GOLO, THE TINY LANGUAGE THAT GIVES SUPER POWERS

Philippe Charrière
GitHub

JDD 2016

Philippe Charrière

@k33g

@k33g_org ⚠️🇬🇧🇫🇷

Golo committer

“Another language for the JVM”

- Agenda -

The basis of Golo

Some specific features

How Golo ❤️ Java

I use it for my job

How to hack Golo with Java or with Golo

What is Golo?

Golo

citi lab

@jponge

A research project > Julien Ponge @ CitiLab

A dynamic language for the JVM

Built from the first day with invokedynamic

Light (~700 kb) and fast (in a dynamic context)

Simple, JVM-friendly Golo may aid IoT developers

Credit: Thinkstock

With concurrency and runtime improvements, the JVM language becomes more attractive for IoT development

By Paul Krill | Follow
InfoWorld | Sep 28, 2016

RELATED TOPICS

Java

Internet of Things

COMMENTS

Developers of the Eclipse Foundation's Golo language for the JVM are exploring improvements like concurrency models and improved runtime performance to boost the language's internet of things (IoT) development capability.

Golo, a simple, dynamic, weakly typed language favoring explicit over implicit, was born out of experiments by the Dynamid research team at the Center for

Programming languages

Philippe

GitHub, Inc. [US] https://github.com/showcases/programming-languages

dart-lang/sdk — Dart ★ 831 ⑂ 125
The Dart SDK, including the VM, dart2js, core libraries, and more.

amber-smalltalk/amber — ★ 720 ⑂ 132
An implementation of the Smalltalk language that runs on top of the JS runtime
http://amber-lang.net

factor/factor — Factor ★ 500 ⑂ 102
Factor programming language - Github mirror of official GIT repo

chapel-lang/chapel — Chapel ★ 320 ⑂ 109
a Productive Parallel Programming Language

dylan-lang/opendylan — Dylan ★ 228 ⑂ 49
Open Dylan compiler and IDE

eclipse/golo-lang — Java ★ 223 ⑂ 71
Golo - a lightweight dynamic language for the JVM.
Starred by 7 people

gosu-lang/gosu-lang — Gosu ★ 154 ⑂ 38
The Gosu programming language

Why Golo?

Easy to learn

Easy to use

Easy to hack

Small

a language to learn and try

DSL for children

```
tiger("tigrou 🐯")
  : start(4, 2)
  : move(2): right(): left()
  : stop()
  : say("hello")
```

a language to learn and try

IOT

```
# firmata4j
let myArduino = device(): port("/dev/cu.usbmodem1411")
myArduino: initialize(): onSet(|self| {

  let redLed = self: getLedInstance("red", 13)
  redLed: switchOn()

  self: loop(50, |i| {
    redLed: blink(1000_L)
  })

}): onFail(|err| {
    println("Huston? We've got a problem!")
})
```

Simple

```
module operations

function addition = |a, b| {
  return a + b
}
```

```
module operations

function addition = |a, b| {
  return a + b
}
```

```
module operations

function addition = |a, b| {
  return a + b
}

function multiplication = |a, b| -> a * b
```

```
module operations

function addition = |a, b| {
  return a + b
}

function multiplication =
  |a, b| -> a * b
```

```
#!/usr/bin/env golosh
module demo

import operations

function main = |args| {

  println(addition(40,2))
  println(multiplication(21,2))

}
```

```
module operations

function addition = |a, b| {
  return a + b
}

function multiplication =
  |a, b| -> a * b
```

```
#!/usr/bin/env golosh
module demo

import operations

function main = |args| {

  println(addition(40,2))
  println(multiplication(21,2))

}
```

Run Golo scripts

several ways

golo command

compilation (class and jar)

as a shell script

Run it

```
#!/usr/bin/env golosh
module demo

import operations

function main = |args| {

  println(addition(40,2))
  println(multiplication(21,2))

}
```

