**Mybatis Plus 3 的语法糖演示**

@Override

@CacheEvict(value = "menu\_details", allEntries = true)

@Transactional(rollbackFor = Exception.class)

public Boolean removeRoleById(Integer id) {

sysRoleMenuMapper.delete(Wrappers

.<SysRoleMenu>update().lambda()

.eq(SysRoleMenu::getRoleId, id));

return this.removeById(id);

}

[**#**](https://pig4cloud.com/#%E5%91%BD%E4%BB%A4%E5%BC%8F%E5%92%8C%E5%87%BD%E6%95%B0%E5%BC%8F) **命令式和函数式**

**命令式编程**：命令“机器”如何去做事情(how)，这样不管你想要的是什么(what)，它都会按照你的命令实现。 **声明式编程**：告诉“机器”你想要的是什么(what)，让机器想出如何去做(how)。

[**#**](https://pig4cloud.com/#%E4%BB%80%E4%B9%88%E6%98%AF%E5%87%BD%E6%95%B0%E5%BC%8F%E7%BC%96%E7%A8%8B%EF%BC%9F) **什么是函数式编程？**

每个人对函数式编程的理解不尽相同。 我的理解是：**在完成一个编程任务时，通过使用不可变的值或函数，对他们进行处理，然后得到另一个值的过程。** 不同的语言社区往往对各自语言中的特性孤芳自赏。现在谈 Java 程序员如何定义函数式编程还为时尚早，但是，这根本不重要！ 我们关心的是如何写出好代码，而不是符合函数式编程风格的代码。

[**#**](https://pig4cloud.com/#%E8%A1%8C%E4%B8%BA%E5%8F%82%E6%95%B0%E5%8C%96) **行为参数化**

把算法的策略（行为）作为一个参数传递给函数。

[**#**](https://pig4cloud.com/#lambda-%E7%AE%A1%E4%B8%AD%E7%AA%A5%E8%B1%B9) **lambda 管中窥豹**

* 匿名：它不像普通的方法那样有一个明确的名称：写得少而想得多！
* 函数：Lambda函数不像方法那样属于某个特定的类。但和方法一样，Lambda有参数列表、函数主体、返回类型，还可能有可以抛出的异常列表。
* 传递：Lambda表达式可以作为参数传递给方法或存储在变量中。
* 简洁：无需像匿名类那样写很多模板代码。

[**#**](https://pig4cloud.com/#%E5%87%BD%E6%95%B0%E6%8F%8F%E8%BF%B0%E7%AC%A6) **函数描述符**

函数式接口的抽象方法的签名基本上就是Lambda表达式的签名，这种抽象方法叫作函数描述符。

[**#**](https://pig4cloud.com/#%E5%87%BD%E6%95%B0%E5%BC%8F%E6%8E%A5%E5%8F%A3%EF%BC%8C%E7%B1%BB%E5%9E%8B%E6%8E%A8%E6%96%AD) **函数式接口，类型推断**

函数式接口定义且只定义了一个抽象方法，因为抽象方法的签名可以描述Lambda表达式的签名。 函数式接口的抽象方法的签名称为函数描述符。 所以为了应用不同的Lambda表达式，你需要一套能够描述常见函数描述符的函数式接口。

[**#**](https://pig4cloud.com/#java-8%E4%B8%AD%E7%9A%84%E5%B8%B8%E7%94%A8%E5%87%BD%E6%95%B0%E5%BC%8F%E6%8E%A5%E5%8F%A3) **Java 8中的常用函数式接口**

| **函数式接口** | **函数描述符** | **原始类型特化** |
| --- | --- | --- |
| Predicate<T> | T->boolean | IntPredicate,LongPredicate, DoublePredicate |
| Consumer<T> | T->void | IntConsumer,LongConsumer, DoubleConsumer |
| Function<T,R> | T->R | IntFunction<R>, IntToDoubleFunction, IntToLongFunction, LongFunction<R>, LongToDoubleFunction, LongToIntFunction, DoubleFunction<R>, ToIntFunction<T>, ToDoubleFunction<T>, ToLongFunction<T> |
| Supplier<T> | ()->T | BooleanSupplier,IntSupplier, LongSupplier, DoubleSupplier |
| UnaryOperator<T> | T->T | IntUnaryOperator, LongUnaryOperator, DoubleUnaryOperator |
| BinaryOperator<T> | (T,T)->T | IntBinaryOperator, LongBinaryOperator, DoubleBinaryOperator |
| BiPredicate<L,R> | (L,R)->boolean |  |
| BiConsumer<T,U> | (T,U)->void | ObjIntConsumer<T>, ObjLongConsumer<T>, ObjDoubleConsumer<T> |
| BiFunction<T,U,R> | (T,U)->R | ToIntBiFunction<T,U>, ToLongBiFunction<T,U>, ToDoubleBiFunction<T,U> |

[**#**](https://pig4cloud.com/#lambdas%E5%8F%8A%E5%87%BD%E6%95%B0%E5%BC%8F%E6%8E%A5%E5%8F%A3%E7%9A%84%E4%BE%8B%E5%AD%90) **Lambdas及函数式接口的例子**

| **使用案例** | **Lambda 的例子** | **对应的函数式接口** |
| --- | --- | --- |
| 布尔表达式 | (List<String> list) -> list.isEmpty() | Predicate<List<String>> |
| 创建对象 | () -> new Project() | Supplier<Project> |
| 消费一个对象 | (Project p) -> System.out.println(p.getStars()) | Consumer<Project> |
| 从一个对象中选择/提取 | (int a, int b) -> a \* b | IntBinaryOperator |
| 比较两个对象 | (Project p1, Project p2) -> p1.getStars().compareTo(p2.getStars()) | Comparator<Project> 或 BiFunction<Project, Project, Integer> 或 ToIntBiFunction<Project, Project> |

[**#**](https://pig4cloud.com/#%E6%96%B9%E6%B3%95%E5%BC%95%E7%94%A8) **方法引用**

方法引用让你可以重复使用现有的方法定义，并像Lambda一样传递它们。

[**#**](https://pig4cloud.com/#%E6%9C%AC%E8%8A%82%E8%AF%BE%E5%B0%8F%E7%BB%93) **本节课小结**

* lambda 表达式可以理解为一种匿名函数：它没有名称，但有参数列表、函数主体、返回 类型，可能还有一个可以抛出的异常的列表。
* lambda 表达式让你可以简洁地传递代码。
* 函数式接口就是仅仅声明了一个抽象方法的接口。
* 只有在接受函数式接口的地方才可以使用 lambda 表达式。
* lambda 表达式允许你直接内联，为函数式接口的抽象方法提供实现，并且将整个表达式作为函数式接口的一个实例。
* Java 8自带一些常用的函数式接口，放在 java.util.function 包里，包括 Predicate<T>、Function<T,R>、Supplier<T>、Consumer<T> 和 BinaryOperator<T>。
* Lambda表达式所需要代表的类型称为目标类型。
* 方法引用让你重复使用现有的方法实现并直接传递它们。
* Comparator``、Predicate和Function` 等函数式接口都有几个可以用来结合 lambda 表达式的默认方法。