#### ORACLE

## Designing an Intuitive Language-Agnostic Integration of Foreign Objects in Ruby

Benoit Daloze

Truffle Workshop, ECOOP 2022

## Who am I?



Benoit Daloze

Twitter: @eregontp GitHub: @eregon Website: https://eregon.me



- TruffleRuby lead at Oracle Labs, Zurich
- Worked on TruffleRuby since 2014
- PhD on parallelism in dynamic languages
- Maintainer of ruby/spec
- CRuby (MRI) committer

## TruffleRuby



- A high-performance Ruby implementation
- Uses the GraalVM. JIT Compiler
- Targets full compatibility with CRuby 3.0, including C extensions
- GitHub: oracle/truffleruby, Twitter: @TruffleRuby Website: https://graalvm.org/ruby

## TruffleRuby: Peak performance on yjit-bench (14 benchmarks)



From https://eregon.me/blog/2022/01/06/benchmarking-cruby-mjit-yjit-jruby-truffleruby.html



# Interoperability with other languages, and their objects, aka foreign objects

## Once upon a time there was a foreign object

# In a Ruby file (.rb)
foreign\_object = Polyglot.eval 'js', '({ a: 1, b: 2 })'

- What can I do with this object?
- Can I print it?
- Can I access members?
- Can I ask its class? Does it have a Ruby class?
- What methods are available on it?

## GraalVM and Truffle's InteropLibrary (130 methods)

Traits:

- executable, instantiable
- members
- hash entries
- array elements, buffer elements
- iterable
- pointer
- associated metaobject (getMetaObject)
- declaring meta object
- source location
- identity
- language
- scope

#### Types:

- Null
- Boolean
- String
- Number
- Date, Time or TimeZone
- Duration
- Exception
- Meta-Object
- Iterator

## Which semantics to prefer?



```
# In a Ruby file (.rb)
js_array = Polyglot.eval 'js', '[1, 2, 3]'
js_array.map { |x| x * 2 }
```

- Should that use JS's Array.prototype.map() or Ruby's Array#map?
- It should be Ruby's Array#map, because this helps more to make existing Ruby code handle foreign objects just fine, just like Ruby objects.
- Still possible to call the foreign object's method explicitly: Interop.invoke\_member(js\_array, :map, proc { |e, idx| e + idx })

## To give a Ruby class for foreign objects or not?

Possibilities:

- Do not give a Ruby class. What should foreign.class return then? How to make code which relies on object.class returning a Ruby class work?
- Give the same Ruby class for all foreign objects
- Give a Ruby class based on the traits (and type) of the foreign objects.
   e.g. foreign object with array elements: ObjectTrait + ArrayTrait = ForeignArrayObject.
   Makes it easy to add methods for a trait, and understand what "kind of object" it is.
- Give a unique Ruby class per foreign object
- Give a per-foreign-metaobject Ruby class

## Method lookup for foreign objects

Method lookup for Ruby objects:

- Search method in object.class.ancestors
- Call

method\_missing(name, \*args)

Method lookup for foreign objects:

- Search method in object.class.ancestors
- invokeMember(name, \*args) if isMemberInvocable(name)
- readMember(name) if
   isMemberReadable(name) & & args.empty?
   (for obj.foo)
- Call

method\_missing(name, \*args)

## A foreign object



#### • Can I print it?

p foreign\_object
#<Polyglot::ForeignObject[JavaScript] Object:0x1221a708 a=1, b=2>
puts foreign\_object
#<Polyglot::ForeignObject[JavaScript] {a: 1, b: 2}>

#### • Can I access members?

foreign\_object[:a] # => 1
foreign\_object.a # => 1
foreign\_object.instance\_variables # => [:a, :b]

## A foreign object



• Can I ask its class? Does it have a Ruby class?

```
foreign_object.class
# => Polyglot::ForeignObject
foreign_object.class.ancestors
# => [Polyglot::ForeignObject, Polyglot::ObjectTrait,
# Object, Kernel, BasicObject]
```

• What methods are available on it?

foreign\_object.methods

# => [:[], :[]=, :==, :nil?, :instance\_variables, :hash, :send, ...

## A foreign array

foreign\_array = Polyglot.eval 'js', '[1, 2, 3]'