Run it

```
chmod +x hello.golo
```

```
./hello.golo
```

```
#!/usr/bin/env golosh
module demo

import operations

function main = |args| {

  println(addition(40,2))
  println(multiplication(21,2))

}
```

Run it

```
./
└── libs
    └── libA.jar
    └── libB.jar
└── commons
    └── utils.golo
    └── others.golo
    └── vendors
        └── otherlib.jar
└── hello.golo
```

```
#!/usr/bin/env golosh
module demo

import operations

function main = |args| {

  println(addition(40,2))
  println(multiplication(21,2))

}
```

Some features
(my favorite features)

Closures

```
#!/usr/bin/env golosh
module demo

function main = |args| {

  let division = |a,b| {
    return a/b
  }

  try {
    println(division(84, 0))
  } catch (err) {
    println("😡: " + err: message())
  }
}
```

Simple and Efficient

Composition rather than Inheritance

Mixed approach (object / functional)

No Class!

but...

```
struct human = {
    firstName,
    lastName
}
```

```
module demo1_structure

struct human = {
  firstName, lastName
}

function main = |args| {
  # no new keyword
  let john = human("John", "Doe")

  # named parameters
  let jane = human(firstName="Jane", lastName="Doe")



}
```

```
module demo1_structure

struct human = {
  firstName, lastName
}

function main = |args| {
  # no new keyword
  let john = human("John", "Doe")

  # named parameters
  let jane = human(firstName="Jane", lastName="Doe")

}
```

```
module demo1_structure

struct human = {
  firstName, lastName
}

function main = |args| {
  # no new keyword
  let john = human("John", "Doe")

  # named parameters
  let jane = human(firstName="Jane", lastName="Doe")

  # colon notation
  let bob = human(): firstName("Bob"): lastName("Morane")

  println("Hello, I'm " + bob: firstName() + " " + bob: lastName())
}
```

```
module demo1_structure

struct human = {
  firstName, lastName
}

function main = |args| {
  # no new keyword
  let john = human("John", "Doe")

  # named parameters
  let jane = human(firstName="Jane", lastName="Doe")

  # colon notation
  let bob = human(): firstName("Bob"): lastName("Morane")

  println("Hello, I'm " + bob: firstName() + " " + bob: lastName())

}
```

Augmentations

```
module demo_structure

struct human = {
  firstName, lastName
}

augment human {
  # this is a ref to human structure
  function sayHello = |this| ->
    "Hello, I'm " + this: firstName() + " " + this: lastName()
}

function main = |args| {
  # colon notation
  let jane = human()
    : firstName("Jane")
    : lastName("Doe")

  println(jane: sayHello())
}
```

```
module demo_structure

struct human = { firstName, lastName }

augment human {
  function sayHello = |this| ->
    "Hello, I'm " + this: firstName() + " " + this: lastName()
}

function Human = |firstName, lastName| {
  # I'm the human constructor
  return human(firstName, lastName)
}

function main = |args| {
  # colon notation
  let jane = Human("Jane", "Doe")

  println(jane: sayHello())
}
```

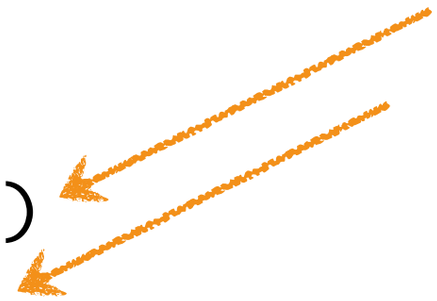
kind of constructor

No Class! Part 2

but...

