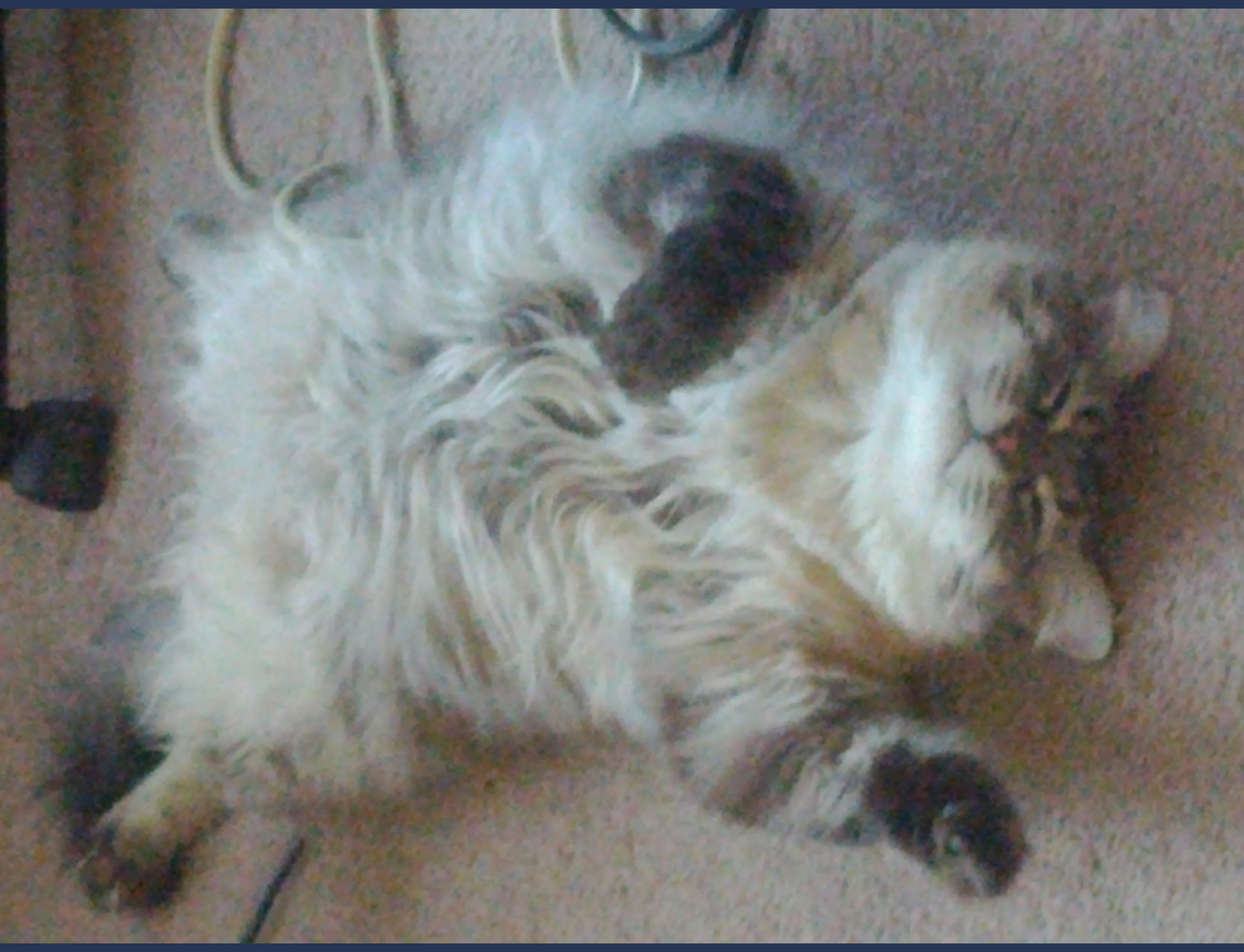
Designing a Great Ruby API

- Sean Griffin
- Software Developer at thoughtbot
- Rails committer
- Maintainer of Active Record (I'm sorry)
- Bikeshed co-host



