# De essentie van Ruby.

Remco Hobo

2004/2005

**Abstract**

In dit document zal ik uiteen zetten wat de essentie van Ruby is. De geschiedenis van Ruby zal behandeld worden, Roby's belangrijkste features. Ook zal het Metamodel worden behandeld en zal er worden gekeken naar de features voor Distributed Computing met Ruby. Dit document bevat de essentie van Ruby vanuit het oogpunt van de schrijver

## De geschiedenis van Ruby

Ruby is ontstaan op 24 februari 2004. De ontwikkelaar van Ruby had het niet hoog op met talen als Perl4 en Python. Python was volgens hem niet strict genoeg OO en Perl4 voelde volgens hem te speelgoedachtig aan. Ruby 1.0 is vrijgegeven in december 1996, 1.1 in augustus 1997, 1.2 (stable) en 1.3(development) zijn in december 1998 vrijgegeven. Op het moment van dit schrijven is de nieuwste stabiele versie 1.8.1 en is er een 3e preview van 1.8.2 uitgekomen. Het doel van Ruby is een taal te zijn die makkelijk te gebruiken is en die compleet OO geprogrammeerd is.

## Features van Ruby.

De Ruby code is niet gecompiled zoals bij talen als C, C++. Er wordt een syntax boom gebouwd waar doorheen gelopen wordt. In toekomstige versies van Ruby wordt gepoogd dichter naar een gecompilede taal toe te gaan.

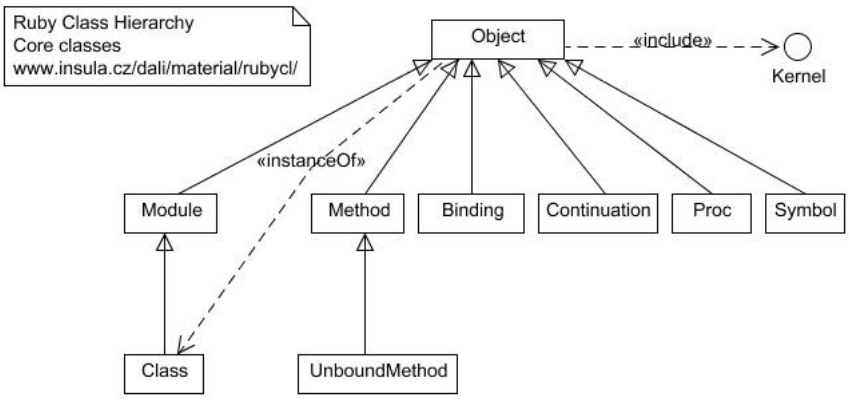
- Ruby heeft een simpele syntax, geinspireerd op talen als Eiffel en Ada.
- Ruby heeft exception handling features zoals Java of Python, wat debuggen vereenvoudigd.
- Operators kunnen makkelijk worden herdefinird.
- De Ruby taal is volledig OO, alles is een object. Zelfs het nummer 1 is een instantie van de klasse Fixnum.
- Roby's OO is ontworpen om compleet te zijn, maar desondanks toch makkelijk aan te passen; het is erg gemakkelijk om een methode toe te voegen aan een klasse of zelfs aan een instantie tijdens runtime.

- Ruby heeft opzettelijk alleen enkele overerving. Dit inverband met de eenvoud hiervan; het is veel makkelijker om een module te importeren met al zijn methodes.
- Ruby ondersteund blokken in de syntax (code omgeven door '{' .... '}' of 'do' ... 'end' ).
- Roby heeft een "mark-and-sweep" garbage collector; ongebruikte code wordt door Ruby zelf vernietigd.
- Er wordt automatisch getypecast tussen kleine integers (Fixnum) en grote integers (Bignum). Dit maakt het leven aangenamer.
- Ruby heeft geen variabele declaraties nodig. Een 'var' is een locale variabele, een '@var' is een instantie variabele, een '$var' is een globale variabele. Het is dus niet nodig om telkens 'self' te gebruiken voor elk lid van de instantie.
- Ruby kan dynamische extentie bibliotheken laden als het OS dit toestaat.
- Roby heeft OS-onafhankelijke threading. Multithreading werkt dus onafhankelijk van het OS. Zelfs in MS-Dos werkt het.
- Ruby is platform-onafhankelijk. Het wordt het meest gebruikt onder Linux, maar het werkt ook op veel versies van UNIX, DOS, Windows 95/98/Me/NT/2000/XP, MacOS, BeOS, OS/2, enz.

## Metamodel

Het meta model van Ruby zal in onderstaand deel, in hoofdonderdelen worden uiteengezet.

- Object is de hoofdklasse van Ruby, alle methodes in deze klasse zijn beschikbaar voor alle objecten. De object module is een kernel module, wat zijn functionaliteit globaal maakt.
- Module is een verzameling van methodes en constanten. Methodes in een module kunnen instantie methodes of module methodes zijn. Methodes doen zich voor als medhodes in een klasse wanneer de module is included, module methodes doen dit niet. .
- Objecten van de klasse Binding bevatten de context van een specifieke code. Binding objecten kunnen later gebruikt worden als referentie naar deze code op een later moment.
- Continuation objecten worden gegenereerd door Kernel#callcc, deze bevatten het return address en de uitvoer context. Hierdoor wordt niet-lokale return naar het eind van callcc blok mogelijk gemaakt vanaf elke plek in de code.
- Proc obejecten zijn blokken code, die zijn gebonden aan een set van lokale variabelen.




- Symbol obecten representeren een Ruby naam welke is gegenereerd door de :name literal syntax.

## Vergelijking met andere talen

Python en Roby zijn bijde object georinteerde talen met een simpele overgang tussen procedures programmeren en OO programmeren. Een taal als Smalltalk is bijvoorbeeld alleen OO, je kan er niets mee totdat je begrippen als objecten, overerving kent en begrijpt. Dit is waarom een taal als Smalltalk nooit het grote publiek heeft geraakt.

Python is een hybride taal, het heeft functies om procedureel te kunnen programmeren en objecten om OO te kunnen programmeren. Python overbrugd deze twee werelden door voor elke functie of methode 'self' te plaatsen, hiermee wijzend naar het eigen object. Het eerste argument wordt hierdoor automagisch een referentie voor de ontvanger.

Ruby is puur OO, dat zich kan voordoen als een procedure taal. Het heeft geen functies alleen methode aanroepen. Net als in Python krijgen alle methodes voor de ontvanger een 'self' maar alle functies die worden aangeroepen zijn eigenlijk default methodes van een object.

Ruby gebruikt, net als Smalltalk, blokken met syntax, iets waar Python geen raad mee weet.

Hieronder staat een vergelijkingstabel met andere talen:

## Conclusie

Ruby is ontwikkeld als verbeterde opvolger voor Python. De Ruby taal is een taal die uitermate geschikt is om OO in te programmeren. Dit omdat de taal zelf 100% OO geprogrammeerd is. Ruby is een taal die uitermate gebruiksvriendelijk is. Doordat Ruby uitermate 'portable' is, kan het op zowat elk platform draaien. Ook heeft Ruby de mogelijkheid om simpel over het netwerk te communiceren met andere Ruby applicaties.

# References

[1] Distributed Systems Concepts and Design, Addison Wesley, ISBN: 0-201-61918-0

[2] Distributed System principles, Wolfgang Emmerich , 1997, `http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/ds98-99/dsee3.pdf`