DynamicObjects

```
module demo_dynamicobjects
function main = |args| {

  let jane =
    DynamicObject()
      : firstName("Jane")
      : lastName("Doe")
      : define("sayHello", |this| {
          return "Hello, I'm " + this: firstName() +
          " " + this: lastName()
        })

  println(jane: sayHello())

}
```



```
module demo_dynamicobjects
function main = |args| {

  let jane =
    DynamicObject()
      : firstName("Jane")
      : lastName("Doe")
      : define("sayHello", |this| {
          return "Hello, I'm " + this: firstName() +
          " " + this: lastName()
        })

  println(jane: sayHello())

}
```

```
module demo_dynamicobjects

function Human = |firstName, lastName| {
  # I'm the human constructor
  return
    DynamicObject()
      : firstName(firstName)
      : lastName(lastName)
      : define("sayHello", |this| {
        return "Hello, I'm " + this: firstName() +
        " " + this: lastName()
      })
}

function main = |args| {
  let jane = Human("Jane", "Doe")
  println(jane: sayHello())
}
```

Golo is functional

functions are 👑

function as output & input

```
module demo_func_01

function addition = |a, b| { return a + b }

function main = |args| {
  # function as output
  let add = |a| {
    return |b| {
      return addition(a,b)
    }
  }
}
```

```
}
```

function as output & input

```
module demo_func_01

function addition = |a, b| { return a + b }

function main = |args| {
  # function as output
  let add = |a| {
    return |b| {
      return addition(a,b)
    }
  }

  let addOne = add(1)
  println(addOne(41)) # 42

  println(
    list[1, 2, 3, 4, 5]: map(addOne)
    # [2, 3, 4, 5, 6]
  )
}
                                  # function as input
```

function as output & input

```
module demo_func_01

function addition = |a, b| { return a + b }

function main = |args| {

  let add = |a| -> |b| -> addition(a,b)

}
```

function as output & input

```
module demo_func_01

function addition = |a, b| { return a + b }

function main = |args| {

  let add = |a| -> |b| -> addition(a,b)

  let addOne = add(1)

  println(
    list[1, 2, 3, 4, 5]
      : map(addOne) # [2, 3, 4, 5, 6]
      : map(addOne) # [3, 4, 5, 6, 7]
  )
}
```

no iteration

no iteration

```
let noChips = |food| -> food isnt "🍟"
# return true if it's not chips

let noOnion = |food| -> food isnt “😭"

noChips("🍞") == true
noChips("🍟") == false
```

no iteration

```
let noChips = |food| -> food isnt "🍟"
let noOnion = |food| -> food isnt "😭"

let cutFood = |food| -> "piece of " + food
```

```
let Food = list[
    "🍞"
  , "🌿"
  , "🍅"
  , "🍗"
  , "🍟"
  , "😭"   # 😡 can't find onion emoji
]
```

no iteration

[🍞, 🌿, 🍅, 🍗, 🍟, 😭]

```
let myKebabRecipe = Food
        : filter(noChips)
        : filter(noOnion)
        : map(cutFood)
```

[🍞, 🌿, 🍅, 🍗, 😭]

[🍞, 🌿, 🍅, 🍗]

[piece of 🍞, piece of 🌿, piece of 🍅, piece of 🍗]

no iteration

```
# ⚡ preparing kebab with myKebabRecipe
let mixFood = |accFood, nextFood| -> accFood + nextFood + " "
```

```
[piece of 🍞, piece of 🍃, piece of 🍅, piece of 🍗]
```

```
let kebab = myKebabRecipe
      : reduce("🌮 with: ", mixFood)
```

```
🌮 with: piece of 🍞 piece of 🍃 piece of 🍅 piece of 🍗
```

Union Types

Golo is functional

```
module SomeThing

union SomeThing = {
  Yum = { value }   # good
  Yuck = { value }  # bad
}
# you're good or bad, but not good and bad
```

```
module SomeThing

union SomeThing = {
  Yum = { value }    # good
  Yuck = { value }   # bad
}
# you're good or bad, but not good and bad
```

```
union SomeThing.Yum{value=🍌}
```

```
let banana = SomeThing.Yum("🍌")
let hotPepper = SomeThing.Yuck("🌶")
```

```
union SomeThing.Yuck{value=🌶}
```

```
let banana = SomeThing.Yum("🍌")
let hotPepper = SomeThing.Yuck("🌶")

println(banana: value())
println(hotPepper: value())
```

```
let banana = SomeThing.Yum("🍌")
let hotPepper = SomeThing.Yuck("🌶")

println(banana: value())
println(hotPepper: value())

# you can do that:
println(banana: isYum())       # true
println(hotPepper: isYuck())   # true
```

```
let onlyBananas = |value| {
  if value is "🍌" { return SomeThing.Yum(value)}
  return SomeThing.Yuck("🌶")
}

if onlyBananas("🍋"): isYum() {
  println("I 💙 bananas")
} else {
  println("💔")
}
```

