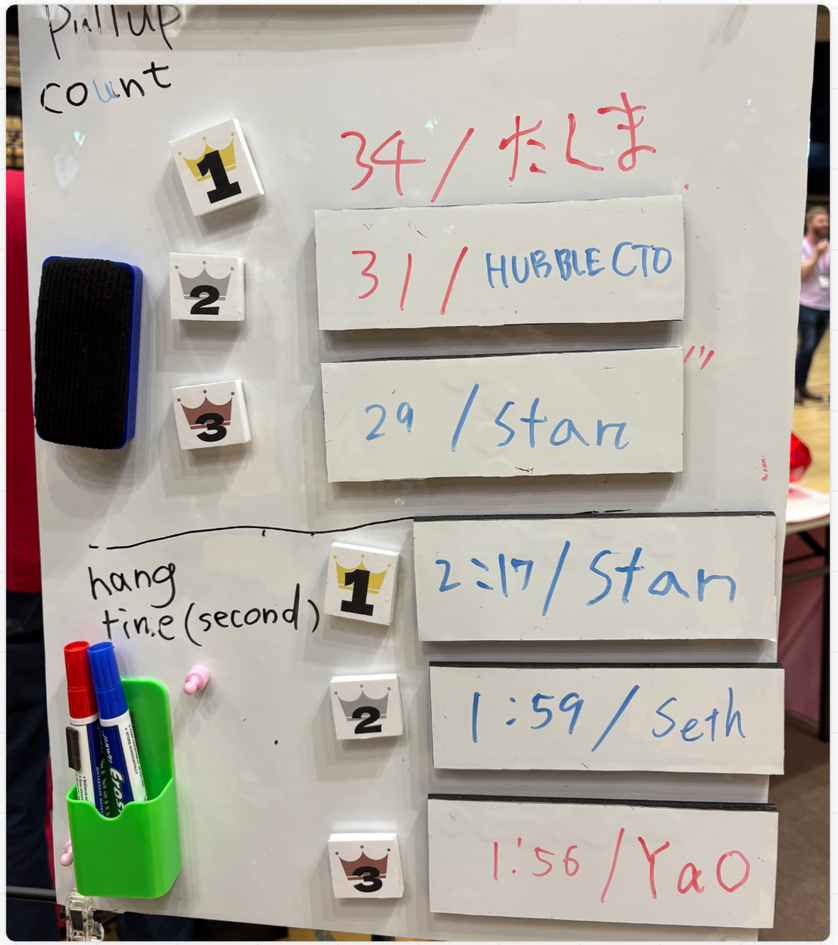
# The Future of Ruby Documentation

Stan Lo

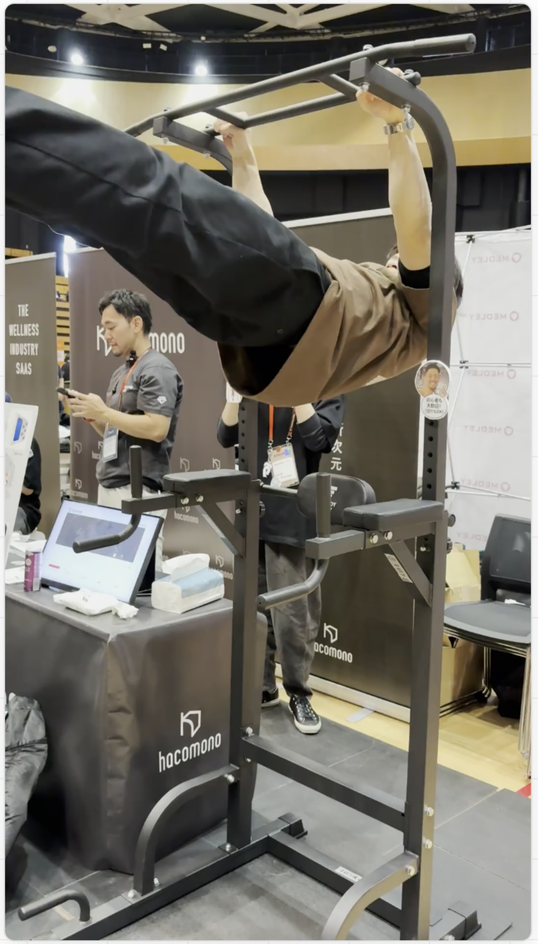
# About me

- Ruby committer — Debug.gem, IRB, RDoc, Reline, a bit of ZJIT

# When I don't write code, I climb and do calisthenics training.



# I also train for tricks like front lever.



# About me

- Ruby committer — Debug.gem, IRB, RDoc, Reline, a bit of ZJIT
- Ruby DX team @Shopify — Tapioca, Ruby LSP, Rubydex

