

Dokumentation

RDoc vs. YARD

RDoc	YARD
Stdlib	Gem
"Nur" Text	@tags

RDoc

```
##  
# Add water to the water tank (+amount+ in ml/g)  
def add_water(amount)  
  @water += amount  
end
```

add_water (amount)

[Source](#)

Add water to the water tank (amount in ml/g)

YARD

```
##  
# Add coffee to the hopper  
# @param amount [Integer, #read] Amount in grams  
# @return [Integer] Amount of coffee in the hopper after the refill  
def add_coffee(amount)  
  @coffee += amount  
end
```

Instance Method Details

`#add_coffee(amount)` ⇒ Integer

[permalink](#)

Add coffee to the hopper

Parameters:

- **amount** (Integer, #read) — Amount in grams

Returns:

- (Integer) — Amount of coffee in the hopper after the refill



RDoc oder YARD

- Docs:
 - [RDoc](#)
 - [YARD](#)
- Rake Task einbinden
- Methoden dokumentieren

Webanwendungen mit Sinatra

- Simpler Ansatz, Webanwendungen mit Ruby zu entwickeln
- Gegenentwurf zum “schwergewichtigen” Ruby on Rails
- Sehr einfache API, flexibel erweiterbar
- Ideal für einfache Anwendungen und Web-APIs
- `gem install sinatra` / `bundle add sinatra`
- [Website](#)

Sinatra DSL

```
require "sinatra"
get '/' do
  "Willkommen"
end

post '/login' do
  "Willkommen #{params[:user]}"
end
```

Templates

```
get '/now' do
  @now = Time.now
  erb :now
end
```

```
<p>Current time: <%= @now %></p>
```

Layouts

- Layouts sind Templates, die ein `yield` enthalten
- Ideal für üblichen HTML-Rahmen, fixe Header und Footer
- Default: `layout.erb` für ERb-Templates
- Für jedes Template konfigurierbar

Parameter

```
post '/login' do
  @username = params[:username]
  erb :welcome
end
```

Parameter in der URL

```
get '/posts/:id' do |id|  
  @id = id  
  erb :post  
end
```

```
<h3>Post: <%= @id %></h3>
```

Praxis

Webbasierte Kaffeemaschine

Entwickelt eine Webseite, über die sich Eure Kaffeemaschine fernsteuern lässt.

Webbasierte Kaffeemaschine

- Erst mal mit Sinatra spielen
- Als gem einbinden
- Annahme: Webserver läuft in single thread / single process

Concurrency

Ein Beispiel

Quelle: [maikhel](#)

Teil 1: Fetch

```
class DataRequester
  def initialize(url)
    @url = url
  end

  def call
    uri = URI(@url)
    response = Net::HTTP.get_response(uri)
    JSON.parse(response.body)["value"]
  end
end
```

Synchron

```
class SyncService
  API_ENDPOINT = 'https://api.chucknorris.io/jokes/random'.freeze

  def self.call(requests_count)
    jokes = []
    requests_count.times do
      jokes << DataRequester.new(API_ENDPOINT).call
    end

    jokes
  end
end
```

```
class AsyncThreadsService
  API_ENDPOINT = "https://api.chucknorris.io/jokes/random".freeze

  def self.call(requests_count)
    jokes = Queue.new
    threads = []

    requests_count.times do
      thread = Thread.new do
        jokes << DataRequester.new(API_ENDPOINT).call
      end
      threads << thread
    end
    threads.each(&:join)
    jokes
  end
end
```

```
require 'concurrent'
class AsyncConcurrentService
  API_ENDPOINT = "https://api.chucknorris.io/jokes/random".freeze

  def self.call(requests_count)
    # jokes = Concurrent::Array.new
    futures = []

    requests_count.times do
      requester = DataRequester.new(API_ENDPOINT)
      futures << Concurrent::Promises.future { requester.call }
    end
    futures.map(&:value)
  end
end
```

UI / TUI

TUI (Text UI)

- ncurses
- ...
- ratatui

GUI

- tk
- FX (Java)
- wxWindows
- Glimmer

Gaming

- `gosu`
- Dragon Ruby (non-free)



Auswahl

- Glimmer-SWT
- Glimmer LibUI
- Glimmer.*
- ratatui
- gosu

Aufgabe

- Kaffeemaschine?
- Adressdatenbank?
- Pong?

Metaprogramming

Introspektion

- `.methods`
- `.method()`
- `.instance_methods`
- `.respond_to?`

Dynamic dispatch

- `method_missing`

Beispiel: nokogiri builder

```
require 'nokogiri'

builder = Nokogiri::XML::Builder.new(encoding: "UTF-8") do |xml|
  xml.RootElement do
    xml.Test do
      xml.Foo "Hallo"
    end
  end
end

puts builder.to_xml
```

Ruby Golf, die zweite