Ruby master - Feature #15526

New way to destruct an object hash

01/11/2019 07:45 PM - alissonbruno.sa (Alisson Santos)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

JavaScript has a nice a neat way to destruct objects.

```
const person = { name: "John Doe", age: 33 };
const { name, age } =  person;
```

Erlang has a similar way to destruct a tuple:

```
Person = {"John Doe", 33}
{Name, Age} = Person
```

I think it's very handy and would be nice if we have something similar in Ruby.

```
config = { host: 'localhost', port: 3000 }
{ host, port } = config
```

I know Ruby has Hash#values_at, but I think this way it's more readable and understandable

What do you guys think?

History

#1 - 01/11/2019 07:57 PM - zverok (Victor Shepelev)

Not exactly what you are describing, but, funny enough, there is a way!

```
config = { host: 'localhost', port: 3000 }

config.then { |host:, port:|
  p "host=#{host}, port=#{port}"
}
```

#2 - 01/11/2019 08:15 PM - shevegen (Robert A. Heiler)

> What do you guys think?

Ultimately you only have to convince matz, so the rest is just people giving opinions. :)

Victor wrote:

> Not exactly what you are describing, but, funny enough, there is a way!

```
config.then { |host:, port:|
```

Admit it, you only wanted to use **.then**. ;)

To the original suggestion:

```
{ host, port } = config
```

Personally I am not a huge fan of the syntax proposal simply due to the {} part. My brain tends to associate this as a Hash, so I get confused when the {} is on the left side. {} already has quite some meanings, e. g. do/end. I would rather prefer to keep any meaning of {} smaller rather than expand on it.

Syntax aside, I am not sure I like the proposal as such either, but I don't care that much really. My opinion is slightly against it but it's not that strong.

> I know Ruby has Hash#values_at, but I think this way it's more readable and understandable

I prefer .values_at since I like object.method notation in general, unless there is a significantly shorter and readable way that does not cause that much confusion. But I think this is difficult to agree because what may be readable or easy to understand for one person, may be difficult to understand for someone else.

Actually, although I myself still am not using yield_self/then, I'd rather prefer the variant shown by Victor, rather than the { } variant on the left hand side, but that may be just my own personal opinion.

If you feel strongly about your proposal you could consider adding your proposal to any upcoming developer meeting.

#3 - 01/11/2019 08:35 PM - zverok (Victor Shepelev)

> Admit it, you only wanted to use .then

That's absolutely not the point. The real point here is:

1) Ruby has very consistent "argument **deconstruction**" rules. Basically, "it is deconstructed the same way it is constructed", e.g., if you can call (construct) with arguments like meth(a, *b, c: 1, d: 2, **e), you can also receive (deconstruct) arguments exactly the same way: a, *b, c:, d: nil, **e.

2) The deconstruction is fully enabled in arguments (including keyword arguments) to procs and methods

3) Initially (before keyword arguments), the variable assignment had the same "expressive power", e.g. you could define a method like

```
def meth(a, b, *rest)
```

...and you could assign variables the same way:

```
a, b, *rest = [1, 2, 3, 4]
```

4) When keyword arguments were introduced, there was no matching syntax for variable assignment. If it **would** exist, it most probably **should** look this way (same as arguments, remember):

```
a:, b:, **kwrest = {a: 1, b: 2, c: 3, d: 4}
```

It obviously looks confusing (and would probably be hard for the parser), and I believe that's the reason it was NOT added to the language.

5) I don't think adding "orphan feature" (looking like nothing else in the language) just for the sake of this particular case would be ever accepted; and I don't think it should.

6) In the meantime, "chainable" code style (with Enumerable and #then) allows to embrace argument deconstruction in its full force:

```
config.then { |host:, port: 8080, **rest|
```

...which is NOT the main (and obviously not the only) reason to use this style, just a nice side-effect of its consistency with the rest of the language.

#4 - 01/12/2019 03:04 PM - janfri (Jan Friedrich)

zverok (Victor Shepelev) wrote:

> 1) Ruby has very consistent "argument **deconstruction**" rules. Basically, "it is deconstructed the same way it is constructed", e.g., if you can call (construct) with arguments like meth(a, *b, c: 1, d: 2, **e), you can also receive (deconstruct) arguments exactly the same way: a, *b, c:, d: nil, **e.
>
> 2) The deconstruction is fully enabled in arguments (including keyword arguments) to procs and methods
>
> 3) Initially (before keyword arguments), the variable assignment had the same "expressive power", e.g. you could define a method like
>
> ```
> def meth(a, b, *rest)
> ```
>
> ...and you could assign variables the same way:

```
a, b, *rest = [1, 2, 3, 4]
```

4) When keyword arguments were introduced, there was no matching syntax for variable assignment. If it **would** exist, it most probably **should** look this way (same as arguments, remember):

```
a:, b:, **kwrest = {a: 1, b: 2, c: 3, d: 4}
```

It obviously looks confusing (and would probably be hard for the parser), and I believe that's the reason it was NOT added to the language.

5) I don't think adding "orphan feature" (looking like nothing else in the language) just for the sake of this particular case would be ever accepted; and I don't think it should.

6) In the meantime, "chainable" code style (with Enumerable and #then) allows to embrace argument deconstruction in its full force:

```
config.then { |host:, port: 8080, **rest|
```

...which is NOT the main (and obviously not the only) reason to use this style, just a nice side-effect of its consistency with the rest of the language.

+1 Great analysis.

#5 - 01/14/2019 03:45 PM - elia (Elia Schito)

Related/duplicate https://bugs.ruby-lang.org/issues/11758