Writing Native Extensions using Crystal

Paul Hoffer

Why?

- Performance
  - (spoiler alert: it's fast!)
- Ease of development
  - Ruby like syntax & constructs
- Feasibility
  - Favorable cost/benefit equation
  - Resources- time and devs

Let’s Experiment!

- ActiveSupport is pure Ruby
- fast_blank was 20x faster for `String#blank?`

- ActiveSupport::Inflector is all string operations

- Sounds good, let’s go!

ActiveSupport::Inflector

vs

Crystal::Inflector

ActiveSupport::Inflector

```ruby
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)

  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match] || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capitalize}" }
    string.gsub!('/'.freeze, '::'.freeze)
    string
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
    word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
    word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
    word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
    word.tr!("-".freeze, "_".freeze)
    word.downcase!
    word
  end
  def humanize(lower_case_and_underscored_word, options = {})
    result = lower_case_and_underscored_word.to_s.dup
    inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
    result.sub!(/\A_+/, ''.freeze)
    result.sub!(/_id\z/, ''.freeze)
    result.tr!('_'.freeze, ' '.freeze)
    result.gsub!(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match] || match.downcase}"
    end
    if options.fetch(:capitalize, true)
      result.sub!(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string)
    string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
  end
  def dasherize(underscored_word)
    underscored_word.tr('_'.freeze, '-'.freeze)
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex('::')
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex('::') || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).include?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  def apply_inflections(word, rules)
    result = word.to_s.dup
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result
    end
  end
end
```

Crystal::Inflector

```ruby
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
  def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

ActiveSupport::Inflector

```ruby
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)

  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match] || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capitalize }" }
    string.gsub!('/'.freeze, '::'.freeze)
    string
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
    word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
    word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
    word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
    word.tr!("-".freeze, "_".freeze)
    word.downcase!
    word
  end
  def humanize(lower_case_and_underscored_word, options = {})
    result = lower_case_and_underscored_word.to_s.dup
    inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
    result.sub!(/\A_+/, ''.freeze)
    result.sub!(/_id\z/, ''.freeze)
    result.tr!('_'.freeze, ' '.freeze)
    result.gsub!(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match] || match.downcase}"
    end
    if options.fetch(:capitalize, true)
      result.sub!(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string)
    string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
  end
  def dasherize(underscored_word)
    underscored_word.tr('_'.freeze, '-'.freeze)
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex('::')
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex('::') || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).include?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  def apply_inflections(word, rules)
    result = word.to_s.dup
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result
    end
  end
end
```

85% **copied**

4% **fixed**

Crystal::Inflector

```ruby
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("-", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

~15 minutes to fix

Basic Differences

```
word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
```

- Replace bang methods
- remove .freeze
- replace single quotes
- add extra \ to regex back references

Basic Differences

```
word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
```

- Replace bang methods
- remove .freeze
- replace single quotes
- add extra \ to regex back references

Is that it?

(yes and no)

Remaining Steps

- Convert between Ruby data and Crystal data
- Wrap Crystal methods
  - this lets us to use pure Crystal libraries
- Initialize Crystal methods for Ruby
  - (make these methods available via C API)
- Write some Ruby!

Data conversion

- C does not have direct variable access
  - Ruby API defines VALUE type in C
  - All Ruby objects are VALUE
    - Allows for dynamic nature of Ruby
- Need to convert to Crystal objects (and back)
- Luckily, this has been done before!
  - (and there's docs on it too)

LibRuby

```
lib LibRuby
  type VALUE = Void*
  type METHOD_FUNC = VALUE, VALUE -> VALUE # STUB

  $rb_cObject : VALUE
  $rb_cNumeric : VALUE

  # integers
  fun rb_num2int(value : VALUE) : Int32
  fun rb_int2inum(value : Int32) : VALUE

  # strings
  fun rb_str_to_str(value : VALUE) : VALUE
  fun rb_string_value_cstr(ptr : VALUE*) : UInt8*
  fun rb_str_new_cstr(str : UInt8*) : VALUE

  fun rb_id2sym(value : ID) : VALUE
  fun rb_intern(name : UInt8*) : ID
end

struct Bool
  def to_ruby
    val = self ? 20_u64 : 0_u64
    Pointer(Void).new(val).as(LibRuby::VALUE)
  end
end
```

```
class String
  def to_ruby
    LibRuby.rb_str_new_cstr(self)
  end
  def self.from_ruby(str : LibRuby::VALUE)
    rb = LibRuby.rb_str_to_str(str)
     ptr = pointerof(rb)
    c_str = LibRuby.rb_string_value_cstr(ptr)
    String.new(c)
  end
end

struct Int
  def to_ruby
    LibRuby.rb_int2inum(self)
  end
  def self.from_ruby(int)
    LibRuby.rb_num2int(int)
  end
end

struct Nil
  def to_ruby
    Pointer(Void).new(8_u64).as(LibRuby::VALUE)
  end
end
```

