Homework 6 - Object Oriented Evaluator

本次作业在 elearning 上提交 oo-eval.rkt 文件

遇到问题可以联系助教 23210240333@m.fudan.edu.cn。

A oo-eval

在这个项目中，我们将把面向对象的概念应用到求值器中。你需要理解面向对象编程的一般思想以及面向对象编程系统的实现。

采用命令

```
racket 'oo-eval.rkt'
```

直接加载并运行 `oo-eval.rkt` ，你会发现它会运行一些测试用例然后报错。这是因为该文件加载了一系列测试定义，然后调用了 `run-all-tests` 。在逐个运行测试用例时，可以把这行调用注释掉，但请记得运行所有测试，确保你的修改没有引入新问题。

A.1 Getting started

思考需要在求值器中添加哪些内容：oo-eval 内部用于管理类和实例的数据抽象，以及需要暴露给 oo-eval 用户用于创建类、实例化类以及调用生成实例方法的特殊形式或原始过程。

本次作业提供了一个旧版本的求值器的一部分，里面实现了对象系统。该系统为实例添加一个简单的实现：一个包含符号 `instance` 的标签列表，以及一个包含所有实例本地状态（该状态指槽名称和当前值）的关联列表[1]。其中，一个名为:class 的特殊槽，用于引用实例所属的类。

相关代码片段如下所示：

```
(define (instance? obj)
  (tagged-list? obj 'instance))

(define (instance-state inst)
  (second inst))

(define (instance-class inst)
(read-slot inst ':class))

; Given an object instance and slot name, find the slot's current value
(define (read-slot instance slotname)
  (let ((result (assq slotname (instance-state instance))))
    (if result
      (cadr result)
      (error "no slot named" slotname))))

; Create an object instance from a class
```

[1] 双元素列表的列表，双元素列表指的是包含名称和其值的列表

```
  ; Store list of slots' data (including class as :class) in a tagged-list
  ; All slots other than :class start out with the value 'uninitialized
  ; Need to call the constructor if there is one
  (define (make-instance class . args)
    (let ((instance
           (list 'instance
             (cons (list ':class class)
                   (map
                   (lambda (x) (list x 'uninitialized))
                   (invoke class 'GET-SLOTS))))))


  (if (class-has-method? class 'CONSTRUCTOR)
      (method-call instance 'CONSTRUCTOR class args))

  ; return the constructed instance
  instance))
```

尝试阅读这段代码，我们提供了一些测试用例，这些用例基于专门为本次测试编写的类表示。测试用例定义在以下程序中：`test-problem1-0slots` 和 `test-problem1-3slots`。要运行上述测试，请运行提供的 `oo-eval.rkt` 文件，该文件将运行所有测试。

如果你想单独运行测试，可以从 REPL（DrRacket Interactions 窗口）执行如下操作命令：

```
> (test-instances-0slots)
RUNNING TEST: Getting started: make-instance with no extra args.
test-passed

> (test-instances-3slots)
RUNNING TEST: Getting started: make-instance with arguments.
test-passed
```

A.2 Problem 1

接下来你需要考虑类是如何表示的。

本次作业决定用以下四个信息来表示一个类：

1. 名字（name）
2. 父类（parent class）
3. 槽（slots，也就是属性）的名称
4. 方法（methods）

但我们并没有把这些信息简单地塞进一个带标签的列表中，而是决定用一个现成的数据抽象：实例（`instance`）。

我们认为“类本身也是对象实例”。但如果一个类是一个实例，那它是哪个类的实例呢？也就是说，“类的类”是什么？答案是：元类（`metaclass`）。

我们开始在求值器中加入一个 元对象协议（*Metaobject Protocol*），让这个新语言的用户可以高度控制对象系统的内部细节。~~但这个我们稍后再谈……~~

为了启动整个系统，我们定义了一个叫做 `default-metaclass` 的类作为引导。它看起来就像一个普通的对象实例，但不是通过 `make-instance` 创建的。其他所有的类都可以通过实例化这个类来创建。它包含以下方法：

- `CONSTRUCTOR`：用来设置新类的槽（slots）
- `GET-SLOTS`：在这个新类被实例化时，组装其实例状态
- `FIND-METHOD`：控制方法的查找机制

哦，对了，不要被槽名前面的冒号吓到。这在 Racket 中并没有什么特殊意义，我们只是用冒号建立一致的命名风格。

修复下面的代码，使其能通过给出的测试用例：

```
(define (create-class name parent-class slots methods)
'BRAINSBRAINSBRAINSBRAINS)
```

你需要参考 `default-metaclass` 的 `CONSTRUCTOR` 方法，来确定这些参数的顺序是什么（不需要提供 `self` 参数）。

本次作业提供了 `test-problem1-simple` 和 `test-problem1-subclass` 两个测试用例来验证你的实现。

A.3 Problem 2

好了，你已经掌握了足够多的基本数据结构。现在是时候开始为你的求值器添加对象支持了。先从创建类开始。以下是运行模拟所需支持的完整语法：

```
(make-class <name>
            <parent-class>
            (<slotname1> <slotname2> ...)
            ((<methodname1> (lambda (<args>...) <method1 body>))
             (<methodname2> (lambda (<args>...) <method2 body>))
            . . .))
```

下面是一个例子：

```
(define named-object
  (make-class 'NAMED-OBJECT root-object (name)
    ((NAME (lambda () name))
     (SET-NAME! (lambda (new-name) (set! name new-name))))))
```

注意，并不是每一个子表达式都会被求值。类名和父类会被求值，但槽的列表会被原样保留。另外还有一个方法名与过程（procedure）之间的关联表（association list），其中只有 lambda 表达式需要被求值。由于不能按通常的组合表达式求值规则来处理这些内容，‘make-class‘ 必须是一个特殊形式。

为 make-class 创建一个语法抽象（包括 make-class? 判断函数，以及用于提取该表达式各部分的选择器函数）。在 oo-eval 中添加一个子句，用于检测是否是 make-class 表达式，并将其分发给 eval-make-class 处理。完成 eval-make-class 的实现，并运行测试。

本次作业提供了 test-problem2-simple 和 test-problem2-subclass 两个测试用例来验证你的实现。

这些测试会在一些测试表达式上运行 oo-eval，并检查其结果是否正确。

A.4 Problem 3

为了实例化类，求值器的用户显然使用了新的特殊形式，如下所示：

```
(new <class> <expression1> <expression2> ...)
```

类名之后的所有内容都是**可选的**，并且应当传递给该类的 CONSTRUCTOR 方法。

下面是两个例子：

```
(define sicp
  (new named-object 'Structure-and-Interpretation-of-Computer-Programs))
(define our-clock (new clock))
```

new 不需要是一个特殊形式，因为它会对它的所有参数进行求值，所以我们可以简单地将它实现为 oo-eval 中的一个原始过程（primitive procedure）。

将一个名为 new 的原始过程添加到原始过程列表中，它应该引用一个你之前已经见过的函数。本部分的测试用例为test-problem3 和 test-problem3-no-args。

助教能力有限，如遇到 BUG 请及时反馈。