Crystal: ràpid com *C*, elegant com *Ruby*

Hash: 4165

Primer Quatrimestre Curs 2019/20

Índex

Història de Crystal

Crystal és un llenguatge de programació de carácter general, orientat a objectes, Turing complet i *cross-platform* que neix l'any 2011, inicialment anomenat *Joy*, creat per Ary Borenszweig, Juan Wajnerman i Brian Cardiff de Manas Technology Solutions.

El llençament oficial va ser al juny de 2014 i ho va fer com a *FOSS* (*Free and open-source software*). Des d'aleshores continua en desenvolupament amb l'ajuda de més 300 contribuidors i això provoca que encara puguin venir canvis significatius en el llenguatge. Cal remarcar que al juny de 2016 va entrar al *TIBOE index* i actualment es troba entre la posició 50 i 100 del rànquing.

Propòsits del llenguatge

Quan neix l'idea de *Crystal* i s'inicia el seu disseny, els seus creadors ho fan amb el següents objectius en ment:

- Una sintaxi similar a la de *Ruby* per la seva eficiència a l'hora d'escriure codi
- Una eficiència i rapidesa per executar codi com la de *C*
- Permetre cridar codi en *C* des de *Crystal* amb *bindigs*
- Orientació a objectes completa
- Comprovació de tipus estàtica sense haver d'especificar el tipus de les variables o dels arguments de funcions, com diuen els propis desenvolupadors: "volem que el compilador ens entengui sense haver d'especificar tipus a tots llocs"
- Generació de codi i evaluació en temps de compilació per evitar el *boilerplate code*
- Compilació a codi natiu eficient

Paradigmes de programació

Crystal és un llenguatge multiparadigma signficiant que es regeix per més d'un paradigma de programació. És clarament un llenguatge amb Programació Orientada a Objectes (en anglés *Object Oriented Programming* o *OOP*), imperatiu i amb alguna característica funcional.

Orientat a objectes

A *Crystal* tot són objectes i per tant permet moltes de les característiques de *OOP* com ara la creació de classes incloent polimorfisme i herència però no herència multiple.

El nom de la classe, i de fet de qualsevol tipus, s'escriu amb majuscula i els atributs de la classe es caracteritzen per anar precedits per un `@`. Una subclasse hereda tot de la superclase: atributs, mètodes i la constructora.

A més es permet fer *overloading* dels mètodes de la classe i les classes poden ser `abstract`. Com a molts altres llenguatges, *Crystal* defineix tres nivells de visibilitat pels mètodes d'una classe: per defecte publics, `private` i `protected`.

En el següent exemple veiem com funciona el polimorfisme, herència i les classes `abstract` a *Crystal*:

```
abstract class Animal
  abstract def talk
end

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

class Cat < Animal
  def talk
    "Miau"
  end
end

class Person
  getter pet

  def initialize(@name : String, @pet : Animal)
  end
end

john = Person.new "John", Dog.new
peter = Person.new "Peter", Cat.new
```

Podem també definir atributs i mètodes de classe que a diferencia dels atributs i mètodes normals d'una classe s'associen a la classe en si i no a una instancia. En el cas dels atributs es declaren precedits de `@@` i pels mètodes indiquem que són de classe precedint el nom del mètode amb el nom de la classe i un punt.

Les variables de classe també s'hereden però per cada classe/subclasse té un valor diferent.

Exemple de variables de classes:

```
class Counter
  @@instances = 0

  def initialize
    @@instances += 1
  end

  def self.instances
    @@instances
  end
end

```

```
13 Counter.instances #=> 0
14 Counter.new
15 Counter.new
16 Counter.new
17 Counter.instances #=> 3
```

Imperatiu

És un llenguatge que adopta la programació imperativa, és a dir li diu al ordinador "com" i no "què" ha de fer, i com a tal té noció d'estat, canvis d'estat, etc.