- Can I read and write elements? foreign\_array[0] = foreign\_array[1] + 2
- Can I iterate it? Can I use any method from Enumerable? foreign\_array.map { |e| e \* 3 }.select { |e| e.even? }

```
• Does it have a Ruby class? What is the hierarchy?
foreign_array.class
# => Polyglot::ForeignArray
foreign_array.class.ancestors
# => [Polyglot::ForeignArray, Polyglot::ArrayTrait,
# Polyglot::IterableTrait, Enumerable,
# Polyglot::ForeignObject, Polyglot::ObjectTrait,
# Object, Kernel, BasicObject]
```

## **Going further**



42 + foreign\_number

foreign\_string.capitalize

```
foreign_array.each_slice { |a, b| ... }
```

foreign\_map.each\_pair { |key, value| ... }

#### begin

```
foreign_object.foo()
rescue Exception => foreign_exception
    puts foreign_exception.message, foreign_exception.backtrace
end
```

## A look at the implementation

module Polyglot module HashTrait module ArrayTrait module ExceptionTrait module ExecutableTrait module InstantiableTrait module IterableTrait module IteratorTrait module MetaObjectTrait module NullTrait module NumberTrait module PointerTrait module StringTrait module ObjectTrait end



## A look at the implementation



```
module Polyglot
    class ForeignObject < Object
        include ObjectTrait
    end</pre>
```

```
class ForeignException < Exception
    include ObjectTrait
    include ExceptionTrait
    end
end</pre>
```

## A look at the implementation

```
module HashTrait
  def [](key)
    Truffle::Interop.read_hash_value_or_default(self, key, nil)
  end
end
```

```
module ArrayTrait
  def [](index)
    size = Truffle::Interop.array_size(self)
    index += size if index < 0
    return nil if index < 0 || index >= size
    Truffle::Interop.read_array_element(self, index)
    end
end
```

### **Inner contexts**

- JavaScript is single-threaded, i.e., can only be executed on one thread at a time
- · Ruby is multi-threaded, many Ruby programs and the standard library use multiple threads
- In practice, this means Truffle errors when Ruby tries to create additional threads when JavaScript is used in the same Context
- Truffle inner contexts to the rescue! Ruby can have multiple threads in the root context, and JavaScript is on its own in an inner context.
- What about passing objects between these contexts? All is transparently proxied via InteropLibrary with TruffleContext#eval(String language, Source source)!

## Polyglot::InnerContext

```
Polyglot::InnerContext.new do |ctx|
js_object = ctx.eval('js', '({ a: 1 })')
# => #<Polyglot::ForeignObject[JavaScript] Object:0x46c805be a=1>
```

```
js_function = ctx.eval('js', '(function(arg) { console.log(arg) })')
ruby_object = [1, 2, 3]
js_function.call(ruby_object)
# => 1,2,3
end
```

- js\_object is a wrapper object implementing InteropLibrary and forwarding all messages to the real JavaScript object. It looks exactly like a foreign object to Ruby and there is no difference
- ruby\_object is wrapped similarly, but for the other direction

## Gems using this deep integration

- rails/execjs: Executing JavaScript from Ruby. Each ExecJS::Context has its own isolated state.
- rubyjs/mini\_racer: Embedded JavaScript engine in Ruby. Supports attaching Ruby lambdas and calling them from JavaScript and many more features.

For both:

- Their GraalVM backend is written entirely in Ruby using ways from this talk.
- They use Truffle inner contexts for isolation (and to let the Ruby context use multiple threads).

## Try it yourself



- Get GraalVM, either a release or from https://github.com/graalvm/graalvm-ce-dev-builds
- export PATH=graalvm/bin:\$PATH
- Install Ruby: gu install ruby
- ruby --polyglot --jvm -e 'Polyglot::InnerContext.new { |ctx|
   p ctx.eval "js", "({ a: 1 })" }'
- Start a shell: ruby --polyglot --jvm

## Conclusion



For best integrations of foreign objects:

- They should be given a class/meta object of the language representing their interop traits
- They should have the same methods as corresponding objects of the language
- Using inner context avoids the issue of multithreading incompatible with single-threaded languages
- When the integration is deep enough, a lot of existing code just works on foreign objects