# Java 核心技术(高阶)

## 第七章 Lambda表达式

## 第三节 方法引用

华东师范大学 陈良育

# Java Lambda表达式

- Lambda表达式
  - 类似于匿名方法，一个没有名字的方法
  - 可以将方法当作变量，传递给其他方法

```
String[] planets = new String[] {
        "Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus",
        "Neptune" };

System.out.println("使用Lambda, 长度从小到大:");
Arrays.sort(planets,
        (String first, String second)
            -> first.length() - second.length());
System.out.println(Arrays.toString(planets));
```

# 方法引用(1)

- 方法引用：Method Reference
  - Lambda表达式支持传递现有的类库函数

```
String[] planets = new String[] {
        "Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus",
        "Neptune" };

Arrays.sort(planets, String::compareToIgnoreCase);
System.out.println(Arrays.toString(planets));
```

# 方法引用(2)

- 方法引用：Method Reference
  - Class::staticMethod，如 Math::abs方法
  - Class::instanceMethod，如String::compareToIgnoreCase方法
  - object::instanceMethod，如System.out::println方法
    - 支持this::instanceMethod 调用
    - 支持super::instanceMethod调用
  - Class::new，调用某类构造函数，支持单个对象构建
  - Class[]::new，调用某类构造函数，支持数组对象构建

# 方法引用(3)

- 类::静态方法(Class::staticMethod)，如 Math.abs方法
  - 等价于提供方法参数的Lambda表达式
  - Math::abs 等价于 x -> Math.abs(x)

```
interface NumFunction {
    double calculate(double num);
}
```

```
public static double worker(NumFunction nf, double num)
{
    return nf.calculate(num);
}
public static void main(String[] args) {
    double a = -5.3;
    double b = worker(Math::abs, a);
    System.out.println(b);

    double c = worker(Math::floor, a);
    System.out.println(c);
}
```

# 方法引用(4)

- Class::instanceMethod，如String::compareToIgnoreCase方法
  - 第一个参数将变成方法的执行体
  - String::compareToIgnoreCase等价于(x,y)->x.compareToIgnoreCase(y)

```
String[] planets = new String[] {
        "Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus",
        "Neptune" };

Arrays.sort(planets, String::compareToIgnoreCase);
System.out.println(Arrays.toString(planets));
```

# 方法引用(5)

- object::instanceMethod，如System.out.println方法
  - 等价于提供方法参数的Lambda表达式
  - System.out::println等价于x->System.out.println(x)

```
interface PrintFunction {
    public void exec(String s);
}
```

```
public static void worker(PrintFunction pf, String s) {
    pf.exec(s);
}

public static void main(String[] args) {
    String a = "abc";
    worker(System.out::println, a);
}
```

# 方法引用(6)

- object::instanceMethod，支持this::instanceMethod

```
public class ThisInstanceMethodTest {
    public static void main(String[] args) {
        ThisInstanceMethodTest obj = new ThisInstanceMethodTest();
        obj.test();
    }

    public void test()   {
        String[] planets = new String[] {
                "Mercury", "Venus", "Earth", "Mars",
                "Jupiter", "Saturn", "Uranus", "Neptune" };

        Arrays.sort(planets, this::lengthCompare);
        System.out.println(Arrays.toString(planets));
    }

    public int lengthCompare(String first, String second){
        return first.length() - second.length();
    }
}
```

# 方法引用(7)

- object::instanceMethod，支持super::instanceMethod

```
public class SuperInstanceMethodTest extends Father{
    public static void main(String[] args) {
        SuperInstanceMethodTest obj = new SuperInstanceMethodTest();
        obj.test();
    }

    public void test()  {
        String[] planets = new String[] {
                "Mercury", "Venus", "Earth", "Mars",
                "Jupiter", "Saturn", "Uranus", "Neptune" };

        Arrays.sort(planets, super::lengthCompare);
        System.out.println(Arrays.toString(planets));
    }
}

class Father {
    public int lengthCompare(String first, String second){
        return first.length() - second.length();
    }
}
```

# 方法引用(8)

- Class::new，调用某类构造函数，创造一个对象

```
class Person
{
    private String name;
    private int age;

    public Person() {
        this.name = "Tom";
        this.age = 18;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

```
Supplier<Person> s = Person::new;
Person p = s.get();
System.out.println(p.getName());
```

# 方法引用(9)

- Class[]::new，支持数组对象创建

```
IntFunction<int[]> intArray = int[]::new;
int[] nums = intArray.apply(10);

Function<Integer, Person[]> personArray = Person[]::new;
Person[] persons = personArray.apply(5);
```

# 总结

- 了解Lambda的方法引用
- 掌握方法引用的具体语法

# 谢 谢！