and... Augmentations

```
union SomeThing = {
  Yum = { value }    # good
  Yuck = { value }   # bad
}

augment SomeThing$Yum {
  function so = |this, ifYum, ifYuck| -> ifYum(this: value())
}

augment SomeThing$Yuck {
  function so = |this, ifYum, ifYuck| -> ifYuck(this: value())
}
```

```
union SomeThing = {
  Yum = { value }    # good
  Yuck = { value }   # bad
}

augment SomeThing$Yum {
  function so = |this, ifYum, ifYuck| -> ifYum(this: value())
}

augment SomeThing$Yuck {
  function so = |this, ifYum, ifYuck| -> ifYuck(this: value())
}
```

```
let onlyBananas = |value| {
  if value is "🍌" { return SomeThing.Yum(value)}
  return SomeThing.Yuck(value)
}
```

```
onlyBananas("🍋"): so(
  |value| {
    println("I 💙 " + value)
  },
  |value| {
    println("I 💔 " + value)
  })
```

Functional Errors

Golo is functional

Result (and Error)

Golo is functional

```
let toInt = |value| {
  try {
    return Result(Integer.parseInt(value))
  } catch(e) {
    return Error(e: getMessage())
  }
}
```

```
let toInt = |value| {
  try {
    return Result(Integer.parseInt(value))
  } catch(e) {
    return Error(e: getMessage())
  }
}

let result = toInt("Fourty-two"): either(
  |value| {
    println("Succeed!")
    return value
  },
  |err| {
    println("Failed!" + err: message())
    return 42
  })
```

all is ok →

ouch! →

```
let toInt = |value| {
  try {
    return Result(Integer.parseInt(value))
  } catch(e) {
    return Error(e: getMessage())
  }
}

let result = toInt("Fourty-two"): either(
  |value| {
    println("Succeed!")
    return value
  },
  |err| {
    println("Failed!" + err: message())
    return 42
  })

println(toInt("Hello"): orElse(42))
```

Trying

```
module ResultError

import gololang.Errors

function main = |args| {

  # get a Result or an Error
  trying({
    return Integer.parseInt("Quarante-deux")
  })
  : either(
    |value| {
      println("Succeed!")
    },
    |err| {
      println("Failed!" + err: message())
    })

}
```

Golo ❤ Java

part 1

```
package acme;

import java.lang.String;
import java.lang.System;

public class Toon {

  public String name;
  private String nickName;

  public String getNickName() { return nickName; }
  public void setNickName(String value) { nickName = value; }

  public Toon(String name) { this.name = name; }

  public Toon() { this.name = "John Doe”; }

  public void hi() { System.out.println("Hi, I'm " + this.name); }

  public static Toon getInstance(String name) { return new Toon(name); }

  public void hug(Toon toon) { System.out.println("I 💙 " + toon.name); }
}
```

```
module toons

import acme

function main = |args| {


}
```

```
module toons

import acme

function main = |args| {
  let buster = Toon("Buster") # no new
}
```

```
module toons

import acme

function main = |args| {
  let buster = Toon("Buster") # no new
  buster: hi()


}
```

```
module toons

import acme

function main = |args| {
  let buster = Toon("Buster") # no new
  buster: hi()

  buster: nickName("busty")

  println(buster: nickName())



}
```

```
module toons

import acme

function main = |args| {
  let buster = Toon("Buster") # no new
  buster: hi()

  buster: nickName("busty")

  println(buster: nickName())

  let elmira = Toon.getInstance("elmira")



}
```

```
module toons

import acme

function main = |args| {
  let buster = Toon("Buster") # no new
  buster: hi()

  buster: nickName("busty")

  println(buster: nickName())

  let elmira = Toon.getInstance("elmira")
  println(elmira: name())
  elmira: name("Elmira")
  println(elmira: name())
  elmira: hug(buster)
}
```

