

Programowanie w Ruby

Wykład 6

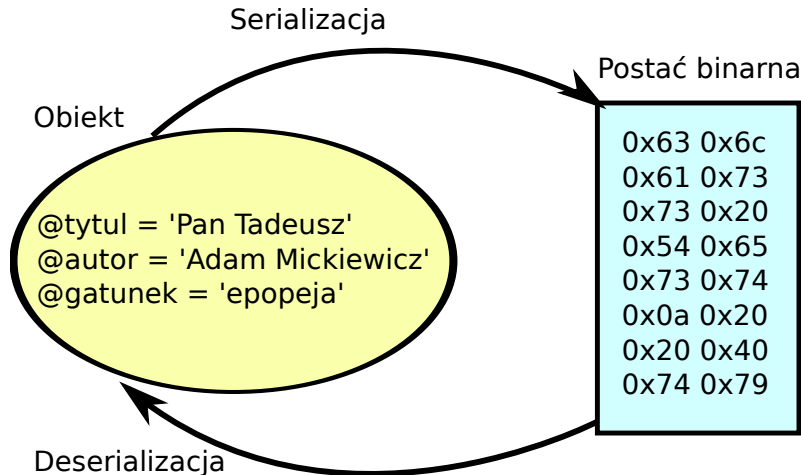
Marcin Młotkowski

19 listopada 2018

Plan wykładu

- 1 Trwałość obiektów
- 2 Bazy danych DBM
- 3 Bazy danych SQL
- 4 Bazy NoSQL
- 5 Active records

Szeregowanie obiektów



Wybrane formaty zapisu

YAML: YAML Ain't Markup Language

- format niezależny od języka
- format: sformatowany tekst

Marshal

- format binarny
- szybki

Stosowanie

Import odpowiedniego modułu

```
require 'yaml'
```

Stosowanie

Import odpowiedniego modułu

```
require 'yaml'
```

Przykład 1.

```
str = YAML.dump([1, 'dwa', 3.0])  
lista = YAML.load(str)
```

Stosowanie

Import odpowiedniego modułu

```
require 'yaml'
```

Przykład 1.

```
str = YAML.dump([1, 'dwa', 3.0])  
lista = YAML.load(str)
```

Przykład 2.

```
open(plik, 'w') { | f | YAML.dump(obiekt, f) }  
obiekt = open(plik, 'r') { | f | YAML.load(f) }
```

Metoda to_yaml

```
[1, "dwa", 3.0].to_yaml
```


Metoda to_yaml

```
[1, "dwa", 3.0].to_yaml
```

```
---  
-1  
- dwa  
- 3.0
```

Marshal

Marshal.dump
Marshal.load

Plan wykładu

- 1 Trwałość obiektów
- 2 Bazy danych DBM
- 3 Bazy danych SQL
- 4 Bazy NoSQL
- 5 Active records

Co to takiego

- DBM: database manager
- Do przechowywania prostych danych
- Każda baza danych to po prostu tablica haszująca
- Przykład: Berkeley DB

Schemat przetwarzania

Import modułu

```
require 'dbm'
```

Schemat przetwarzania

Import modułu

```
require 'dbm'
```

```
DBM.open(@dbm_name) do | db |  
  db[klucz] = wartosc  
  puts db[klucz]  
end
```

Porada

DBM-y można traktować jak kolekcje

Porada

DBM-y można traktować jak kolekcje

```
DBM.open(@dbm_name) do | db |  
  db.each { |k, v| puts "#{k} => #{v}" }  
end
```


Przechowywanie obiektów

```
class Osoba
end

o1 = Osoba.new(...)
o2 = Osoba.new(...)
o3 = Osoba.new(...)

DBM.open("telefon.dbm") do | db |
  db['111'] = o1
  db['222'] = o2
  db['333'] = o3
end
```

Odczyt danych

```
DBM.open("telefon.dbm") do | db |  
  db.each { | k, v | puts "#{k} => #{v.to_s}" }  
end
```

Plan wykładu

- 1 Trwałość obiektów
- 2 Bazy danych DBM
- 3 Bazy danych SQL**
- 4 Bazy NoSQL
- 5 Active records

Mechanizm dostępu do bazy danych

DataBase Interface — DBI

definiuje zbiór funkcji i wyjątków jakie powinny być implementowane przez moduły do obsługi różnych baz danych

Przykład

Sqlite

- Implementuje SQL
- Nie wymaga zewnętrznego serwera (osadzona baza danych)
- Baza jest pamiętana w jednym pliku
- Implementuje DBI

Połączenie z serwerem bazy danych

```
require 'dbi'
```

```
db = DBI.connect("DBI:SQLite3:test.db")
```

Inny przykład: MySQL

```
db = DBI.connect('DBI:Mysql:test:localhost', 'testuser',  
"testpass")
```

Połączenie z serwerem bazy danych

```
require 'dbi'
```

```
db = DBI.connect("DBI:SQLite3:test.db")
```

Inny przykład: MySQL

```
db = DBI.connect('DBI:Mysql:test:localhost', 'testuser',  
"testpass")
```

Zamknięcie połączenia

```
db.disconnect
```

Operacje na otwartej bazie danych

- Operacje niezwracające żadnych danych
- Operacje zwracające listę wierszy
- Operacje zwracające liczbę