# Rubydex

- A Ruby code indexer — `Shopify/rubydex`
- Static code intelligence for Ruby

EN

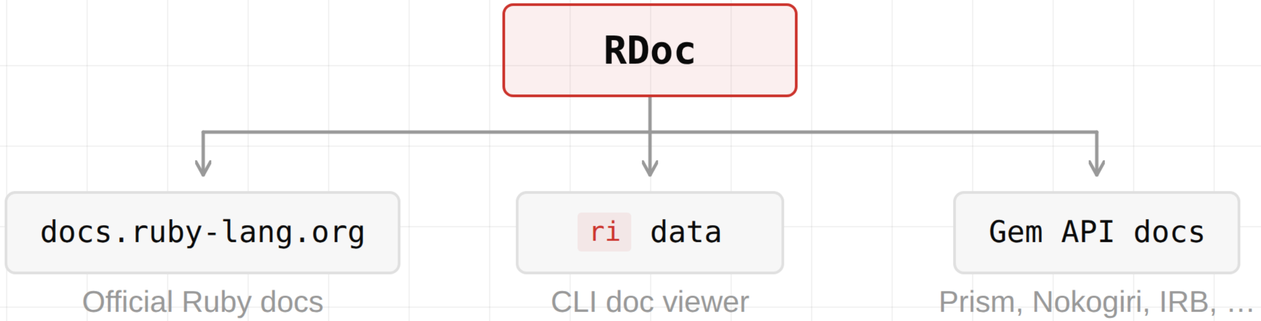
**Blazing-fast Code Indexing for Smarter Ruby Tools**

**Alexandre Terrasa**
@Morriar

# The future of Ruby documentation has to work for AI too.

# And most of this falls on one project: RDoc

The backbone of Ruby documentation for 20+ years.



# But RDoc isn't ready for that yet.

# What AI needs from documentation

- **Clear, accurate documentation**
- **Simpler content formats** — less context overhead (currently Markdown)
- **Clear intent** — structured info like type signatures tell it what code expects

# What AI needs from tooling

- **Quick, deterministic feedback** — coverage checks, reference validation
- **Incremental preview** — see changes without regenerating everything

