Nim

Workshop: Programming Languages

Niklas Peter

Universität Siegen

5. Februar 2024

Inhalt

Allgemeines

- Andreas Rumpf startete 2005 die Entwicklung
- Veröffentlichung in 2008 unter MIT Lizenz
- Version 1.0 kam am 23.09.2019 raus
- Soll effizient, ausdrucksvoll und elegant sein
- Generiert native ausführbare Programme ohne Abhängigkeiten
- Unterstützt BSD, Linux, macOS und Windows

Design

- Syntax inspiriert von Python
  - Kennzeichnung von Codeblöcken durch Einrückung statt Klammern
  - Schlüsselwörter (and, not, elif, ...)
  - Kein Semikolon nach Statements
- Style-Insensitive für Mischung verschiedener Stile: *ein_beispiel* und *einBeispiel* sind gleich (nur erster Buchstabe muss übereinstimmen)
- Statische Typisierung aber mit einfacher Typumwandlung
- Uniform Function Call Syntax (UFCS)
  - Funktionsaufruf durch Methoden-Syntax
  - Receiver wird erster Parameter
  - $x = multiply(y, z)$ entspricht $x = y.multiply(z)$

- Programmierungsparadigmen
  - Funktionale Programmierung
  - Objektorientierte Programmierung
    - In erster Linie imperative und funktionale Sprache
    - Grundlegende Elemente wie z.B. Polymorphie und Vererbung
  - Metaprogrammierung
    - Templates
    - Macros
    - Generics
  - Foreign Function Interface (FFI)
    - Zugriff auf in anderen Sprachen geschriebene Bibliotheken
  - Parallelisierung und Nebenläufigkeit
    - Funktionen müssen gekennzeichnet werden
    - Kompilierer benötigt *-threads:on* Option
    - Gewisse Regeln wie Einschränkung der Speicherzugriffe

```
# Fibonacci function
proc fib(n: Natural): Natural =
  if n < 2:
    return n
  else:
    return fib(n-1) + fib(n-2)

#Uniform Function Call Syntax example
let numbers = @[1, 2, 3, 4, 5, 4, 3, 2, 1]
echo numbers.filter(x => x > 2).deduplicate.foldr(a + b) # Output: 12

#Foreign Function Interface
proc printf(formatstr: cstring) {.header: "<stdio.h>", importc: "printf", varargs}
```

Figure: Code Beispiel

Infrastruktur

- Kompilierer
  - Vollständig in Nim geschrieben
  - Gibt optimierten C Code aus. Kompilierung zu C++, Objective-C und JavaScript ebenfalls möglich
  - Objektcode durch externen Kompilierer wie GCC
  - Verschiedene Garbage Collectors verfügbar
    - Standardmäßig Garbage Collector mit Referenzzählung
    - Deaktivierung des Garbage Collectors möglich

- Entwicklungswerkzeuge (Auswahl)
  - Nim's Paketmanager Nimble
  - Nimsuggest zur Integration in IDEs
  - Hot code reloading zum Neuladen des Codes zur Laufzeit
- Bibliotheken
  - Pure und Impure Bibliotheken
  - Standardbibliothek mit grundlegenden Funktionen vollständig in Nim geschrieben
  - Einbindung von Bibliotheken in C, C++, Objective-C und JavaScript möglich
  - Kein zentrales Repository; jeder kann eigenes hosten

Anwendungsfälle

Viele Einsatzmöglichkeiten durch Kompilierung nach C, C++ und JavaScript:

- Shell Scripting
- Front- und Backend für Webdienste
- Machine Learning
- Eingebettete Systeme

Prominentes Beispiel ist **nitter**, ein Frontend für X (Twitter)

Stärken und Schwächen

- Einfache und flexible Syntax
- Moderne Konzepte
- Viele Anwendungsbereiche
- Nicht für alles eine native Bibliothek vorhanden, dafür aber Import von Bibliotheken in anderen Sprachen möglich
- Gute Dokumentation
  - Ausführliche Dokumentation der Standardbibliothek
  - Zahlreiche Tutorials für Neulinge
  - Aktive Community in Foren und Chats

Implementierung

- Nim war vorher unbekannt
- Einarbeitung durch Vorkenntnisse von Python recht einfach
- Großer Vorteil: Trotz Python's Syntax keine interpretierte Sprache, daher sehr schnell und speichereffizient

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?