Operacje bez wyniku

Utworzenie nowej tabeli

```
db.do("CREATE TABLE Wyklady\  
( Nr INT NOT NULL,\  
Tresc VARCHAR(40) )")
```

Operacje zwracające liczbę

Wstawienie wiersza do tabeli

```
row = @db.do("INSERT INTO Wyklady\  
(Nr, Tresc) VALUES (#{id}, '#{t}')")  
puts ""Wstawiono #{row} wierszy do tabeli"
```

Wyszukiwanie

Przykład 1.

```
rows = db.execute("SELECT * FROM Wyklady")
rows.each do | row |
  puts "#{row[0]}, #{row[1]}"
end
rows.finish
```

Przykład 2.

```
rows = db.execute("SELECT * FROM Wyklady")
rows.each do | row |
  puts "#{row['Nr']}, #{row['Tresc']}"
end
rows.finish
```

Kilka drobnych przykładów

```
db.select_all("SELECT * FROM Wyklady") do | row |  
  puts "#{row[0]}, #{row[1]}"  
end
```

```
db.select_one("SELECT ... WHERE NR=1")
```

Plan wykładu

- 1 Trwałość obiektów
- 2 Bazy danych DBM
- 3 Bazy danych SQL
- 4 Bazy NoSQL**
- 5 Active records

Bazy NoSQL

Bazy danych oparte na modelowaniu danych inne niż relacyjne.

Bazy NoSQL

Bazy danych oparte na modelowaniu danych inne niż relacyjne.

Rodzaje

- obiektowe
- zorientowane na dokumenty
- klucz–wartość
- grafowe

MongoDB

Baza danych zorientowana na dokumenty (JSON).

MongoDB

Baza danych zorientowana na dokumenty (JSON).
Dokumenty są zgromadzone w kolekcje.

Otwarcie i zamknięcie z MongoDB

Otwarcie

```
require 'mongo'
```

```
Mongo::Logger.logger.level = ::Logger::FATAL
```

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database =>  
'testdb')
```

Zamknięcie

```
client.close
```

Utworzenie kolekcji

```
Mongo::Collection.new(client, 'kraje')
```

Dodanie elementów

```
client[:kraje].insert_one {  
  :_id => BSON::ObjectId.new,  
  :nazwa => 'Malta',  
  :powierzchnia => 26  
}
```

```
client[:kraje].insert_one {  
  :_id => BSON::ObjectId.new,  
  :nazwa => 'Tuvalu',  
  :powierzchnia => 316  
}
```

Odczyt

```
client[:kraje].find.each { | kraj | puts kraj }
```

Odczyt

```
client[:kraje].find.each { | kraj | puts kraj }
```

```
client[:kraje].find("powierzchnia" => { '$It' => 150 }).each do | kraj |  
  puts kraj  
end
```

Plan wykładu

- 1 Trwałość obiektów
- 2 Bazy danych DBM
- 3 Bazy danych SQL
- 4 Bazy NoSQL
- 5 Active records

Dygresja: symbole

Symbol

identyfikator odpowiadający łańcuchowi znaków

Tworzenie identyfikatorów

:obiekt

:`'obiekt'`

Odwzorowania obiektowo-relacyjne

```
class Student  
  @imie  
  @nazw  
  ...  
end
```

```
CREATE TABLE stud (  
  imie VARCHAR(30),  
  nazw VARCHAR(40),  
  ...);
```

Zarządzanie odwzorowaniem O-R w Rubym

ActiveRecord

- Zapewnia automatyczne odwzorowanie między obiektami a bazą danych
- Ukrywa wykorzystanie SQL
- Implementuje związki między tabelami
- **Wymaga przestrzegania wielu konwencji**

Użycie ActiveRecord

Zadanie

Zaprogramować bazę danych zawierającą dane o

- studentach
- zadaniach oddanych przez studentów

Rozwiązanie w SQL

Deklaracja tabeli studs

```
'CREATE TABLE studs (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  index VARCHAR (20),  
  punkty INT,  
  PRIMARY KEY(id))'
```

Deklaracja tabeli studs

```
'CREATE TABLE stud_exercises (  
  id INT NOT NULL AUTO_INCREMENT,  
  stud_id INT(11),  
  zad INT,  
  tresc TEXT,  
  PRIMARY KEY(id))'
```

Rozwiązanie obiektowe

Deklaracja klas

```
require 'activerecord'  
  
class Stud < ActiveRecord::Base  
  has_many :stud_exercises  
end  
  
class StudExercise < ActiveRecord::Base  
  belongs_to :stud  
end
```

Active record

Klasa	tabela
Obiekt	wiersz tabeli
Pole obiektu	kolumna

Konwencje

`class stud`

`CREATE TABLE studs`

`class StudExercise`

`CREATE TABLE stud_exercise`

Tworzenie obiektów

```
madry = Stud.create(:index => '10011101', :punkty => 0)  
zadanie = StudExercise.create(:stud_int => madry,  
  :zad => 2, tresc => 'puts 0')
```


Odwołania do elementów bazy danych

Pobieranie danych

```
studs = Stud.find (:all,  
                  :conditions => "index > '100011' ",  
                  :order => 'index DESC')
```

Odwołania do elementów bazy danych

Pobieranie danych

```
studs = Stud.find (:all,  
                  :conditions => "index > '100011' ",  
                  :order => 'index DESC')
```

Modyfikacja danych

```
studs.each do |s|  
  s.punkty = s.punkty + 10  
  s.save  
end
```