Funcional

Apart de *OOP* i imperatiu *Crystal* té algunes característiques del paradigma funcional com ara les *closures*. Una *closure* la podem definir com una funció que recorda el seu àmbit lèxic inclús fora d'ell. Aquí un exemple de com fer-ho amb *Crystal*:

```
1 def counter
2   x = 0
3   ->{ x += 1; x }
4 end
5
6 proc = counter
7 proc.call #=> 1
8 proc.call #=> 2
```

En el exemple anterior tot i que `x` és una variable local dins de `counter`, en aquest cas la variable és guarda a la *heap* i no al *stack* i per tant és accessible per `proc`.

Sistema de tipus

Crystal funciona amb un tipat estàtic, que vol dir que el tipus d'una variable queda definit en temps de compilació i no d'execució. Això no només ens permet optimitzar més el codi sinó que també evita tots els possible errors i *bugs* deguts als tipus de les variables, com per exemple *null pointer exception*. Però encara hi ha més: *Crystal* implementa un sistema d'inferència de tipus. Així doncs ens dona el millor del cada món: tipat estàtic com a *Java* o *C++* i inferència de tipus com a *Ruby* o *Python*.

Els tipus estàndars definits per *Crystal* són els esperats i presents a qualsevol llenguatge de programació: *Nil*, *Bool*, *Integer*, *Float*, *Char*, *String*, *Symbol*, *Array*, *Hash*, *Tuple*, etc.

Inferencia de tipus

Complint un dels principals objectius que es van marcar els creadors del llenguatge, *Crystal* té un sistema d'inferència de tipus durant la compilació que ens permet no tenir que especificar un tipus per cada variable. Si bé és cert que esta permés assignar tipus i de fet és recomanable i necessari fer-ho en situacions ambigües, hi ha situacions en les que podem no fer-ho i el llenguatge ja ho inferirà.

Aquest sistema es regeix per bastantes regles però hi ha tres que són les principals i més utilitzades. La primera és l'assignació d'un tipus literal. En el exemple següent s'inferirà que `@name` és `String` i `@age` és `Int32`.

```
class Person
  def initialize
    @name = "John Doe"
    @age = 0
  end
end
```

En segon hi ha l'assignació del resultat de crear una instancia d'una classe, per exemple `@address` serà clarament del tipus `Address` i per tant no cal especificar-ho.

```
class Person
  def initialize
    @address = Address.new("somewhere")
  end
end
```

Finalment, la tercera de les regles més importants és l'assignació d'una variable que és un argument d'un mètode amb un tipus definit. En el cas que se'ns presenta aquí, `@name` serà del tipus `String` ja que se li assigna el valor de la variable `name` que es del tipus `String`.

```
class Person
  def initialize(name : String)
    @name = name
  end
end
```

Union type

Union type consisteix en que una variable pot ser de diversos tipus alhora. Aquí en tenim un exemple:

```
if 1 + 2 == 3
  a = 1
else
  a = "hello"
end
```

Al acabar el `if` la variable `a` tindrà tipus `Int32|String` és a dir la unió de `String` i `Int32` i és el compilador el que genera aquest *union type*.

Sistema d'execució

Aquest és un llenguatge compilat i ho fa en codi natiu per ser més eficient. Per compilar i executar programes escrits en *Crystal* tenim diverses formes de fer-ho depenent del que volem. A més tenim altres comandes molt utils com `crystal init` que inicialitza un projecte de *Crystal* o `crystal docs` que genera la documentació del codi basant-se en els comentaris del codi.

Cal remarcar que des de novembre de 2013 *Crystal* és *self-hosting*, és a dir que el compilador ha estat escrit en el propi llenguatge.

Com a exemple utilitzarem un programa anomenat `hello_world.cr` que simplement imprimeix per pantalla "Hello world!".

Crystal run

La comanda `crystal run` seguida del nom d'un fitxer programat en *Crystal* (amb extensió .cr) compila el programa a un executable binari en una localització temporal i l'executa.