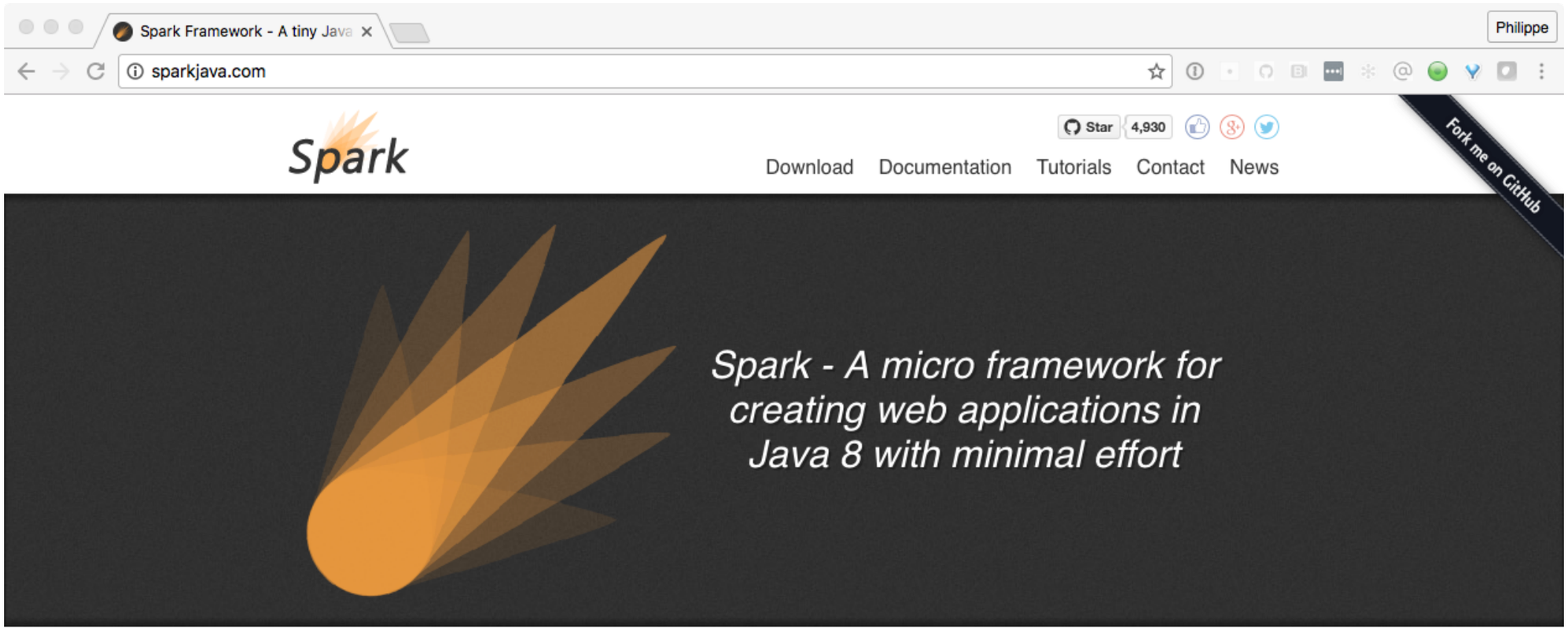
Augmentations again!

```
module toons2

import acme

augment acme.Toon {
  function hi = |this, message| {

  }
}

function main = |args| {
  let buster = Toon("Buster") # no new
  buster: hi("where is babs")
}
```

```
module toons2

import acme

augment acme.Toon {
  function hi = |this, message| {
    this: hi()
    println("my message: " + message)
  }
}

function main = |args| {
  let buster = Toon("Buster") # no new
  buster: hi("where is babs")
}
```

