns);
```

通配符 – 无限制

```java
// 类型参数是什么类型都可以
void test(Box<?> box) {}

Box<Integer> p1 = null;
Box<String> p2 = null;
Box<Object> p3 = null;
Box<? extends Number> p4 = null;
Box<? super String> p5 = null;
Box<?> p6 = null;
test(p1);
test(p2);
test(p3);
test(p4);
test(p5);
test(p6);
```

```java
void printList(List<?> list) {
    for (Object obj: list) {
        System.out.print(obj + " ");
    }
    System.out.println();
}
List<Integer> is = Arrays.asList(1, 2, 3);
// 1 2 3
printList(is);

List<Double> ds = Arrays.asList(1.2, 2.3, 3.5);
// 1.2 2.3 3.5
printList(ds);
```

通配符 – 继承



通配符 – 注意

■ 编译器在解析 List<E>.set(int index, E element) 时，无法确定 E 的真实类型，所以报错

```
void foo(List<?> list) {
    Object obj = list.get(0); // ok
    list.set(0, obj); // error
    list.set(0, list.get(0)); // error
}
```

```
void foo(List<?> list) {
    fooHelper(list); // ok
}

<T> void fooHelper(List<T> l) {
    l.set(0, l.get(0));
}
```

```
void swapFirst(List<? extends Number> l1, List<? extends Number> l2) {
    Number temp = l1.get(0); // ok
    l1.set(0, l2.get(0)); // error
    l2.set(0, temp); // error
}
```

泛型的使用限制

- 基本类型不能作为类型参数

```java
// error
Map<int, char> map1 = new HashMap<>();
// ok
Map<Integer, Character> map2 = new HashMap<>();
```

- 不能创建类型参数的实例

```java
public class Box<E> {
    public void add(Class<E> cls) throws Exception {
        // error
        E e1 = new E();

        // ok
        E e2 = cls.newInstance();
    }
}
```

小码哥教育 SEEMYGO

泛型的使用限制

■ 不能用类型参数定义静态变量

```
public class Box<E> {
    // error
    private static E value;
}
Box<Integer> box1 = new Box<>();
Box<String> box2 = new Box<>();
// 请问静态变量value是什么类型?Integer还是String?
```

■ 泛型类型的类型参数不能用在静态方法上

```
public class Box<E> {
    // error
    public static void show(E value) {}
}
```

泛型的使用限制

- 类型参数不能跟 instanceof 一起使用

```java
ArrayList<Integer> list = new ArrayList<>();
// error
if (list instanceof ArrayList<Integer>) {

}
```

- 不能创建带有类型参数的数组

```java
// error
Box<Integer>[] boxes1 = new Box<Integer>[4];

// ok
Box<Integer>[] boxes2 = new Box[4];
```

泛型的使用限制

- 下面的方法不属于重载

```
// error
void test(Box<Integer> box) {

}

void test(Box<String> box) {

}
```

```
// error
void foo(Box<? extends Number> box) {

}

void foo(Box<String> box) {

}
```

泛型的使用限制

- 不能定义泛型的异常类

```java
// error
public class MyException<T> extends Exception {

}
```

- catch 的异常类型不能用类型参数

```java
public static <T extends Exception> void test(Box<T> box) {
    try {

    // error
    } catch (T e) {}
}
```

泛型的使用限制

- 下面的代码是正确的

```
class Parser<T extends Exception> {
    // ok
    public void parse(File file) throws T {

    }
}
```