What goes into a great API

1. Identify what is missing
2. **Roughly** decide what you want it to look like
3. HAVE GOOD TESTS
4. Create the objects which will make up your system
5. Use your objects internally
6. Manually compose those together as needed
7. Extract DSLs where there is duplication or pain

Identifying Holes in Your API

Overriding an Attribute

```
class Product < ActiveRecord::Base
  def price
    unless super.nil?
      Money.new(super)
    end
  end

  def price=(price)
    if price.is_a?(Money)
      super(price.amount)
    else
      super
    end
  end
end
```

Things you might break

- `_before_type_cast`
- Dirty checking
- Form builder integration

Things you might want

- Control over SQL representation
- Ability to query with your values

```
Product.where(price: Money.new(50))
```


Warning!
Rails Internals Ahead
Not for the faint of heart!

Time Zone Conversion

```ruby
# lib/active_record/attribute_methods/time_zone_conversion.rb

if create_time_zone_conversion_attribute?(attr_name, columns_hash[attr_name])
  method_body, line = <<-EOV, __LINE__ + 1
    def #{attr_name}=(time)
      time_with_zone = convert_value_to_time_zone("#{attr_name}", time)
      previous_time = attribute_changed?("#{attr_name}") ? changed_attributes["#{attr_name}"] : read_attribute(:#{attr_name})
      write_attribute(:#{attr_name}, time)
      #{attr_name}_will_change! if previous_time != time_with_zone
      @attributes_cache["#{attr_name}"] = time_with_zone
    end
  EOV
  generated_attribute_methods.module_eval(method_body, __FILE__, line)
end
```

Time Zone Conversion

- Overriding an attribute reader/writer
- Duplicates code from other parts of Active Record
- Jumps through significant hoops for a relatively minor behavior change

Time Zone Conversion

- Overriding an attribute reader/writer ✅
- Duplicates code from other parts of Active Record ✅
- Jumps through significant hoops for a relatively minor behavior change ✅

Time Zone Conversion

- Overriding an attribute reader/writer ✅
- Duplicates code from other parts of Active Record ✅
- Jumps through significant hoops for a relatively minor behavior change ✅
- Introduces large number of subtle bugs 💣

Serialized Attributes

```ruby
# lib/active_record/attribute_methods/serialization.rb

def typecasted_attribute_value(name)
  if self.class.serialized_attributes.include?(name)
    @attributes[name].serialized_value
  else
    super
  end
end
```

```ruby
module ClassMethods # :nodoc:
  def initialize_attributes(attributes, options = {})
    serialized = (options.delete(:serialized) { true }) ? :serialized : :unserialized
    super(attributes, options)

    serialized_attributes.each do |key, coder|
      if attributes.key?(key)
        attributes[key] = Attribute.new(coder, attributes[key], serialized)
      end
    end

    attributes
  end
end

def should_record_timestamps?
  super || (self.record_timestamps && (attributes.keys & self.class.serialized_attributes.keys).present?)
end

def keys_for_partial_write
  super | (attributes.keys & self.class.serialized_attributes.keys)
end
```

```ruby
def type_cast_attribute_for_write(column, value)
  if column && coder = self.class.serialized_attributes[column.name]
    Attribute.new(coder, value, :unserialized)
  else
    super
  end
end

def raw_type_cast_attribute_for_write(column, value)
  if column && coder = self.class.serialized_attributes[column.name]
    Attribute.new(coder, value, :serialized)
  else
    super
  end
end

def _field_changed?(attr, old, value)
  if self.class.serialized_attributes.include?(attr)
    old != value
  else
    super
  end
end
```

```ruby
def read_attribute_before_type_cast(attr_name)
  if self.class.serialized_attributes.include?(attr_name)
    super.unserialized_value
  else
    super
  end
end

def attributes_before_type_cast
  super.dup.tap do |attributes|
    self.class.serialized_attributes.each_key do |key|
      if attributes.key?(key)
        attributes[key] = attributes[key].unserialized_value
      end
    end
  end
end

def attributes_for_coder
  attribute_names.each_with_object({}) do |name, attrs|
    attrs[name] = if self.class.serialized_attributes.include?(name)
                    @attributes[name].serialized_value
                  else
                    read_attribute(name)
                  end
  end
end
```

