REiA

A Ruby-like language for the Erlang VM

http://github.com/tarcieri/reia

Rhymes with Leia...

...not diarrhea

What does it mean?

What does it mean?

- Not named after Princess Leia

What does it mean?

- Not named after Princess Leia
- An acronym?

What does it mean?

- Not named after Princess Leia
- An acronym?
- Ruby Erlang something something?

What does it mean?

- Not named after Princess Leia
- An acronym?
- Ruby Erlang something something?
- It doesn’t mean anything

What does it mean?

- Not named after Princess Leia
- An acronym?
- Ruby Erlang something something?
- It doesn’t mean anything
- I made it up

State of Reia

State of Reia

- One year old (as of May 11th)

State of Reia

- One year old (as of May 11th)
- Prototype

State of Reia

- One year old (as of May 11th)
- Prototype
- Nearing alpha-stage

State of Reia

- One year old (as of May 11th)
- Prototype
- Nearing alpha-stage
- Ready for eager early adopters

Ruby features you may miss...

Ruby features you may miss...

- Modifiable core

Ruby features you may miss...

- Modifiable core
- Monkeypatching

Ruby features you may miss...

- Modifiable core
- Monkeypatching
- Mutable state

Ruby features you may miss...

- Modifiable core
- Monkeypatching
- Mutable state
- Everything is an object

Ruby features you may miss...

- Modifiable core
- Monkeypatching
- Mutable state
- Everything is an object
- Perlisms

What you get instead

What you get instead

- Erlang-style concurrency

What you get instead

- Erlang-style concurrency
- Excellent I/O support

What you get instead

- Erlang-style concurrency
- Excellent I/O support
- Sane “live update” code swapping

What you get instead

- Erlang-style concurrency
- Excellent I/O support
- Sane “live update” code swapping
- Pattern matching

What you get instead

- Erlang-style concurrency
- Excellent I/O support
- Sane “live update” code swapping
- Pattern matching
- List comprehensions

What you get instead

- Erlang-style concurrency
- Excellent I/O support
- Sane “live update” code swapping
- Pattern matching
- List comprehensions
- Binaries

Syntax Tour

No longer indent sensitive!!!

Syntax example

```
# The Greeter class
class Greeter
  def initialize(name)
    @name = name.capitalize()
  end

  def salute
    "Hello #{@name}!".puts()
  end
end

# Create a new object
g = Greeter("world")
g.salute()
g.kill()
```

Lists

Reia

```
[1,2,3]
[1,2,3,*rest]
```

Erlang

```
[1,2,3]
[1,2,3|Rest]
```

Tuples

Reia

```
(1,2,3)
(1,)
()
```

Erlang

```
{1,2,3}
```

Atoms

Reia

```
:foobar
```

Erlang

```
foobar
```

Maps (i.e. Dicts)

Reia

```
{:foo=>1,:bar=>2,:baz=>3}
```

Erlang

```
{dict,3,16,16,8,80,48,...}
```

Strings

Reia

```
“Hello, #{name}”
‘Hello, Robert’
```

Erlang

```
“Hello, Joe”
```

Binaries

(Same as Erlang)

```
<<1,2,3>>
<<“Foobar”>>
```

Regular Expressions

Reia

```
/fo{2}b[a-z]r/
```

Erlang

```
N/A
```

Ranges

Reia

```
1..10
```

Erlang

```
lists:seq(1,10) % kinda
```

Pattern Matching

Reia

```
(a,b,c)=(1,2,3)
```

Erlang

```
{A,B,C}={1,2,3}
```

Blocks

Brace form

```
[1,2,3].map { |n| n * 2 }
```

Do/End Form

```
Mnesia.transaction do
  Mnesia.read(:user, id)
end
```

Funs

Reia

```
fn = fun(n) { n + 2 }
fn(2)
```

Erlang

```
Fun = fun(N) -> N + 2 end,
Fun(2).
```

Function Calls

Reia

```
Foobar.baz()
```

Erlang

```
‘Foobar’:baz().
```

Function References

Reia

```
fn = Foo.baz
Baz.qux(&fn)
```

Erlang

```
Fn = fun foo:baz/0,
baz:qux(Fn).
```

Processes

Reia

```
pid = Process.spawn(&myfunc)
pid ! message
```

Erlang

```
Pid = proc_lib:spawn(fun myfunc/0),
Pid ! Message.
```

List Comprehensions

Reia

```
[n * 2 | n in 0..10]
```

Erlang

```
[N * 2 || N <- lists:seq(0, 10)]
```

Object System

What is OOP?

