Java 核心技术(高阶)

第七章 Lambda表达式

第四节 Lambda表达式应用

华东师范大学 陈良育

Lambda表达式应用(1)

- Lambda表达式应用
  - 类型信息
  - 重载
  - 变量遮蔽
  - this指代
  - Lambda表达式使用，以及和接口方法、嵌套类的关系

Lambda表达式应用(2)

- Lambda表达式
  - 类似于匿名方法，一个没有名字的方法
  - 被赋值后，可以看作是一个函数式接口的实例(对象)
  - 但是Lambda表达式没有存储目标类型(target type)的信息

```
x -> x * 2;
```

```
interface IntOperation {
    int operate(int i);
}

interface DoubleOperation {
    double operate(double i);
}
```

```
IntOperation iop = x -> x * 2;
DoubleOperation dop = x -> x * 2;

Object obj1 = iop;
Object obj2 = dop;
```

Lambda表达式应用(3)

- Lambda表达式
  - 重载调用，依据重载的规则和类型参数推理

```
interface Runnable {
    void run();
}

interface Callable<V> {
    V call();
}
```

```
void exec() {
    String s = invoke(() -> "done");
    System.out.println(s);
}

void invoke(Runnable r) {
    r.run();
}

<T> T invoke(Callable<T> c) {
    return c.call();
}
```

Lambda表达式应用(4)

- Lambda表达式变量遮蔽
  - Lambda表达式和匿名内部类/局部内部类一样，可以捕获变量(capture variables)，即访问外部嵌套块的变量
    - 但是变量要求是final或者是effectively final的
  - Lambda表达式没有变量遮蔽问题，因为它的内容和嵌套块有着相同的作用域
    - 在Lambda表达式中，不可以声明与(外部嵌套块)局部变量同名的参数或者局部变量

Lambda表达式应用(5)

- Lamdba的this指代
  - 表达式中的this，就是创建这个表达式的方法的this参数

```
interface StringOperation {
    String name = "abc";
    public void operate();
}
```

```
public class ThisScopeTest {
    private String name = "def";

    public static void main(String[] args) {
        new ThisScopeTest().test();
    }

    public void test() {
        StringOperation obj = ()->
        {
            System.out.println(this.name);
            System.out.println(getName());
        };
        obj.operate();
    }
    public String getName() {
        return this.name;
    }
}
```

Lambda表达式应用(6)

- Lambda表达式
  - 填充接口的一个抽象方法
  - 从JDK8开始支持的接口有默认方法/静态方法/私有方法

```
//从JDK5开始支持for-each语法，但遍历仍显笨拙
for(String s : pList)
{
    System.out.println(s);
}

//从JDK8开始支持Lamdba表达式，遍历更简洁
pList.forEach(System.out::println);
```

```
public interface Iterable<T> {
    /**
     */
    Iterator<T> iterator();

    /**
     */
    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
```

Lambda表达式应用(7)

- Lambda表达式
  - 优先级比嵌套类要高
    - 短小精干，本身可以自描述的
    - 无法创建命名实例，无法获取自身的引用(this)

```
Comparator<String> c = (String first, String second) ->
    first.length() - second.length();
```

```
Comparator<String> c = new Comparator<String>() {
            public int compare(String first, String second)
            {
                return first.length() - second.length();
            }
        };
```

Lambda表达式应用(8)

- Lambda表达式
  - 方法引用比自定义Lambda表达式的优先级高
    - 系统自带的方法引用更简洁高效
    - 对于复杂的Lambda表达式，采用方法引用比内嵌Lambda表达式更清晰，更容易维护

```
HashMap<String, Integer> counter = new HashMap<>();

for(String word:words)
{
    //counter.merge(word, 1, (count, incr) -> count + incr);
    counter.merge(word, 1, Integer::sum);
}
```

Lambda表达式应用(9)

- Lambda表达式
  - 坚持使用标准的函数式接口
    - 更容易学习，提高互操作性

```
StringChecker evenLength = s ->
     {
          if(s.length()%2 == 0)
               return true;
          return false;
     };

for(String p : planets) {
     if(evenLength.test(p)) {
          System.out.println(p);
     }
}
```

```
Predicate<String> oddLength = s ->
     s.length()%2 == 0 ? false:true;

for(String p : planets) {
     if(oddLength.test(p)) {
          System.out.println(p);
     }
}
```

Lambda表达式应用(10)

- Lambda表达式(官方文档)
  - Use it if you are encapsulating a single unit of behavior that you want to pass to other code. 如果你需要封装一个单独的行为，并且传递给其他的代码。
  - Use it if you need a simple instance of a functional interface and none of the preceding criteria apply (for example, you do not need a constructor, a named type, fields, or additional methods).如果你需要一个简单的函数式接口的实例，并且不需要执行任何的先决条件(如，调用构造函数，给属性赋值，执行额外的方法)

总结

- 学习Lambda的使用注意点
- 了解Lambda的使用场景

谢 谢！