**Most of these help**
**human developers too.**

# RDoc wasn't delivering

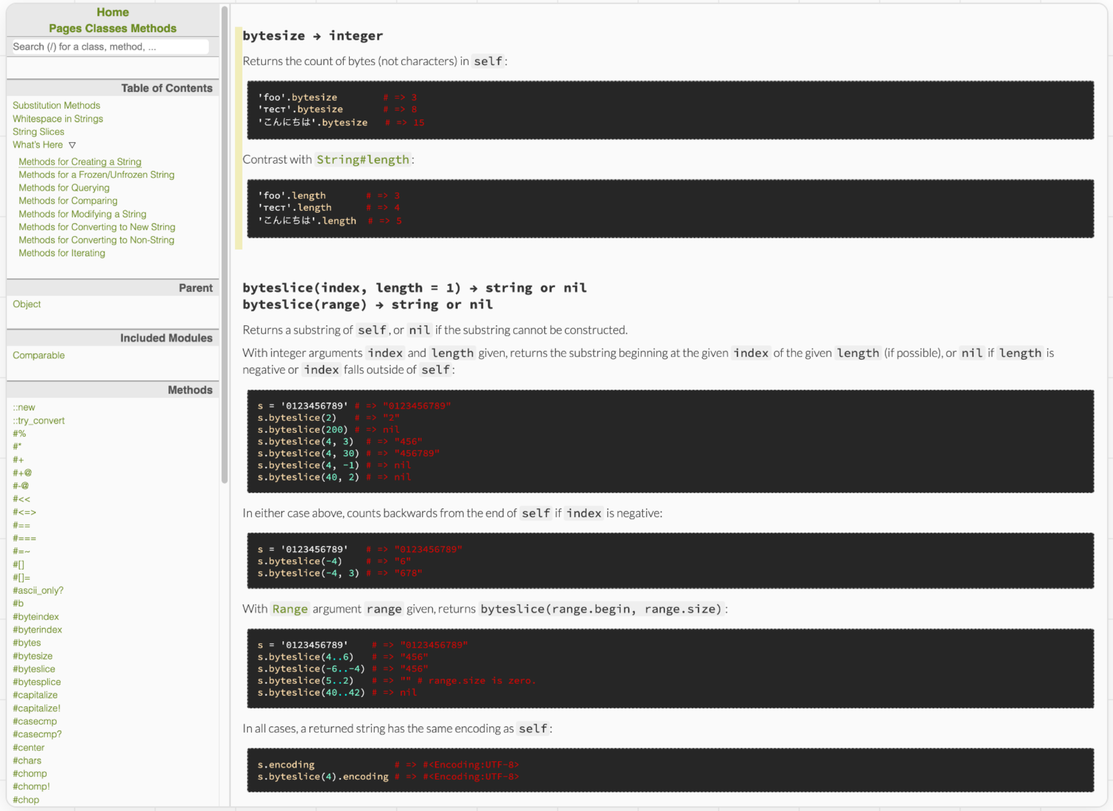
- Parser couldn't handle some newer Ruby syntax
- Incomplete Markdown support
- No RBS type signatures
- No server mode
- Dated theme, bad reading experience

# So here's what we did.

- New theme
- New Prism-based parser
- Markdown improvements
- RBS type signatures
- Server mode

NEW THEME

# Before: Darkfish



Home
Pages Classes Methods

Search (/) for a class, method, ...

Table of Contents

Substitution Methods
Whitespace in Strings
String Slices
What's Here ▽
Methods for Creating a String
Methods for a Frozen/Unfrozen String
Methods for Querying
Methods for Comparing
Methods for Modifying a String
Methods for Converting to New String
Methods for Converting to Non-String
Methods for Iterating

Parent

Object

Included Modules

Comparable

Methods

::new
::try_convert
#%
#*
#+
#+@
#-@
#<<
#<=>
#==
#===
#=~
#[]
#[]=
#ascii_only?
#b
#byteindex
#byterindex
#bytes
#bytesize
#byteslice
#bytesplice
#capitalize
#capitalize!
#casecmp
#casecmp?
#center
#chars
#chomp
#chomp!
#chop

**bytesize → integer**

Returns the count of bytes (not characters) in `self`:

```
'foo'.bytesize        # => 3
'тест'.bytesize       # => 8
'こんにちは'.bytesize   # => 15
```

Contrast with `String#length`:

```
'foo'.length       # => 3
'тест'.length      # => 4
'こんにちは'.length  # => 5
```

**byteslice(index, length = 1) → string or nil**
**byteslice(range) → string or nil**

Returns a substring of `self`, or `nil` if the substring cannot be constructed.

With integer arguments `index` and `length` given, returns the substring beginning at the given `index` of the given `length` (if possible), or `nil` if `length` is negative or `index` falls outside of `self`:

```
s = '0123456789' # => "0123456789"
s.byteslice(2)   # => "2"
s.byteslice(200) # => nil
s.byteslice(4, 3)  # => "456"
s.byteslice(4, 30) # => "456789"
s.byteslice(4, -1) # => nil
s.byteslice(40, 2) # => nil
```

In either case above, counts backwards from the end of `self` if `index` is negative:

```
s = '0123456789'   # => "0123456789"
s.byteslice(-4)    # => "6"
s.byteslice(-4, 3) # => "678"
```

With `Range` argument `range` given, returns `byteslice(range.begin, range.size)`:

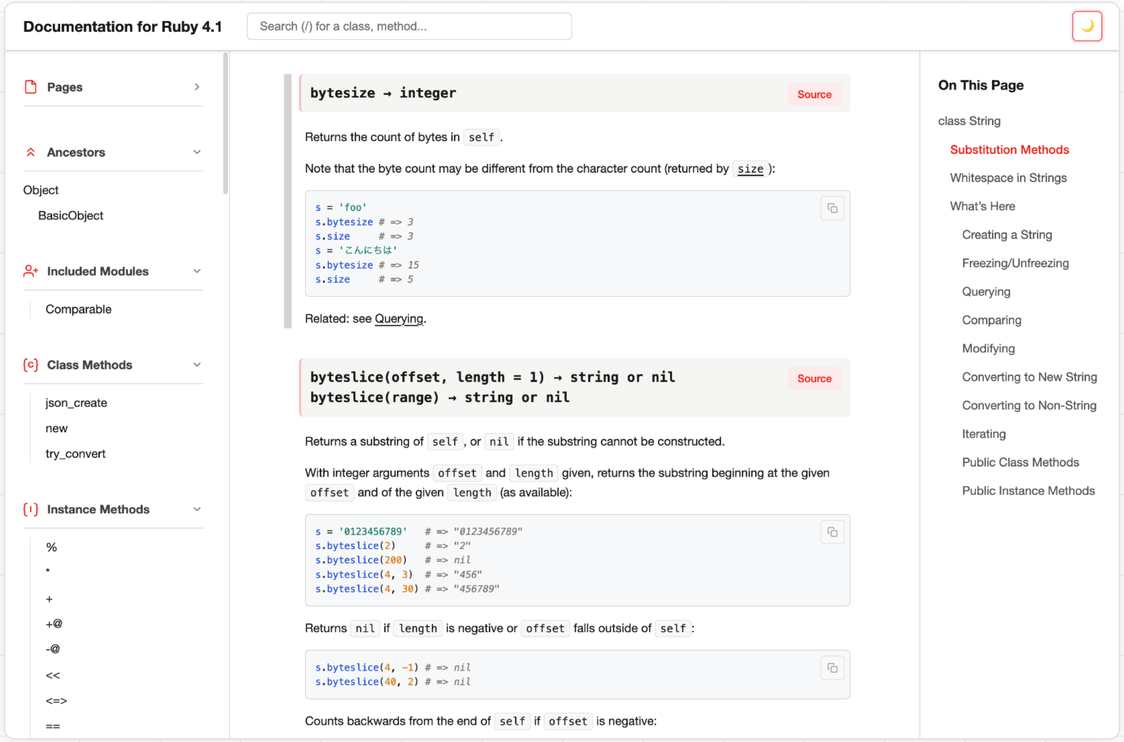
```
s = '0123456789'    # => "0123456789"
s.byteslice(4..6)   # => "456"
s.byteslice(-6..-4) # => "456"
s.byteslice(5..2)   # => "" # range.size is zero.
s.byteslice(40..42) # => nil
```

In all cases, a returned string has the same encoding as `self`:

```
s.encoding              # => #<Encoding:UTF-8>
s.byteslice(4).encoding # => #<Encoding:UTF-8>
```