What is OOP?

- Messaging

What is OOP?

- Messaging
- Hidden state

What is OOP?

- Messaging
- Hidden state
- Polymorphism/inheritance

Messaging

"I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages (so messaging came at the very beginning -- it took a while to see how to do messaging in a programming language efficiently enough to be useful)."

— Alan Kay, creator of Smalltalk

Imperative OOP

Kool-Aid

Kool-Aid

- Object

Kool-Aid

- Object
- Send message

Kool-Aid

- Object
- Send message
- Invoke method

Kool-Aid

- Object
- Send message
- Invoke method

Kool-Aid

- Object
- Send message
- Invoke method

Reality

Kool-Aid

- Object
- Send message
- Invoke method

Reality

- State

Kool-Aid

- Object
- Send message
- Invoke method

Reality

- State
- Call function

Kool-Aid	Reality
• Object	• State
• Send message	• Call function
• Invoke method	• Mutate state

Kool-Aid

- Object
- Send message
- Invoke method

Reality

- State
- Call function
- Mutate state

FAIL

C++ Style OOP

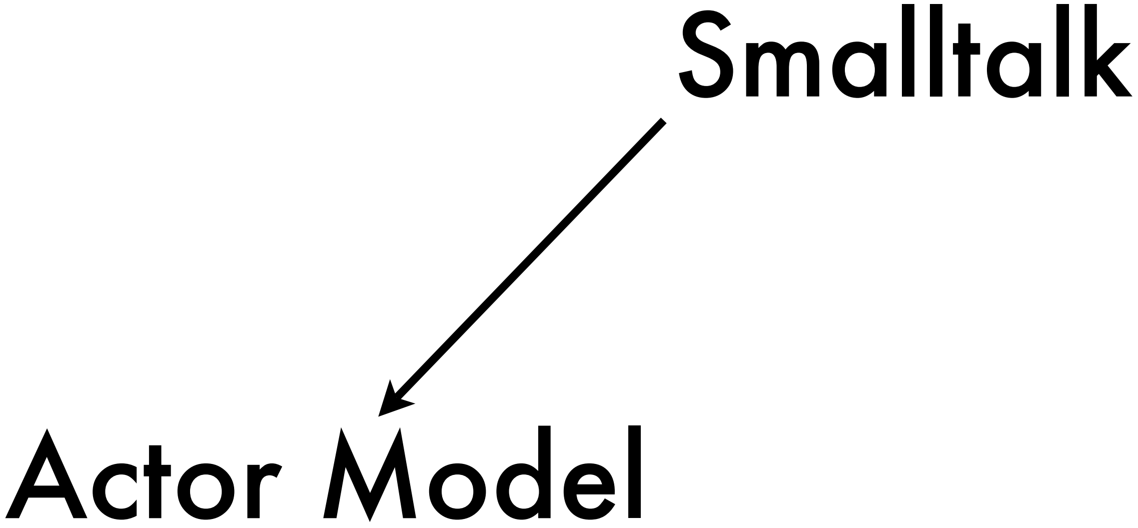
"I made up the term object-oriented, and I can tell you I did not have C++ in mind."