Serialized Attributes

- Overriding an attribute reader/writer
- Duplicates code from other parts of Active Record
- Jumps through significant hoops for a relatively minor behavior change
- Overrides literally everything
- Introduces large number of subtle bugs

Serialized Attributes

- Overriding an attribute reader/writer ❌
- Duplicates code from other parts of Active Record ✅
- Jumps through significant hoops for a relatively minor behavior change ✅
- Overrides literally everything ✅
- Introduces large number of subtle bugs 💣 💣 💣 💣

Enum

```
# def status=(value) self[:status] = statuses[value] end
klass.send(:detect_enum_conflict!, name, "#{name}=")
define_method("#{name}=") { |value|
  if enum_values.has_key?(value) || value.blank?
    self[name] = enum_values[value]
  elsif enum_values.has_value?(value)
    # Assigning a value directly is not a end-user feature, hence it's not documented.
    # This is used internally to make building objects from the generated scopes work
    # as expected, i.e. +Conversation.archived.build.archived?+ should be true.
    self[name] = value
  else
    raise ArgumentError, "'#{value}' is not a valid #{name}"
  end
}

# def status() statuses.key self[:status] end
klass.send(:detect_enum_conflict!, name, name)
define_method(name) { enum_values.key self[name] }

# def status_before_type_cast() statuses.key self[:status] end
klass.send(:detect_enum_conflict!, name, "#{name}_before_type_cast")
define_method("#{name}_before_type_cast") { enum_values.key self[name] }
```

Enum

- Overriding an attribute reader/writer
- Duplicates code from other parts of Active Record
- Jumps through significant hoops for a relatively minor behavior change
- Introduces large number of subtle bugs

Enum

- Overriding an attribute reader/writer ✅
- Duplicates code from other parts of Active Record ✅ ✅ ✅
- Jumps through significant hoops for a relatively minor behavior change ✅
- Introduces large number of subtle bugs 💣 💣 💣 💣

We've Found a Missing Concept

Typed attributes are found and overridden *all over the place*. If we want to do this so much, maybe others do as well.

Type Casting

Type Casting

```
x = "1"
x.class # => String

x.to_i # => 1
x.to_i.class # => Fixnum
```

Type Coercion

```
user = User.new
user.age = "30" # => "30"

user.age # => 30
user.age.class # => Fixnum
```

```ruby
class Product < ActiveRecord::Base
  attribute :name, String
  attribute :price, Integer
end
```

Always design a thing by considering it in its next larger context -- a chair in a room, a room in a house, a house in an environment, an environment in a city plan

-- Eliel Saarinen

Rule #1 of Refactoring

Have good test coverage

Rule #2 of Refactoring

HAVE GOOD TEST COVERAGE

Rule #3 of Refactoring

See rules 1 and 2

Where we start

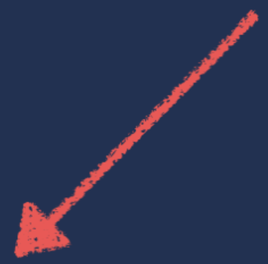
```ruby
# lib/active_record/connection_adapters/column.rb

# Casts value (which is a String) to an appropriate instance.
def type_cast(value)
  return nil if value.nil?
  return coder.load(value) if encoded?

  klass = self.class

  case type
  when :string, :text        then value
  when :integer              then klass.value_to_integer(value)
  when :float                then value.to_f
  when :decimal              then klass.value_to_decimal(value)
  when :datetime, :timestamp then klass.string_to_time(value)
  when :time                 then klass.string_to_dummy_time(value)
  when :date                 then klass.value_to_date(value)
  when :binary               then klass.binary_to_string(value)
  when :boolean              then klass.value_to_boolean(value)
  else value
  end
end
```