Per exemple:

```
$ crystal run hello_world.cr

Hello world!
```

Crystal build

`Crystal build` simplement compila el programa i genera l'executable amb el mateix nom però sense l'extensió. Després, per poder executar el programa fem com amb qualsevol altre executable i el cridem precedit de `./`.

Utilitzant `helloworld.cr`:

```
$ crystal build hello_world.cr

$ ./hello_world

Hello world!
```

Per defecte la compilació no està optimitzada i per activar les optimitzacions cal afegir la *flag* `--release` quan compilem. És recomanable fer-ho només quan volem executar *benchmarks* o quan generem executables per distribuir.

Principals aplicacions

Crystal és un llenguatge de caràcter general i a més permet utilitzar totes les llibreries de *C* a través dels *bindings* proporcionats per *Crystal* per tant es pot utilitzar, i ja és utilitzat, per infinitat de projectes i aplicacions en àmbits molt variats.

Algunes de les companyies que utilitzen *Crystal*:

- **NeuraLegion** - un *fuzzer* avançat amb *machine learning*
- **Level UP Solutions** - desenvilupament web especialitzat en *e-commerce*
- **Diploid** - interpretació del genoma humà
- **Nikola Corp** - camions que funcionen amb hidrogen
- **MeasureChina** - predicció de tendències de *e-commerce* a China amb inteligència artificial
- **BlockVue** - tours en realitat virtual d'edifics en lloguer

LP similars

Un dels llenguatges de programació més similars a *Crystal* és ***Ruby*** i és d'esperar si tenim en compte que un dels objectius dels creadors de *Crystal* era tenir una sintàxis molt semblant a *Ruby*. També s'hi assembla força ***Python*** ja que cap de les dues necessita especificació de tipus de les variables i tenen una sintaxi semblant. D'altra banda ***C/C++*** s'hi assemblen també en que tenen tipat estàtic.

Descripció de les fonts d'informació

Les principal font d'informació emprada en aquest treball ha estat la documentació del propi llenguatge trobada a internet. Si bé és cert que és un llengutge que té pocs anys i no molt popular i per tant no hi han disponibles gaires fonts d'informació; també és cert que la documentació del llenguatge és força completa i fàcil d'entendre. Pel que respecte a la qualitat d'aquestes fonts, proporcionades directament pels creadors del llenguatge, són inmillorables. També he trobat alguns articles que remarcaven les qualitats de *Crystal*, un d'ells escrit per Sam Johnson, *CTO* a BlockVue, una de les companyies que han adoptat *Crystal* al proces de desenvolupament dels seus productes. Per tant la coneixença del llenguatge per part de Johnson és molt bona. L'altre article que he consultat està escrit per un enginyer de software i tot i que potser conneix bé el llenguatge no hi ha res que ens ho demostri així que la qualitat d'aquesta font és regular. L'última font usada per aquest treball és un repositori de *Github* anomenat *Crystal by example* en el que a través d'exemples simples de codi t'explica com funciona *Crystal*. M'ha resultat molt útil per entendre alguns aspectes del llenguatge.

Bibliografia

- Documentació *Crystal*

  Source: https://github.com/crystal-lang/crystal

- Web de *Crystal*

  Source: https://crystal-lang.org/

- Article a la Viquipèdia de *Crystal*

  Source: https://en.wikipedia.org/wiki/Crystal_(programming_language)

- Aşkın Gedik, *Crystal by example*

  Source: https://github.com/askn/crystal-by-example

- Sam Johnson, *Why Crystal is the most promising programming language of 2018*

  Source: https://medium.com/@DuroSoft/why-crystal-is-the-most-promising-programming-language-of-2018-aad669d8344f

- Stanislav Kozlovski, *Crystal — the Ruby you've never heard of*

  Source: https://hackernoon.com/crystal-the-ruby-youve-never-heard-of-57bad2efac9c