Ruby - Feature #21386

Introduce `Enumerable#join_map`

05/30/2025 04:33 PM - matheusrich (Matheus Richard)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Problem

The pattern .map { ... }.join(sep) is extremely common in Ruby codebases:

```
users.map(&:name).join(", ")
```

It's expressive but repetitive (both logically and computationally). This pattern allocates an intermediate array and does two passes over the collection.

Real-world usage is widespread:

- Open source Ruby projects using this pattern
- Within Rails
- Within Ruby itself

Proposal

Just like filter_map exists to collapse a common map + compact, this proposal introduces Enumerable#join_map, which maps and joins in a single pass.

```
users.join_map(", ", &:name)
```

A Ruby implementation could look like this:

```
module Enumerable
  def join_map(sep = "")
    return "" unless block_given?

    str = +""
    first = true

    each do |item|
      str << sep unless first
      str << yield(item).to_s
      first = false
    end

    str
  end
end
```

The name join_map follows the precedent of filter_map, emphasizing the final operation (join) over the intermediate (map).

Prior Art

Some other languages have similar functionality, but with different names or implementations:

Elixir

Elixir has this via the Enum.map_join/3 function:

```
Enum.map_join([1, 2, 3], &(&1 * 2))
"246"

Enum.map_join([1, 2, 3], " = ", &(&1 * 2))
"2 = 4 = 6"
```

Crystal

Crystal, on the other hand, uses Enumerable#join with a block:

```
[1, 2, 3].join(", ") { |i| -i } # => "-1, -2, -3"
```

Kotlin

Kotlin has a similar function called joinToString that can take a transformation function:

```
val chars = charArrayOf('a', 'b', 'c')
println(chars.joinToString() { it.uppercaseChar().toString() }) // A, B, C
```