Programowanie w Ruby

Wykład 7

Marcin Młotkowski

2 grudnia 2016

Plan wykładu

Refleksje (introspekcje)

- Przegląd zaimplementowanych metod
- Przegląd stanu obiektu
- I inne ...

Analiza stanu obiektu

Metoda inspect

```
[ 1, 2, 3..4, 'five' ].inspect
```

Analiza stanu obiektu

Metoda inspect

```
[ 1, 2, 3..4, 'five' ].inspect
```

wynik

```
[1, 2, 3..4, \"five\"]
```

Zmienne obiektu

```
class Klasa
    def initialize
        @a = 1; @b = 2
    end
end
```

Zmienne obiektu

```
class Klasa
    def initialize
        @a = 1; @b = 2
    end
end

Klasa.new.instance_variables
```

Wynik

```
["@a", "@b"]
```

Metody obiektu

3.14.methods

Metody obiektu

3.14.methods

["%", "inspect", "singleton_method_added", "clone", "round", "method", "public_methods", "instance_variable_defined?", "divmod", "nan?", "equal?", "freeze", "integer?", "*", "+", "to_i", "methods", "respond_to?", -", ... "untaint", "is_a?"]

Przykład zastosowania

Zrzut stanu programu

- zrzut stanu wszystkich obiektów
- dane są prezentowane w formie stringu ze znacznikami html

Specyfikacja zadania

- Każdy obiekt powinien mieć metodę to_html, która zwraca stan obiektu w formie html
- Jeśli obiekt nie ma własnej metody to korzystamy z metody to_s

Implementacja

```
html = '<html><body>'
ObjectSpace.each_object(Object) do |o|
    if o.respond_to?('to_html')
        html += o.to_html
    else
        html += '<div>' + o.to_s + '</div>'
    end
end
```

Plan wykładu

Zlecanie pracy innym hostom

- Potrzebny jest serwer, który będzie działał na ustalonym porcie, np. 9000
- Klient powinien znać adres serwera i port

Zadanie serwera

- Serwer logów
- Implementacja metody save(str), która zapisuje str w pliku; nazwa pliku powinna odzwierciedlać czas

Zadanie serwera

- Serwer logów
- Implementacja metody save(str), która zapisuje str w pliku; nazwa pliku powinna odzwierciedlać czas

Rozwiązanie

Drb: Distributed Ruby, distributed object system

Implementacja: serwer drb

```
require "drb"

class LogSerwer
    def save(str)
        fn = Time.new.to_s
        open(fn, 'w') { |f| f << str }
    end

    def LogSerwer.Run
        @@server = LogSerwer.new
        DRb.start_service('druby://localhost:9000', @@server)
        DRb.thread.join
    end
end
```

Uruchomienie serwera

LogSerwer.run

Implementacja klienta

```
obj = DRbObject.new_with_uri('druby://localhost:9000')
if obj.respond_to?('save')
    obj.save(html)
end
```

A jak ktoś nam zrobi kawał

```
def save(str)
    system(str)
end
```

A jak ktoś nam zrobi kawał

```
def save(str)
    system(str)
end
```

```
obj.save('rm -rf *')
```

System bezpieczeństwa

Podział obiektów w Ruby:

- Obiekty zaufane
- Obiekty niezaufane: pochodzące z sieci, plików etc.

Kontrola zaufania

Object#tainted? sprawdzenie czy obiekt jest zaufany

Object#taint ustawienie obiektu na niezaufany

Object#untaint usunięcie 'braku zaufanie' z obiektu (nie zawsze działa)

Poziomy zabezpieczeń

Wartości zmiennej $SAFE

0 brak kontroli (domyślne)
1 m.in. zabrania wywoływań eval dla niepewnych stringów
2 m.in. nie można usuwać i tworzyć katalogów oraz wykonywać pewnych operacji z modułu Kernel
3 nie można wywołać untain
4 nie można modyfikować bezpiecznych obiektów

Problemy

Można obejść mechanizm zabezpieczeń, więc nie należy używać w środowiskach niezaufanych.

Zabezpieczenie

Użycie ACL (Access Control Lists)

Klasa ACL

```
list = %w[
  deny all
  allow 192.168.1.1
  allow ::ffff:192.168.1.2
  allow 192.168.1.3
]

acl = ACL.new(list, ACL::DENY_ALLOW)
```

Zabezpieczenie

Użycie ACL (Access Control Lists)

Klasa ACL

```
list = %w[
  deny all
  allow 192.168.1.1
  allow ::ffff:192.168.1.2
  allow 192.168.1.3
]

acl = ACL.new(list, ACL::DENY_ALLOW)
```

DRb.install_acl(acl)

Plan wykładu

Systemowe punkty zaczepienia

Procedury uruchamiane przy okazji wywołania wybranej funkcji czy metody

Systemowe punkty zaczepienia

Procedury uruchamiane przy okazji wywołania wybranej funkcji czy metody

Przykład punktu zaczepienia

Jakie są wywoływane polecenia systemowe (rm, ls, mkdir etc.) podczas biegu programu?

Realizacja

- Polecenia systemowe są wywoływane funkcją Kernel.system
- Zamiast funkcji Kernel.system wstawimy własną funkcję

Implementacja

```
module Kernel
    alias_method :old_system, :system
    def system(*args)
        puts "system(#{args.join(',')})"
        old_system(args)
    end
end
```

Inspekcja biegu programu

caller – zwraca stos wywołań w postaci tablicy

set_func_proc – pozwala na wywołanie bloku dla każdej instrukcji

global_variables – tablica nazw zmiennych globalnych

local_variables

Śledzenie zmian zmiennych

trace_var(symbol, blok) – śledzi przypisywanie wartości zmiennej

untrace_var(symbol) – usunięcie śledzenia zmiennej

Debugger

```
ruby -r debugger plik.rb
```