Golo ❤ Java

part 2



Quick Start

```
import static spark.Spark.*;

public class HelloWorld {
    public static void main(String[] args) {
        get("/hello", (req, res) -> "Hello World");
    }
}
```

Run and View

```
http://localhost:4567/hello
```

Built for Productivity

Spark Framework is a simple and lightweight Java web framework built for rapid development. Spark was originally inspired by the web framework Sinatra, but it's intention isn't to compete with Sinatra, or other similar web frameworks in different languages. Sparks intention is to provide a pure Java alternative for developers that

```
import spark.Spark

function main = |args| {

  setPort(8888)
  externalStaticFileLocation("public")

  get("/hi", |request, response| {
    response: type("text/plain")
    return "Hi :earth_africa: at JDD 2016"
  })

  get("/hello", |request, response| {
    response: type("application/json")
    return JSON.stringify(
      DynamicObject(): message("Hello :earth_africa: at JDD 2016")
    )
  })
}
```

```
import spark.Spark

augment spark.Response {
  function jsonPayLoad = |this, content| {
    this: type("application/json")
    return JSON.stringify(content)
  }
  function textPayLoad = |this, content| {
    this: type("text/plain")
    return content
  }
}
```

```
function main = |args| {

  setPort(8888)
  externalStaticFileLocation("public")

  get("/hi", |request, response| ->
    response: textPayLoad("Hi :earth_africa: at JDD 2016")
  )

  get("/hello", |request, response| ->
    response: jsonPayLoad(
      DynamicObject(): message("Hello :earth_africa: at JDD 2016")
    )
  )
}
```

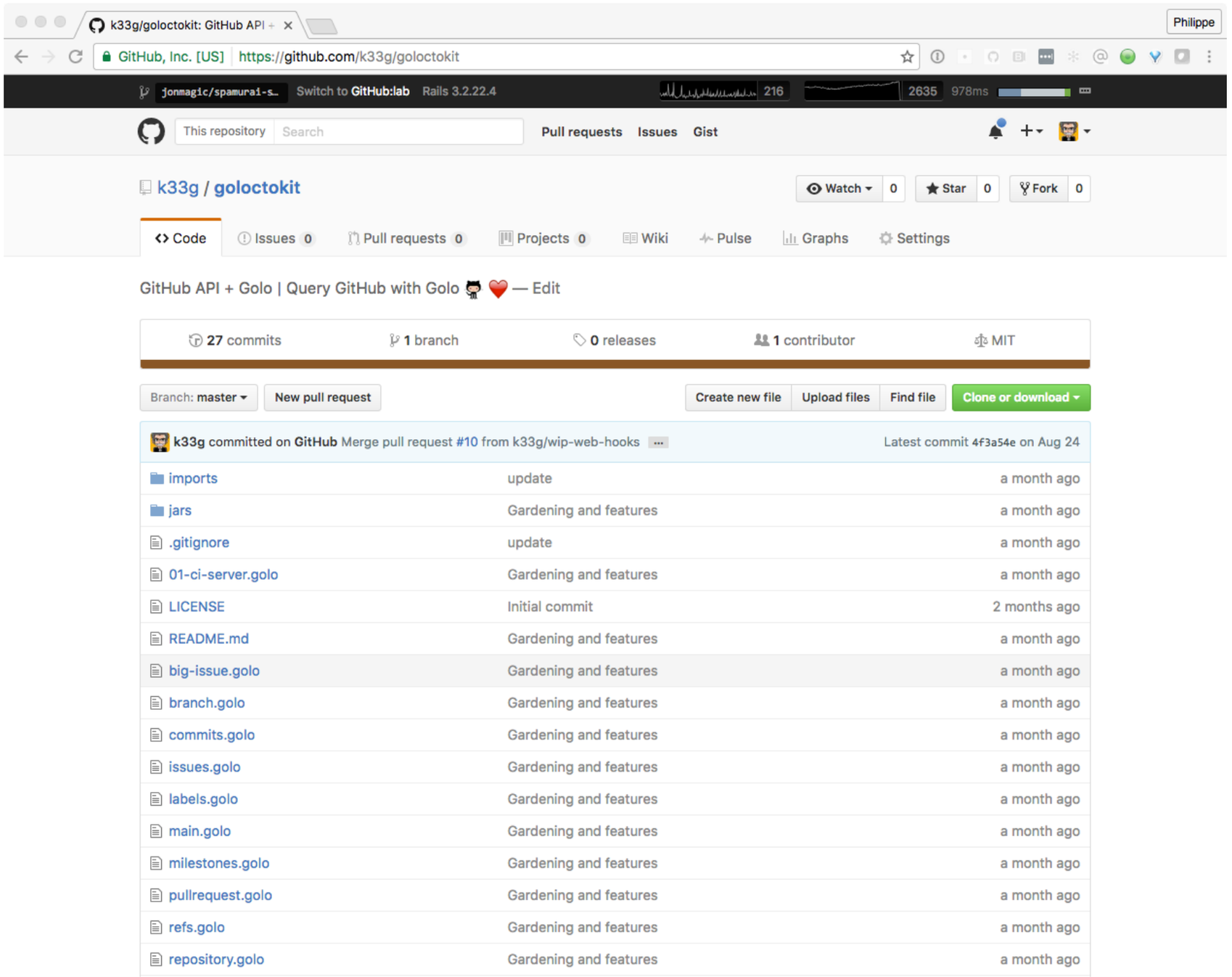
a more functional way

```
get("/hi", |request, response| -> trying({
  return "Hi :earth_africa: at JDD 2016"
})
: either(
  |message| -> response: textPayLoad(message),
  |error| -> response: textPayLoad(error: message())
))
```

```
get("/hello", |request, response| -> trying({
  #let r = 42 / 0
  return DynamicObject(): message("Hello :earth_africa: at JDD 2016")
})
: either(
  |content| -> response: jsonPayLoad(content),
  |error| -> response: jsonPayLoad(map[["message", error: message()]])
))
```