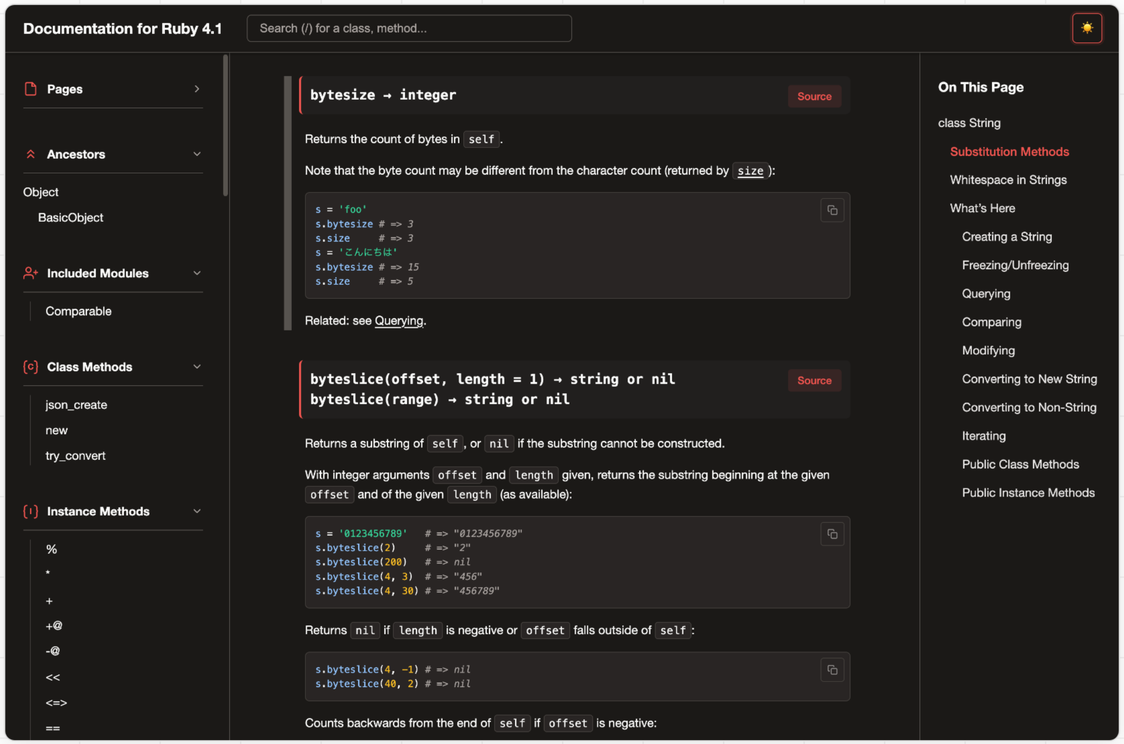
NEW THEME

# After: Aliki



NEW THEME

# After: Aliki (dark mode)



PARSER

# The new parser

- RDoc's Ruby parser couldn't handle modern syntax and was hard to maintain
- Tomoya Ishida (@tompng) rewrote it on Prism, fixing many issues along the way
- Fixed 7 long-standing bugs — some open since 2016e

PARSER

# What the rewrite changed

- Roughly half the code (1,243 vs 2,241 lines)
- ~30% faster (0.69s vs 0.97s on 112 files)
- New Ruby syntax is handled by Prism — not by RDoc

PARSER

# Bug fix example: endless methods

```ruby
class Foo       # open Foo
  def bar = 42  # open bar — where's end?
end             # this closes bar, not Foo

class Baz; end # Ripper thinks we're still inside Foo
```

## Ripper

```
class Foo
  — method bar
  — class Baz ← wrong
```

## Prism

```
class Foo
  — method bar
class Baz
```

MARKDOWN

# Writing Markdown in RDoc

- Markdown is widespread — GitHub, Stack Overflow, every doc tool uses it
- It should be the default in RDoc
- RDoc has supported it since 2011, but the parser was buggy and incomplete

MARKDOWN

# RDoc markup vs Markdown

## RDoc markup

```
= Heading

*bold* and _italic_

- item 1
- item 2

  indented code block

{link text}[http://example.com]
```

## Markdown

```
# Heading

**bold** and *italic*

- item 1
- item 2

```ruby
fenced code block
```

[link text](http://example.com)
```

But most people don't know how to write it.

MARKDOWN

# Example: `[text](url)`

*Markdown*

```
[RDoc](rdoc.sh)
```

↓ *Markdown parser rewrites to RDoc* *(extra step)*

***RDoc (no rewrite)***

```
{RDoc}[rdoc.sh]
```

↓ *Shared inline layer tokenizes*

```
TIDYLINK node
```

↓ *ToHtml emits HTML*

```
<a href="rdoc.sh">RDoc
</a>
```

MARKDOWN

# GitHub Flavored Markdown compatibility

| Feature | GFM | RDoc now | v6.11.0 (Jan 2025) |
|---|---|---|---|
| Headings, paragraphs, lists | ✅ | ✅ | ✅ |
| Fenced code + language highlight | ✅ | ✅ | ⚠️ |
| Strikethrough | ✅ | ✅ | ❌ |
| Table inline styling | ✅ | ✅ | ❌ |
| GitHub-style heading anchors | ✅ | ✅ | ❌ |
| Ordered lists, nested lists | ✅ | ⚠️ | ⚠️ |
| Tilde fences, backslash breaks | ✅ | ❌ | ❌ |