— Alan Kay, creator of Smalltalk

Journey to Reia

Smalltalk

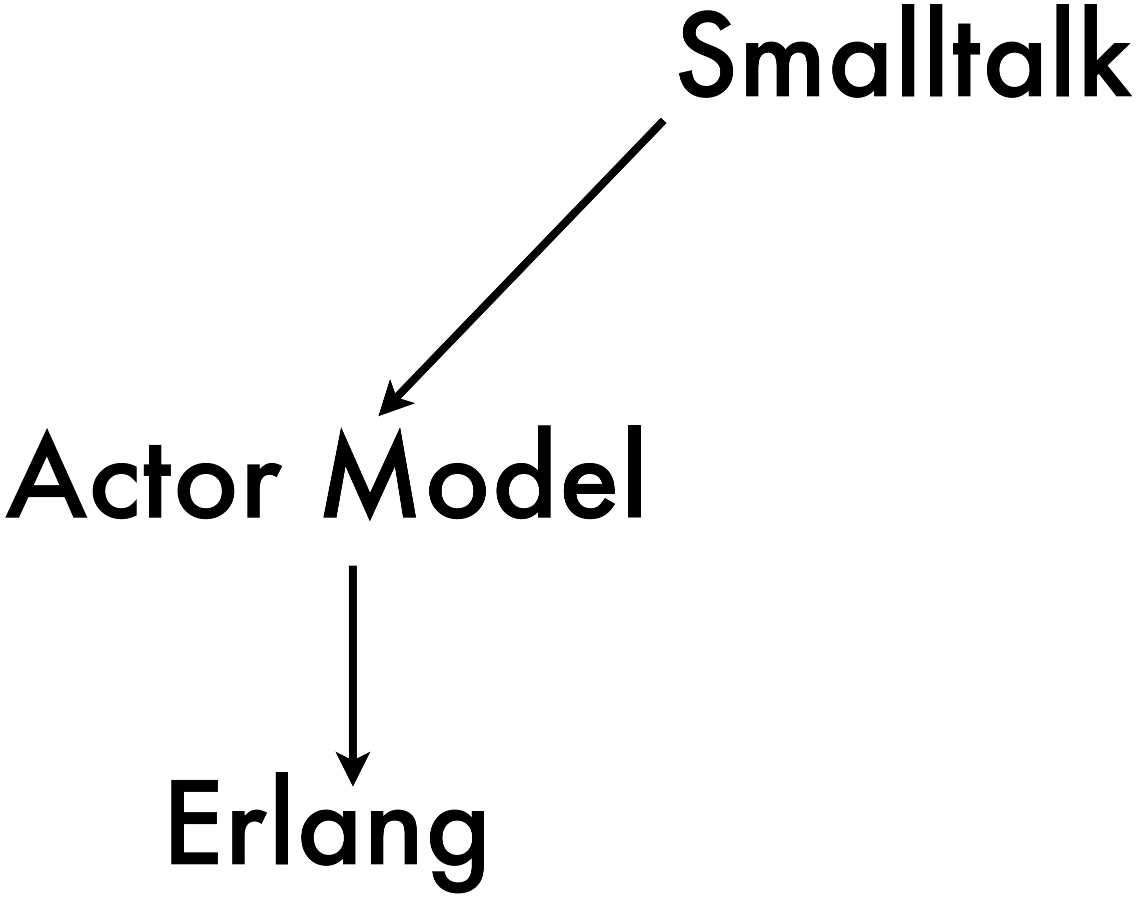
Journey to Reia



Smalltalk

Actor Model

Journey to Reia

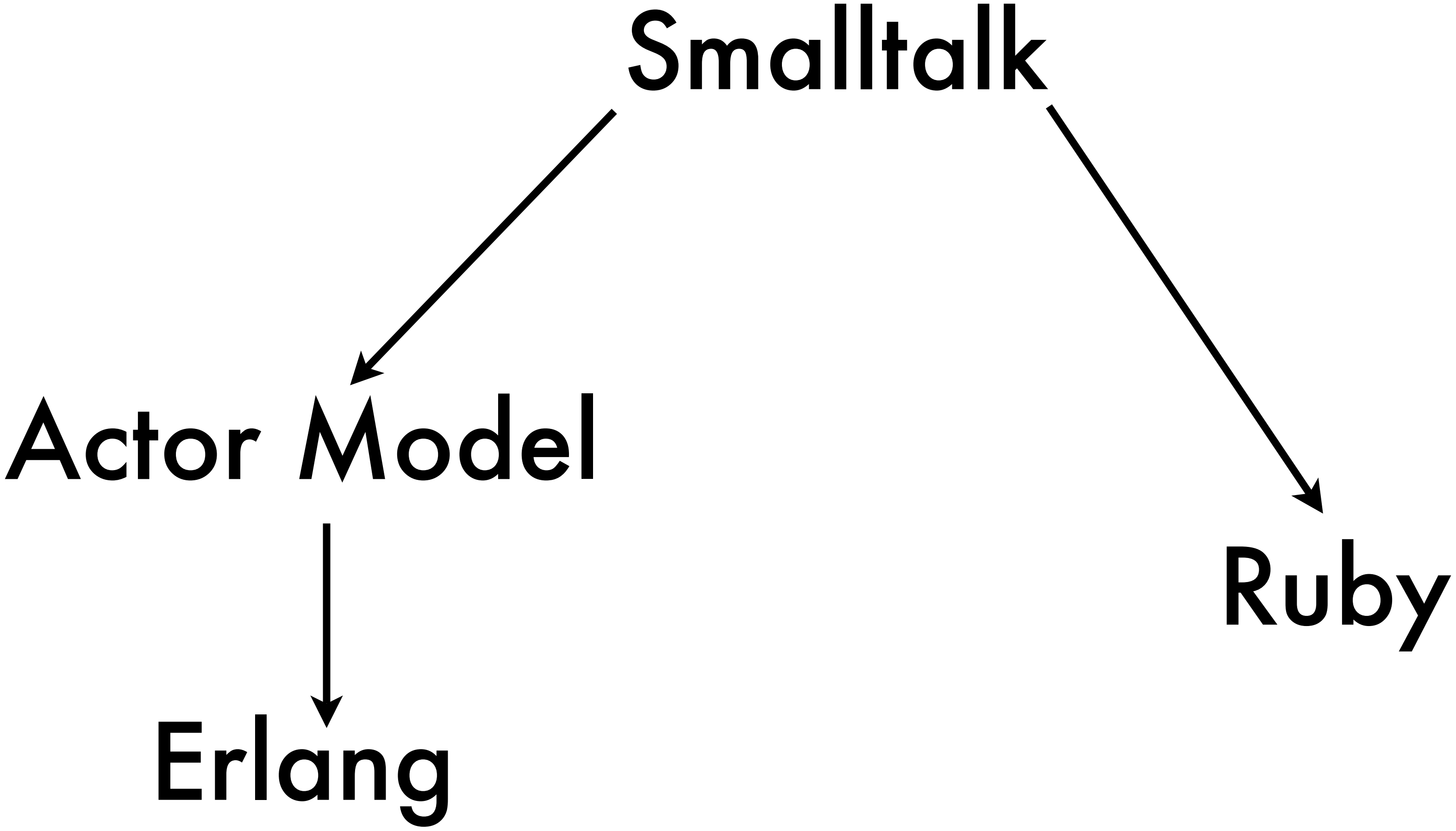


Smalltalk

Actor Model

Erlang

Journey to Reia



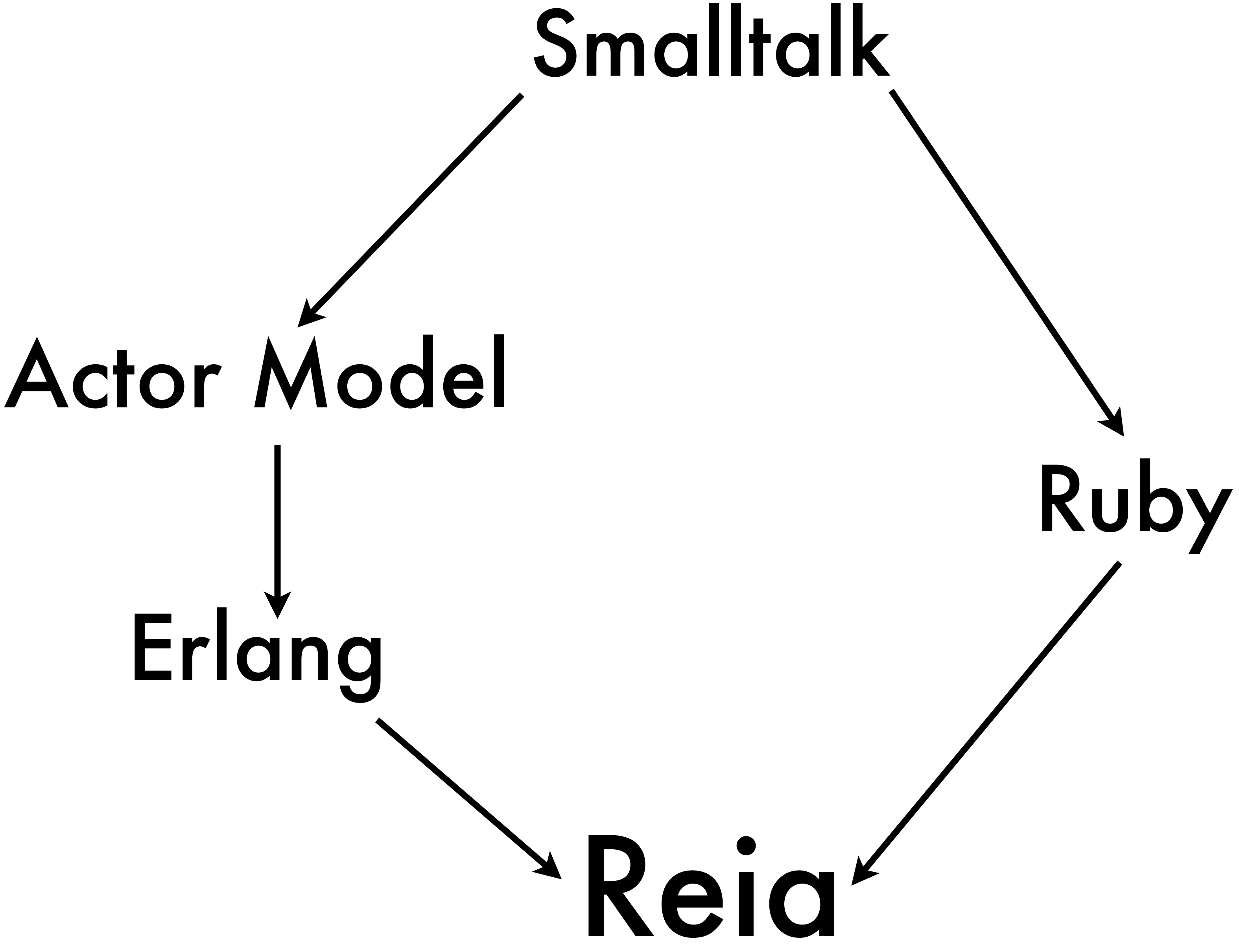
Smalltalk

Actor Model

Erlang

Ruby

Journey to Reia



Reia objects are concurrent

Reia objects really communicate with messages

Objects aren't a one-size-fits-all solution

Messaging

“95% of the time standard synchronous RPCs will work - but not all the time, that's why it's nice to be able to open up things and muck around at the message passing level.”

— Joe Armstrong, creator of Erlang

Defining Classes

```
class Adder
  def initialize(n)
    @n = n
  end

  def plus(x)
    @n + x
  end
end
```

Instantiating Classes

```
Adder.spawn(2)
Adder.spawn_link(2)

Adder(2)

>> a = Adder(2)
=> #<Adder:0.214.0>
```

Invoking Methods

Synchronous (RPC)

```
a.plus(2)
```

Asynchronous (Cast)

```
a<-plus(2)
```

Inheritance & Polymorphism

Inheritance

“Coordinating activities involving multiple actors is very difficult. You can't observe anything without its cooperation/coordination - making ad-hoc reporting or analysis impossible, instead forcing every actor to participate in each protocol.”

— Rich Hickey, creator of Clojure

Inheritance

```
class Animal
  def initialize(name)
    @name = name
  end

  def name
    @name
  end
end

class Cat < Animal
  def talk
    'Meow!'
  end
end

class Dog < Animal
  def talk
    'Woof! Woof!'
  end
end

animals = [Cat('Missy'), Dog('Mr. Bojangles'), Dog('Lassie')]

animals.each do |animal|
  "#{animal.name()} the #{animal.class()} says: #{animal.talk()}".puts()
end
```

What’s the catch?

