Боремся со сложностью по-функциональному

Роман Чепляка

28 января 2012
AgileBaseCamp

Борьба со сложностью

- повторное использование кода
- комбинируемость
- DRY

Функциональный

Функциональный

- Функции высшего порядка

```
filter :: (a -> Bool) -> [a] -> [a]
```

Функциональный

- Функции высшего порядка

```
filter :: (a -> Bool) -> [a] -> [a]
```

- λ-функции

```
filteredList :: [Int]
filteredList = filter (\x -> x > 5) [1..10]
```

Итерация

Задача: вычислить сумму элементов коллекции

```
int s = 0;
for ( iter = a.begin(); iter != a.end(); iter++ )
{
    s += *iter;
}
```

Итерация

```haskell
fold :: (s -> a -> s) -> s -> [a] -> s
```

Итерация

```haskell
fold :: (s -> a -> s) -> s -> [a] -> s

sum list  = fold (\s a -> s + a) 0 list
```

Итерация

```haskell
fold :: (s -> a -> s) -> s -> [a] -> s

sum list  = fold (\s a -> s + a) 0 list

prod list = fold (\s a -> s * a) 1 list
```

Итерация

```haskell
fold :: (s -> a -> s) -> s -> [a] -> s

sum list  = fold (\s a -> s + a) 0 list

prod list = fold (\s a -> s * a) 1 list

size list = fold (\s a -> s + 1) 0 list
```

Maybe

```haskell
data Maybe a = Nothing | Just a
```

Maybe

```haskell
data Maybe a = Nothing | Just a

foo =
    case bar of
        Just n -> n
        Nothing -> 0
```

Maybe

```haskell
fromMaybe :: a -> Maybe a -> a
fromMaybe d m =
    case m of
        Just n -> n
        Nothing -> d
```

Maybe

```
foo =
    case bar of
        Just n -> n
        Nothing -> factorial 1000000
```

Maybe

```haskell
foo = fromMaybe (factorial 1000000) bar
```

Ленивый

Ленивый

Функция вычисляет свой аргумент только

если он нужен

Ленивый

Функция вычисляет свой аргумент только

если он нужен

в той степени, в которой он нужен

Повторное использование кода

```haskell
lotteryTickets :: Statistics -> [Ticket]
```

Повторное использование кода

```
lotteryTickets :: Statistics -> [Ticket]

myTickets = take 10 (lotteryTickets stats)
```

Чистый

Чистый

Упрощает комбинирование функций

Чистый

Упрощает тестирование функций

```
prop :: [Int] -> Bool
prop list = isSorted (mySort list)
```

Чистый

Упрощает тестирование функций

```
prop :: [Int] -> Bool
prop list = isSorted (mySort list)

> quickCheck prop
+++ OK, passed 100 tests.
```

Итоги

Функции	помогают структурировать программы
Ленивость	снижает стоимость комбинирования функций
Чистота	делает код модульным и упрощает тестирование