MARKDOWN

# Inline styling in tables

## Before

| Name | Type |
| --- | --- |
| age | 'Integer` |

Backtick code broken in table cells

## After

| Name | Type |
| --- | --- |
| age | `Integer` |

Backticks, links, bold, italic now work in cells

MARKDOWN

# Bash syntax highlighting

## Before

```
gem install rdoc
rdoc --server
curl localhost:4000
```

Language tag ignored — highlighted as Ruby

## After

```bash
gem install rdoc
rdoc --server
curl localhost:4000
```

Recognized as bash — commands in blue, options in cyan

MARKDOWN

# C syntax highlighting

## Before

```
static VALUE
rb_str_length(VALUE str)
{
    return LONG2NUM(
      str_strlen(str, NULL));
}
```

No highlighting

## After

```
static VALUE
rb_str_length(VALUE str)
{
    return LONG2NUM(
      str_strlen(str, NULL));
}
```

Keywords red, types cyan, functions purple, macros orange

Important for Ruby's C source code on docs.ruby-lang.org.

MARKDOWN

# Tracking GFM compatibility

| Feature | GFM | RDoc | Notes |
|---|---|---|---|
| ATX Headings ( `#` ) | ✅ | ✅ | Both support levels 1-6, optional closing `#` |
| Setext Headings | ✅ | ✅ | `=` for H1, `-` for H2 |
| Paragraphs | ✅ | ✅ | Full match |
| Indented Code Blocks | ✅ | ✅ | 4 spaces or 1 tab |
| Fenced Code (backticks) | ✅ 3+ | ⚠️ 3 only | RDoc doesn't support 4+ backticks for nesting |
| Fenced Code (tildes) | ✅ `~~~` | ❌ | Conflicts with strikethrough syntax |
| Info strings (language) | ✅ any | ⚠️ limited | `ruby` / `rb`, `c`, and `bash` / `sh` / `shell` / `console` highlighted; others accepted as CSS class |
| Blockquotes | ✅ | ✅ | Full match, nested supported |
| Lazy Continuation | ✅ | ⚠️ | Continuation text is included in blockquote but line break is lost (becomes a space) |
| Bullet Lists | ✅ | ✅ | `*`, `+`, `-` supported |
| Ordered Lists | ✅ `.` `)` | ⚠️ `.` only | RDoc doesn't support `)` delimiter; numbers are always renumbered from 1 |
| Nested Lists | ✅ | ✅ | 4-space indentation |
| Tables | ✅ | ✅ | Full alignment support |
| Thematic Breaks | ✅ | ✅ | `---`, `***`, `___` |
| HTML Blocks | ✅ 7 types | ⚠️ | See below |

ruby.github.io/rdoc/doc/markup_reference/markdown_md.html

TYPE SIGNATURES

# RBS type signatures

- Inline `#:` annotations — type info lives next to the code
- `sig/` directory — `.rbs` files, including stdlib declarations
- Displayed in HTML and `ri`, with linked type names

TYPE SIGNATURES

# RBS in HTML output

**collect {|element| ... } → new_array**
**collect → new_enumerator**

Source

```
[U] () { (Elem item) → U } → ::Array[U]
() → ::Enumerator[Elem, ::Array[untyped]]
```

← RBS signatures

With a block given, calls the block with each element of `self` ; returns a new array whose elements are the return values from the block:

```ruby
a = [:foo, 'bar', 2]
a1 = a.map {|element| element.class }
a1 # => [Symbol, String, Integer]
```