Wrap Crystal Methods

- We don’t have to wrap our methods
- Could just define everything directly
- However…
  - maybe we want to use an existing Crystal library?
  - DRY up our Crystal
  - Easier debugging
  - Also, it’s not that cumbersome

Wrap Crystal Methods

```
module Wrapper

  def self.ordinal(self : LibRuby::VALUE)
    int = Int.from_ruby(self)
    int.ordinal.to_ruby
  end

  def self.ordinalize(self : LibRuby::VALUE)
    int = Int.from_ruby(self)
    int.ordinalize.to_ruby
  end

  def self.squish(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.squish.to_ruby
  end

  def self.blank?(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.blank?.to_ruby
  end

  def self.titleize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.titleize.to_ruby
  end

  def self.titlecase(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.titlecase.to_ruby
  end

  def self.dasherize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.dasherize.to_ruby
  end

  def self.deconstantize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.deconstantize.to_ruby
  end

  def self.tableize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.tableize.to_ruby
  end

  def self.classify(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.classify.to_ruby
  end
end
```

Init C functions for Ruby

```
require "../lib_ruby"
require “./wrapper"

fun init = Init_inflector
  GC.init
  LibCrystalMain.__crystal_main(0, Pointer(Pointer(UInt8)).null)

  string = LibRuby.rb_define_class("String", LibRuby.rb_cObject)

  LibRuby.rb_define_method(string, "cr_squish",        ->Wrapper.squish,        0)
  LibRuby.rb_define_method(string, "cr_blank?",        ->Wrapper.blank?,        0)
  LibRuby.rb_define_method(string, "cr_pluralize",     ->Wrapper.pluralize,     0)
  LibRuby.rb_define_method(string, "cr_humanize",      ->Wrapper.humanize,      0)

  integer = LibRuby.rb_define_class("Integer", LibRuby.rb_cNumeric)
  LibRuby.rb_define_method(integer, "cr_ordinal",      ->Wrapper.ordinal,       0)
  LibRuby.rb_define_method(integer, "cr_ordinalize",   ->Wrapper.ordinalize,    0)

end
```

Ruby Usage

```ruby
require "./inflector"

puts 1.cr_ordinalize                                  # => "1st"
puts 2.cr_ordinalize                                  # => "2nd"
puts ''.cr_blank?                                     # => true
puts '   '.cr_blank?                                  # => true
puts "apples".cr_pluralize                            # => "apples"
puts "apples".cr_singularize                          # => "apple"
puts "active_record/errors".cr_camelize               # => "ActiveRecord::Errors"
puts "fancyCategory".cr_tableize                      # => "fancy_categories"
puts "employee_salary".cr_humanize                    # => "Employee salary"
puts "author_id".cr_humanize                          # => "Author"
```

Benchmark-ips results

iterations/second	ActiveSupport	Crystal	Improvement
ordinal	418,430	2,027,814	4.85x
ordinalize	140,863	556,205	3.95x
blank?	241,471	785,621	3.25x
squish	206,708	735,772	3.56x
pluralize	5,985	25,061	4.19x
singularize	6,276	28,546	4.55x
camelize	36,658	79,380	2.17x
titleize	14,837	38,707	2.61x
underscore	20,560	73,844	3.59x
demodulize	608,325	788,773	1.30x
deconstantize	532,506	797,424	1.50x
tableize	8,302	28,792	3.47x
classify	14,909	56,535	3.79x
humanize	40,904	82,314	2.01x
upcase_first	987,707	1,423,886	1.44x
foreign_key	13,642	66,009	4.84x

Crystal Resources

- Official docs
  - http://crystal-lang.org/api/
- Crystal for Rubyists
  - https://github.com/crystal-lang/crystal/wiki/Crystal-for-Rubyists
  - http://www.crystalforrubyists.com
- Awesome Crystal
  - https://github.com/veelenga/awesome-crystal
- Introducing Crystal (Will Leinweber @ Ruby on Ales 2016)
  - https://confreaks.tv/videos/roa2016-introducing-the-crystal-programming-language

Ruby C API Resources

- Definitive Guide to Ruby's C API
  - https://silverhammermba.github.io/emberb/
- Chris Lalancette blog series
  - http://clalance.blogspot.com/2011/01/writing-ruby-extensions-in-c-part-1.html
  - all 12 parts can be found by changing last number
- Ruby Cross Reference
  - http://rxr.whitequark.org/mri/ident
- Ruby C API Basics
  - http://blog.x-aeon.com/2012/12/13/the-ruby-c-api-basics/

Github repos

- Native extensions in Crystal
  - https://github.com/phoffer/crystalized_ruby
- Crystal version of AS::Inflector
  - https://github.com/phoffer/inflector.cr
- Crystal lang
  - http://github.com/crystal-lang/crystal