Garbage Collection

Garbage Collection

- It doesn’t exist for objects

Garbage Collection

- It doesn’t exist for objects
- Use linking

Garbage Collection

- It doesn’t exist for objects
- Use linking
- Use explicit termination

Garbage Collection

- It doesn’t exist for objects
- Use linking
- Use explicit termination
- Use deterministic finalization strategies

Garbage Collection

- It doesn’t exist for objects
- Use linking
- Use explicit termination
- Use deterministic finalization strategies
- Use fewer objects

Circular Calls

Circular Calls

- Call loops cause deadlocks

Circular Calls

- Call loops cause deadlocks
- Magical workaround???

Circular Calls

- Call loops cause deadlocks
- Magical workaround???
- No

Circular Calls

- Call loops cause deadlocks
- Magical workaround???
- No
- But it can be detected

Destructive Assignment

State in Reia is immutable

Static Single Assignment

Reia

```
a = a + 1
```

Erlang

```
A1 = A0 + 1.
```

Rebind on Update

Reia

```
m = {}
m[:foo] = 42
```

Erlang

```
D0 = dict:new(),
D1 = dict:store(foo, 42, D0).
```

Ruby-style Immutability

Pure

```
list.reverse()
```

“Dangerous”

```
list.reverse!()
```

Matz on "Immutable Ruby"

"I have once dreamed of a such language, and had a conclusion that was not Ruby at least, although a pretty interesting language."

— Yukihiro "Matz" Matsumoto, Creator of Ruby

So what?

Ryan

A Web Framework for Reia

By Phil Pirozhkov

http://github.com/pirj/ryan

http://groups.google.com/group/ryan-framework

Ryan Demo

Ryan Controller

```
class Mail < Controller
  def new
    selected = {}.insert(:new, :selected)
    total = Mailbox.total()
    values = {}
    if(@parameters[:error] != nil)
      to = @parameters[:to]
      message = @parameters[:message]
      error = @parameters[:error]
      values = {}.insert(:to, to).insert(:message, message).insert(:error,
error).insert(:error_class, :error)
    end
    contents = view('mail/new', values)
    bindings = {}.insert(:contents, contents).insert(:total, total).insert(:selected,
selected)
    render('home', bindings, [])
  end
```

Retem Template

```
      <div id=menu class=float>
        <a class={selected.home} icon=home href='/app/home'>home<span>general info</
span></a>
        <a class={selected.new} icon=mail_new href='/app/mail/new'>new<span>create 
message</span></a>
        <a class={selected.unread} icon=mail_unread href='/app/mail/
unread'>unread<span>{total.unread} new messages</span></a>
        <a class={selected.inbox} icon=mail_inbox href='/app/mail/
inbox'>inbox<span>{total.inbox} messages</span></a>
        <a class={selected.sent} icon=mail_sent href='/app/mail/
sent'>sent<span>{total.sent} messages</span></a>
        <a class={selected.spam} icon=mail_spam href='/app/mail/
spam'>spam<span>{total.spam} messages</span></a>
        <a class={selected.trash} icon=mail_trash href='/app/mail/
trash'>trash<span>{total.trash} messages</span></a>
      </div>
```

Future Features

Immutable Objects

Immutable Objects

- Immutable objects don’t need to be Erlang processes

Immutable Objects

- Immutable objects don’t need to be Erlang processes
- Immutable objects can be garbage collected

Immutable Objects

- Immutable objects don’t need to be Erlang processes
- Immutable objects can be garbage collected
- Almost everything could be an object

Immutable Objects

- Immutable objects don’t need to be Erlang processes
- Immutable objects can be garbage collected
- Almost everything could be an object
- Immutable objects could tap into “rebind on update”

Default Arguments

Declaration

```
def foo(bar, baz: 2, qux: 3)
```

Invocation

```
foo(1)
```

Keyword Arguments

Declaration

```
def foo(bar, baz: 2, qux: 3)
```

Invocation

```
foo(1, qux: 4, baz: 3)
```

“Splatted” Arguments

Declaration

```
def foo(*list)
```

Invocation

```
foo(:foo, :bar, :baz)
foo(*list)
```

Namespaces

Operator Overloading

instance_eval

Class Bodies

Metaclasses

DSLs

Reflection

Links

Main Site:

http://reia-lang.org

Github:

http://github.com/tarcieri/reia

Blog:

http://unlimitednovelty.com

Twitter:

http://twitter.com/bascule