Ruby - Feature #20298

Introduce `Time()` type-cast / constructor.

02/26/2024 12:34 AM - ioquatix (Samuel Williams)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Many Ruby primitive types have constructors, e.g. Integer(...), String(...), Float(...), etc. These usually convert from some subset of types, e.g. Float(1) -> 1.0 will convert an Integer to a Float, and Float("1") -> 1.0 will parse a String to a Float, and similar for other type casts/constructors.

I'd like to propose we introduce something similar for Time (and possibly this extends to Date/DateTime in a follow up proposal).

Suggested implementation could look something like this:

```
def Time(value)
  case value
  when Time
    value
  when Integer
    Time.at(value)
  when String # The format is assumed to be the result of `Time#to_s`.
    Time.new(value)
  else
    value.to_time
  end
end
```

Alternatively, we might like to be a little more specific with the else clause/error handling.

Background

In a project, I need to support multiple serialization formats/coders, including MessagePack, Marshal and JSON.

While Marshal and MessagePack are capable of serializing Time instances, JSON is not.

The code looks a bit like this:

```
data = fetch_job_data
job = @coder.load(data)
scheduled_at = Time(job[:scheduled_at]) # Hypothetical type-cast as outlined above
```

Additional Observations

While some primitive data types accept themselves as arguments and construct by copy, e.g. Array.new(Array.new), others do not, e.g. Hash and Time. Perhaps Time.new(Time.new) should behave similarly to Array.new(Array.new) - the outer instance is a copy of the inner.

History

#1 - 02/26/2024 12:44 AM - ioquatix (Samuel Williams)

- Description updated

#2 - 02/27/2024 08:05 PM - zverok (Victor Shepelev)

I would like to register a dissenting opinion about the protocol :)

I believe that ClassName() convention, while old, is one of the clumsy parts of the Ruby API: it is kinda the only place where two absolutely unrelated things (a method and a class) are made related only by a convention; and the only where thing an thing(...) are referring to absolutely different entities.

The language user can't find ClassName() method's docs in the ClassName's docs; the method doesn't "follow" the renaming or inheriting of the ClassName.

I don't know the whole story, but probably the convention had emerged on early stages of Ruby's life, somewhat inspired by Python's callable classes, and somewhat compensates for absence of "properly callable objects" in Ruby (Probably the sugar foo.call() → foo.() was another idea how to compensate for that; but nobody seems to like it except me :)

I think it would be better to follow something that is natural for Ruby (in a sense of discoverability and relations between concepts), not just traditional. Like, just ClassName.coerce(), for example.

#3 - 03/12/2024 05:38 PM - matheusrich (Matheus Richard)

To give some outside perspective, Rust often uses from to convert from a type to another. This could be one option for us

```
Time.from(whatever)
```