see it in action

I use Golo everyday (almost) at work


k33g/goloctokit: GitHub API
Philippe
GitHub, Inc. [US] https://github.com/k33g/goloctokit
jonmagic/spamurai-s… Switch to GitHub:lab Rails 3.2.22.4 216 2635 978ms
This repository Search Pull requests Issues Gist
k33g / goloctokit
Watch 0 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings
GitHub API + Golo | Query GitHub with Golo ❤️ — Edit
27 commits 1 branch 0 releases 1 contributor MIT
Branch: master New pull request Create new file Upload files Find file Clone or download
k33g committed on GitHub Merge pull request #10 from k33g/wip-web-hooks Latest commit 4f3a54e on Aug 24
imports update a month ago
jars Gardening and features a month ago
.gitignore update a month ago
01-ci-server.golo Gardening and features a month ago
LICENSE Initial commit 2 months ago
README.md Gardening and features a month ago
big-issue.golo Gardening and features a month ago
branch.golo Gardening and features a month ago
commits.golo Gardening and features a month ago
issues.golo Gardening and features a month ago
labels.golo Gardening and features a month ago
main.golo Gardening and features a month ago
milestones.golo Gardening and features a month ago
pullrequest.golo Gardening and features a month ago
refs.golo Gardening and features a month ago
repository.golo Gardening and features a month ago

setup a demo & faking a CI server

Hacking Golo

Demo

What else?

quickly

Decorators: kind of annotations

Workers: a little bit like JavaScript workers

Promises: asynchronous code

Observables: with a change event

Dynamic evaluation of code

and a lot of other things...

to conclude

http://golo-lang.org/

https://github.com/eclipse/golo-lang

https://github.com/k33g/golo-jdd-2016/issues

Thank you & Thank you to JDD 😍

–

Questions? (speak slowly)