函数式入门

> 施程航 18/10/19

- ## 函数式编程是个啥子哟

  简单来说，就是把函数的地位与变量等同，我们可以像操作、组合变量那样折腾函数。这里的函数可以直接认为是数学上的映射。

  - ### 开始之前

    让我们回顾下列表、序对和递归

  - ### 数据的映射

    比如，为了得到一个翻转的序对，(1,2) -> (2,1)，函数式点的做法是：

    ```
    ;;不难看出，这段代码用x的cdr和car新建了一个序对
    (define (_reverse x)
        (cons (cdr x) (car x))
    )
    ;;让我们测试下
    (define a (cons 1 2))
    (define b (_reverse a))
    ;;打印a,b
    (display a);;(1,2)
    (display b);;(2,1)
    ```

    也就是说，我们在意的是数据到数据的映射，而不是改变原来的数据，这与数学上的函数相呼应。这里的“不变”可以带来一些好处，后面会提到...

  - ### 高阶函数

    好了，我们又引进了一个新概念，概念总是无聊的：高阶函数就是以函数为参数或者返回函数的函数。这里就是前面说的把函数的地位与变量等同

    ...show me the code

    - 求和函数

    ```
    ;;比如我们想要求两数/两数平方/两数立方之和
    ;;我们可能会这样干
    ;两数之和
    (define (sum-of-self x y)
        (+ x y)
    )
    ;平方
    ```

```
(define (sum-of-square x y)
    (+ (* x x) (* y y))
)
;立方
(define (sum-of-cube x y)
    (+ (* x x x) (* y y y))
)
;;;我是可爱的分割线
;;我们可以发现这三个函数内部实际上都是:
;;(+ (f x) (f y))
;所以我们也可以这样搞
(define (self x) x);;这里可能会有疑问
(define (square x) (* x x))
(define (cube x)(* x x x))
;
(define (sum x y f)
    (+ (f x) (f y))
)
;
(define (sum-of-self x y)
    (sum x y self)
)
(define (sum-of-square x y)
    (sum x y square)
)
(define (sum-of-cube x y)
    (sum x y cube)
)
```

测试代码

```
(display (sum-of-self 3 2))
(display (sum-of-square 3 2))
(display (sum-of-cube 3 2))
```

你可能会说，代码变长了，的确。不过，让我们看看我们是怎样把代码“变长”的：提取公共模式，每个sum函数内部都有(+ (f x) (f y))的形式，把这部分抽出来，用f去代替self square cube，我们得到能够代表三个求和函数的形式sum，它有一个参数f，它表示我们如何转化x y。

- filter(表的过滤) filter的定义类似：

```
(define (filter predicate sequence)
    balabala...
)
```

- map(表到表的映射) map的定义类似：

```
(define (map transform sequence)
    ...
)
```

```
;;这里把列表里的数转成了它们的相反数，然后组成一个新的列表
(map - (list 1 2 3 4));;-> (-1 -2 -3 -4)
;
(define (bigger-than-0 x)
    (> x 0))
(filter bigger-than-0 (list -1 2 4 -2 0 3))
```

- 手写快排？

```
;;本地环境的同学可以在qsort前面贴上这段代码
(define (filter pred seq)
    (if (null? seq)
        '()
        (if (pred (car seq))
            (cons (car seq) (filter pred (cdr seq))
            (filter pred (cdr seq))
            )
        )
    )
)
```

快排是啥？balabala...

```
(define (qsort seq)
    (if (null? seq)
        '()
        (append
        (qsort (filter (lambda (x) (< x (car seq))) seq))
        (filter (lambda (x) (= x (car seq))) seq)
        (qsort (filter (lambda (x) (> x (car seq))) seq))
        )
    ))
(qsort (list 2 1 4 3 -1))
```

如果我们想降序/非升序排列呢…

如果我们想按照其他方式排列呢…

问题来了，借一步说话！

- 我是写C/C++/python/...的，函数式可以干嘛

函数指针、仿函数...这里还是拿快排做例子，用C试试？可以在这里跑下下面的代码

```
这是qsort的声明
void qsort(void *base,//数组的第一个元素的指针
            size_t nitems,//元素个数
            size_t size,//元素占内存的大小
            int (*comp)(const void *, const void*)//比较元素大小的函数
        );
```

```
//ps:这里看不懂不要紧，因为指针这部分是有、难...知道大概的想法就Ok(吐槽，这里指针的语法真的丑...)
#include <stdio.h>
#include <stdlib.h>

int values[] = {3, 6, 7, 1, 2, 3,10};
int cmp1(const void* a, const void *b)
{
    return (*(int*)a - *(int*)b);
}
int cmp2(const void* a, const void *b)
{
    int _a = *(int*)a, _b = *(int*)b;
    if(_a%3 == _b%3)
        return _a - _b;
    else
        return (_a%3 - _b%3);
}

int main()
{
    qsort(values, 7, sizeof(int), cmp1);
    int n;
    printf("第一次排序:\n");
    for(n = 0; n < 7;n++)
    {
        printf("%d  ",values[n]);
    }
    printf("\n");

    //我是超可爱的分割线


    qsort(values, 7, sizeof(int), cmp2);
    printf("第二次排序:\n");
    for(n = 0; n < 7;n++)
    {
        printf("%d  ",values[n]);
    }
    printf("\n");
```

```
    return 0;
}
```

C++的仿函数有兴趣的同学欢迎来py，待我去学习一波然后与你讨论分享

python我不熟就不班门弄斧了，有兴趣的可以看看廖雪峰老师的python课程中的函数式编程部分

hint:接触不同的语言和思想

- 拓展 ---- 延时求值

```
;;写个从n开始计数的序列
(define (counter-from n)
    (cons n (counter-from (+ n 1)))
)
(display (counter-from 1))
(display "来追我呀！")
;;;看看发生了啥
```

下面顺理成章地引入延时求值的概念，听着有点高大上，虽然确实有、东西。就是需要一个表达式的值再去求值这个表达式。延时求值和函数式有啥关系吗

- 总结与希望与反省....

语言和范式影响思考的方式

第一次讲课...

- 欢迎大家下次、下下次、下下下次...来蹭大佬们开的课！