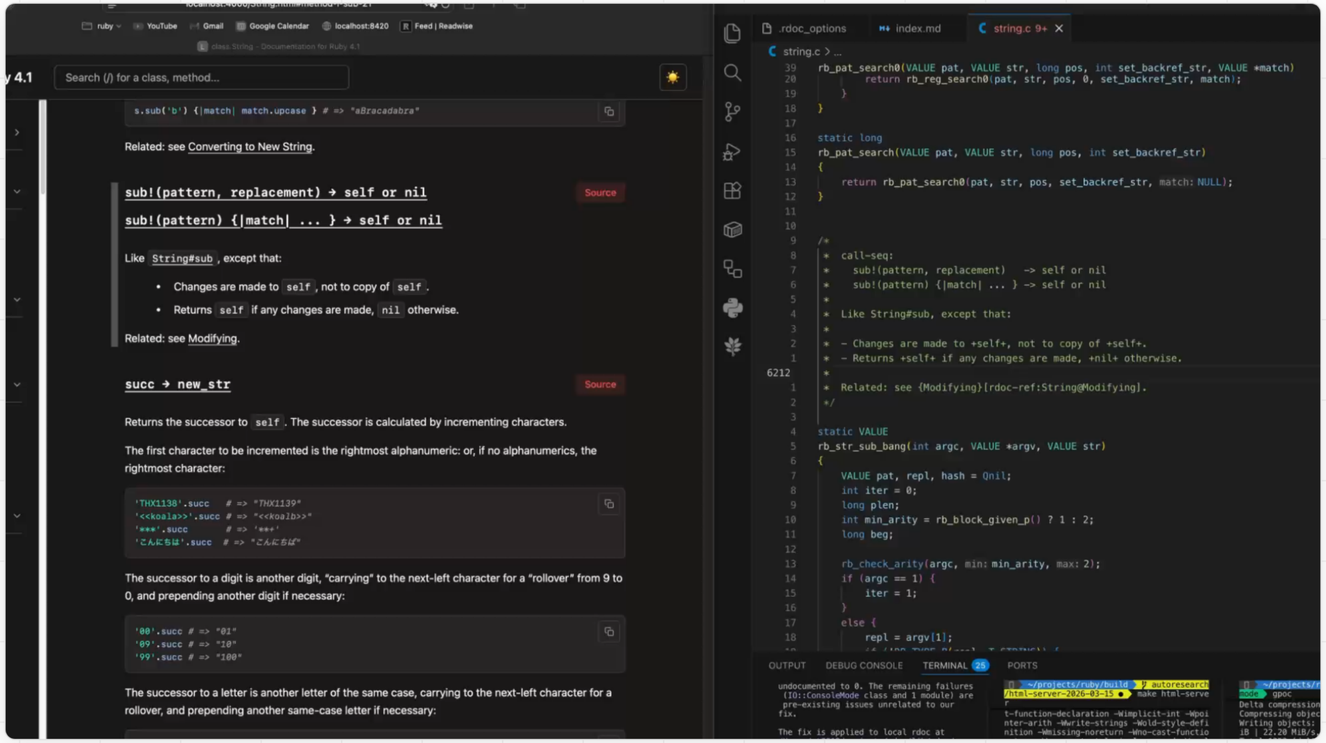
With no block given, returns a new `Enumerator` .

Related: `collect!` ; see also Methods for Converting.

*Also aliased as: map*

SERVER MODE

# rdoc --server



SERVER MODE

# `rdoc --server`

- Changes reflected in browser automatically
- Zero external dependencies — uses Ruby's `TCPServer`
- Incremental re-parsing — shorter response, not instant (yet)
- Run `rdoc --server` in any gem using RDoc to preview its docs
- `make html-server` in ruby/ruby — the same flow for Ruby core docs

# Coming in RDoc 8.0

- Prism parser (default)
- More Markdown improvements
- RBS type signatures in HTML and `ri`
- Server mode

# So what's still ahead for RDoc?

# Three areas ahead

| | |
|---|---|
| **MIGRATE TO MARKDOWN** | Let contributors write in the language they already know |
| **ANNOTATIONS** | Make documentation reflect code design |
| **LLM-READY DOCS** | Optimize documentation consumption for AI |

MIGRATE TO MARKDOWN

# Fully migrate to Markdown

- Automatic RDoc markup → Markdown translation
- Migrate Ruby core's documentation to Markdown
- Change RDoc's default markup to Markdown

A large effort — but AI-assisted tooling can help us get there faster.

ANNOTATIONS

# Make documentation reflect code design

- New RDoc annotations (not RBS): `@abstract`, `@override`, `@interface`
- Describe relationships across components

Design intent in the docs — useful for both humans and AI.

ANNOTATIONS

# Example: `@abstract` + `@override`

```ruby
class Formatter
  # @abstract
  def convert(doc)
    raise NotImplementedError
  end
end
```

```ruby
class ToHtml < Formatter
  # @override
  def convert(doc) = ...
end
```

```ruby
class ToMarkdown < Formatter
  # @override
  def convert(doc) = ...
end
```

Formatter#convert abstract ⟷ ToHtml#convert override

ToMarkdown#convert override

LLM-READY DOCS

# Optimize documentation consumption for AI

But why does AI need something special?

