泛型

TYIC 桃高資訊社

泛型

泛型(generic)是在不知道類別時使用代號來代替類別

在編譯時編譯器會把代號擦除，等執行時才會把真正的類別補上

泛型代號通常使用一個大寫英文字母，如 T、R、E、K、V

在原本可以填類別的地方都可以填上泛型

但需特別注意，不可以建立泛型或泛型陣列的新實例：

```java
new T();
new T[5];
// java: unexpected type
//   required: class
//   found:    type parameter T
```

泛型方法

若要在方法中使用泛型，格式如下：

```
修飾子 <泛型代號1, 泛型代號2, ..., 泛型代號n> 返回值型別 方法名稱(...) {...}
```
java

呼叫靜態泛型方法：

```
類別.靜態方法名稱<泛型代號1, 泛型代號2, ..., 泛型代號n>(...)
```
java

呼叫動態泛型方法：

```
物件.動態方法名稱<泛型代號1, 泛型代號2, ..., 泛型代號n>(...)
```
java

```java
public class Main {
    public static void main(String[] args) {
        Integer[] integerArr =
                new Integer[]{1, 2, 3, 4, 5};
        System.out.println(Main.<Integer>lastElement(integerArr));
    }

    public static <T> T lastElement(T[] arr) {
        return arr[arr.length - 1];
    }
}
```
java

```
5
```
output

泛型類別

若要在類別或介面中使用泛型，則須使用以下格式：

```java
修飾子 類別名稱<泛型代號1, 泛型代號2, ..., 泛型代號n> {...}
```

需要注意，泛型類別的靜態欄位不可以使用泛型

創建泛型類別實例：

```java
new 類別名稱<泛型代號1, 泛型代號2, ..., 泛型代號n>(...)
```

存取泛型類別成員：

```java
類別名稱<泛型代號1, 泛型代號2, ..., 泛型代號n>.成員
```

若泛型方法或泛型介面中的泛型的實際型別未確定

則泛型的實際型別為 Object

泛型類別

```java
public class Main {
    public static void main(String[] args) {
        Pair<String, String> pair1 =
                new Pair<String, String>("羅志翔", "小朱");
        pair1.set("羅至祥", "小諸");
        System.out.println(pair1);
        var pair2 =
                new Pair<String, String>("梁靜如", "晴歌天后");
        System.out.println(pair2.getLeft());
        var pair3 = new Pair<>("王新凌", "甜欣教主");
        System.out.println(pair3.getRight());
    }
}
```

```
羅至祥：小諸
梁靜如
甜欣教主
```
output

```java
class Pair<L, R> {
    private L left;
    private R right;

    public Pair(L left, R right) {
        set(left, right);
    }

    public void set(L left, R right) {
        this.left = left;
        this.right = right;
    }

    public L getLeft() {
        return left;
    }

    public R getRight() {
        return right;
    }

    @Override
    public String toString() {
        return left.toString() + ':' + right.toString();
    }
}
```
java

泛型介面

與泛型類別
用法相同

```java
import java.util.Arrays;

// 25_interface_06 改寫
public class Main {
    public static void main(String[] args) {
        Integer[] intArr = {1, 2, 3, 4, 5, 6, 7};
        Num[] numArr = new Num[7];
        ArrayHelper.map(intArr, Num::new, numArr);
        System.out.println(Arrays.toString(numArr));
        ArrayHelper.map(numArr, Num::square, numArr);
        System.out.println(Arrays.toString(numArr));
        ArrayHelper.map(numArr, Num::negative, numArr);
        System.out.println(Arrays.toString(numArr));
    }
}

abstract class ArrayHelper {
    @FunctionalInterface
    public interface Mapper<E, R> {
        R map(E value);
    }

    public static <E, R> void map(E[] srcArray, Mapper<E, R> mapFunction, R[] dstArray) {
        for (int i = 0; i < Math.max(srcArray.length, dstArray.length); i++) {
            dstArray[i] = mapFunction.map(srcArray[i]);
        }
    }
}

class Num {
    int number;

    public Num(int number) {
        this.number = number;
    }

    public Num square() {
        return new Num(number * number);
    }

    public static Num negative(Num num) {
        return new Num(-num.number);
    }

    @Override
    public String toString() {
        return String.valueOf(number);
    }
}
```

java

限定泛型

若想要指定泛型型別的繼承關係，可以使用以下格式：

```
修飾子 類別名稱<泛型代號1 extends 父類別1, ..., 泛型代號n extends 父類別n> {
    ...
}
```
java

```
修飾子 <泛型代號1 extends 父類別1, ..., 泛型代號n extends 父類別1> 返回值型別 方法名稱(...) {
    ...
}
```
java

```java
public class Main {
    public static void main(String[] args) {
        Integer[] intArr = {1, 2, 3, 4, 5, 6, 7};
        ArrayHelper.map(intArr, (value) -> value / 2);
        System.out.println(Arrays.toString(intArr));
    }
}
```

```
[0, 1, 1, 2, 2, 3, 3]
```
output

```java
abstract class ArrayHelper {
    @FunctionalInterface
    public interface Mapper<E, R extends E> {
        R map(E value);
    }

    public static <E, R extends E> void map(
            E[] srcArray,
            Mapper<E, R> mapFunction) {
        for (int i = 0; i < srcArray.length; i++) {
            srcArray[i] = mapFunction.map(srcArray[i]);
        }
    }
}
```
java