LLM-READY DOCS

# HTML costs more tokens

MARKDOWN

```
# RDoc 8.0

RDoc 8.0 is a **major release** focused on AI-ready
documentation. For details, see the
[release notes](https://github.com/ruby/rdoc/releases).

## What's new

- New *Prism* parser — faster and more accurate
- Markdown improvements — full GFM table support
- RBS type signatures — inline `#:` annotations
- Server mode — live preview with `rdoc --server`
```

**100** tokens

HTML

```
<h1>RDoc 8.0</h1>

<p>RDoc 8.0 is a <strong>major release</strong> focused
on AI-ready documentation. For details, see the
<a href="https://github.com/ruby/rdoc/releases">release notes</a>.</p>

<h2>What's new</h2>

<ul>
  <li>New <em>Prism</em> parser — faster and more accurate</li>
  <li>Markdown improvements — full GFM table support</li>
  <li>RBS type signatures — inline <code>#:</code> annotations</li>
  <li>Server mode — live preview with <code>rdoc --server</code></li>
</ul>
```

**152** tokens (1.52×)

LLM-READY DOCS

# HTML wastes AI context

`docs.ruby-lang.org/en/master/Array.html`

| Format | Tokens |
|---|---|
| Plain text | 51k |
| HTML → Markdown | 69k |
| Raw HTML | **183k** |

LLM-READY DOCS

# Ship Markdown alongside HTML

```
docs.ruby-lang.org/
├── llms.txt       # sitemap
├── Array.html
├── Array.md
├── Hash.html
├── Hash.md
└── ...
```

Two changes: generate `.md` next to every `.html`, and list them in `/llms.txt`.

Tradeoff: doc generation gets slower.

LLM-READY DOCS

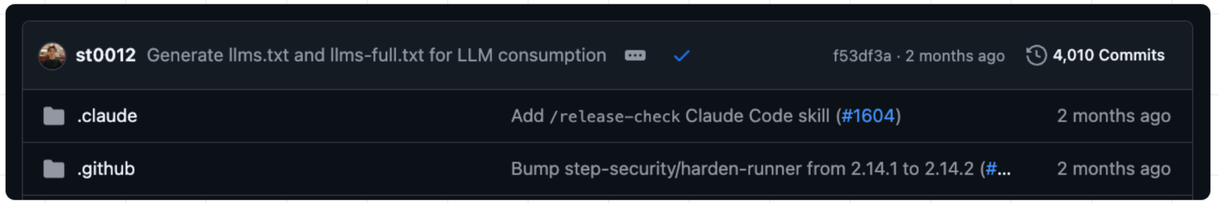
# llms.txt: a sitemap for AI

- A single `/llms.txt` at the site root
- Lists canonical Markdown paths for every doc page
- Agents read it first, then fetch only the `.md` files they need

```
# Ruby docs

- [Array](Array.md): core class
- [Hash](Hash.md): core class
- [String](String.md): core class
```

# I started prototyping this two months ago.



LLM-READY DOCS

# Will this matter long-term?

| Agent | Fetch strategy | .md helps? |
|---|---|---|
| Claude Code | HTML → Markdown + summarize | Marginal |
| opencode | HTML → Markdown | Marginal |
| Codex CLI | Delegated to OpenAI servers | Likely not |

Most mainstream agents don't need this

**No.** LLM-ready docs aren't needed.

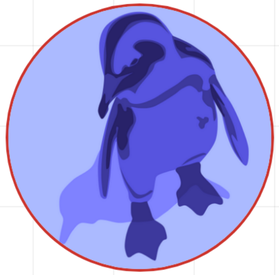
**The best thing RDoc can do for AI is be good at its job.**

# Summary

- What makes docs useful to AI is what makes them useful to developers
- RDoc 8.0 modernizes the foundation — new parser, Markdown improvements, RBS, server mode
- Still ahead — Markdown migration, design annotations, Markdown parser refactor, better integration with other tools

# Special thanks to

@kou



@tompng

For all the reviews on my RDoc PRs.

# Thank you!

GitHub: @st0012
BlueSky: @st0012.dev