限定泛型

若想要指定泛型型別的實作關係，可以使用以下格式：

若同時限定繼承類別和實作介面，須將類別放在介面前方

```
修飾子 類別名稱<泛型代號1 extends 實作介面1 & 實作介面2 & ... & 實作介面n, ...> {...}
```
java

```
修飾子 <泛型代號1 extends 實作介面1 & 實作介面2 & ... & 實作介面n, ...> 返回值型別 方法名稱(...) {
    ...
}
```
java

```
public class Main {
    public static void main(String[] args) {
        swimThenFly(new FlyingFish());
    }

    public static <T extends Fish & CanFly> void swimThenFly(T canFlyFish) {
        canFlyFish.swim();
        canFlyFish.fly();
    }
}
class Fish {
    public void swim() {
        System.out.println("魚兒魚兒水中遊");
    }
}
```

```
interface CanFly {
    void fly();
}
```

```
魚兒魚兒水中遊
飛魚用魚翅飛
```
output

```
class FlyingFish extends Fish implements CanFly {
    @Override
    public void fly() {
        System.out.println("飛魚用魚翅飛");
    }
}
```
java

協變

我們會很直觀的認為

`Class<T1>` 是 `Class<T2>`(其中 `T2` 繼承 `T1`) 的父型別

這稱為協變性(covariant)，然而事實上並非如此

Java 的泛型不具有協變性，而是不變性(invariant)

所以泛型的繼承關係會被無視

若想讓泛型具有協變性，須使用型別通配字元 "?" 與 `extends`

型別通配字元只能在不需要知道實際型別的泛型類別才能使用

若沒有 `extends`，編譯器會自動補上 `extends Object`

型別通配字元只能單一上界或下界，不可限定泛型多個類別或介面

```java
泛型類別名稱<? extends 類別或介面, ..., ? extends 類別或介面>
```

```java
泛型類別名稱<?, ..., ?>
```

協變

在右方的程式中
第一行會編譯失敗
而第二行則不會
因為第二行使用了
通配型別字元
讓泛型具協變性

```
public class Main {
    public static void main(String[] args) {
        Pair<Integer, Aircraft> pair1 =
                new Pair<Integer, Helicopter>(1, new Helicopter());
        // incompatible types: Pair<java.lang.Integer,Helicopter>
        // cannot be converted to Pair<java.lang.Integer,Aircraft>
        Pair<Integer, ? extends Aircraft> pair2 =
                new Pair<Integer, Helicopter>(2, new Helicopter());
    }
}
```

```
class Pair<L, R> {
    public L left;
    public R right;

    public Pair(L left, R right) {
        this.left = left;
        this.right = right;
    }
}
```

```
abstract class Aircraft {
}
```

```
class Helicopter extends Aircraft {
}
```

逆變

若想要反過來讓 Class<T1> 成為 Class<T2>(其中 T2 繼承 T1)的子型別，即顛倒繼承或實作關係

須使用型別通配字元 "?" 與 super 讓泛型具有逆變性(contravariant)

逆變主要用於保證能對該類別實例操作

```
泛型類別名稱<? super 類別或介面, ...>
```
java

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        Group<Integer> integerGroup1 =
                new Group<>(new Integer[]{1, 2, 3, 4, 5, 6, 7});
        Group<Object> integerGroup2 = new Group<>(new Object[7]);
        integerGroup1.map(integerGroup2, (value) -> value * 8);
        System.out.println(integerGroup2);
    }
}
```

```
[8, 16, 24, 32, 40, 48, 56]
```
output

```java
class Group<E> {
    @FunctionalInterface
    public interface Mapper<E, R extends E> {
        R map(E value);
    }

    private final E[] data;

    public final int length;

    public Group(E[] data) {
        this.data = data;
        this.length = data.length;
    }

    public void setElement(E element, int index) {
        data[index] = element;
    }

    public E getElement(int index) {
        return data[index];
    }

    @Override
    public String toString() {
        return Arrays.toString(data);
    }

    public void map(
            Group<? super E> dstGroup,
            Mapper<E, ? extends E> mapFunction
    ) {
        for (int i = 0; i < data.length; i++) {
            dstGroup.setElement(mapFunction.map(getElement(i)), i);
        }
    }
}
```
java

生產者與消費者

生產者(producer)是指提供物件的型別

消費者(consumer)是指接收物件的型別

使用泛型類別時，遵守 PECS 原則：

"*Producer extends, Consumer super*"

也就是對於生產者時使用協變

對於消費者時使用逆變

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        Group<Integer> integerGroup1 =
                new Group<>(new Integer[]{1, 2, 3, 4, 5, 6, 7});
        Group<Object> integerGroup2 = new Group<>(new Object[7]);
        integerGroup1.map(integerGroup2, (value) -> value * 8);
        System.out.println(integerGroup2);
    }
}
```

```
[8, 16, 24, 32, 40, 48, 56]
```
output

```java
class Group<E> {
    @FunctionalInterface
    public interface Mapper<E, R extends E> {
        R map(E value);
    }

    private final E[] data;

    public final int length;

    public Group(E[] data) {
        this.data = data;
        this.length = data.length;
    }

    public void setElement(E element, int index) {
        data[index] = element;
    }

    public E getElement(int index) {
        return data[index];
    }

    @Override
    public String toString() {
        return Arrays.toString(data);
    }

    public void map(
            Group<? super E> dstGroup,
            Mapper<E, ? extends E> mapFunction
    ) {
        for (int i = 0; i < data.length; i++) {
            dstGroup.setElement(mapFunction.map(getElement(i)), i);
        }
    }
}
```

生產者(欲提供物件)

消費者(欲接收物件)

生產者(欲提供物件)

java