DIRTY LIES



```
# Casts value (which is a String) to an appropriate instance.
def type_cast(value)
  return nil if value.nil?
  return coder.load(value) if encoded?
```

```
product.name = NotAString.new
product.created_at = Time.now
product.price *= 2
```

```
diff --git a/activerecord/lib/active_record/connection_adapters/column.rb b/activerecord/lib/active_record/connection_adapters/column.rb
index 38efebe..3bab325 100644
--- a/activerecord/lib/active_record/connection_adapters/column.rb
+++ b/activerecord/lib/active_record/connection_adapters/column.rb
@@ -22,12 +22,14 @@ module ActiveRecord
       #
       # +name+ is the column's name, such as <tt>supplier_id</tt> in <tt>supplier_id int(11)</tt>.
       # +default+ is the type-casted default value, such as +new+ in <tt>sales_stage varchar(20) default 'new'</tt>.
+      # +cast_type+ is the object used for type casting and type information.
       # +sql_type+ is used to extract the column's length, if necessary. For example +60+ in
       # <tt>company_name varchar(60)</tt>.
       # It will be mapped to one of the standard Rails SQL types in the <tt>type</tt> attribute.
       # +null+ determines if this column allows +NULL+ values.
-      def initialize(name, default, sql_type = nil, null = true)
+      def initialize(name, default, cast_type, sql_type = nil, null = true)
         @name             = name
+        @cast_type        = cast_type
         @sql_type         = sql_type
         @null             = null
         @limit            = extract_limit(sql_type)
```

```
diff --git a/activerecord/lib/active_record/connection_adapters/column.rb
b/activerecord/lib/active_record/connection_adapters/column.rb
index 3bab325..0087c20 100644
--- a/activerecord/lib/active_record/connection_adapters/column.rb
+++ b/activerecord/lib/active_record/connection_adapters/column.rb
@@ -13,11 +13,13 @@ module ActiveRecord
+      delegate :type, to: :cast_type
+
       # Instantiates a new column in the table.
       #
       # +name+ is the column's name, such as <tt>supplier_id</tt> in
<tt>supplier_id int(11)</tt>.
@@ -35,7 +37,6 @@ module ActiveRecord
         @limit            = extract_limit(sql_type)
         @precision        = extract_precision(sql_type)
         @scale            = extract_scale(sql_type)
-        @type             = simplified_type(sql_type)
         @default          = extract_default(default)
         @default_function = nil
         @primary          = nil
@@ -263,40 +266,6 @@ module ActiveRecord
-
-        def simplified_type(field_type)
-          case field_type
-          when /int/i
-            :integer
-          when /float|double/i
-            :float
-          when /decimal|numeric|number/i
-            extract_scale(field_type) == 0 ? :integer : :decimal
-          when /datetime/i
-            :datetime
-          when /timestamp/i
-            :timestamp
-          when /time/i
-            :time
-          when /date/i
-            :date
-          when /clob/i, /text/i
-            :text
-          when /blob/i, /binary/i
-            :binary
-          when /char/i
-            :string
-          when /boolean/i
-            :boolean
-          end
-        end
     end
   end
```

```
diff --git a/activerecord/lib/active_record/connection_adapters/column.rb
b/activerecord/lib/active_record/connection_adapters/column.rb
index 11b2e72..f46f9af 100644
--- a/activerecord/lib/active_record/connection_adapters/column.rb
+++ b/activerecord/lib/active_record/connection_adapters/column.rb
@@ -94,28 +94,10 @@ module ActiveRecord

       # Casts value to an appropriate instance.
       def type_cast(value)
-        return nil if value.nil?
-        return coder.load(value) if encoded?
-
-        klass = self.class
-
-        case type
-        when :string, :text
-          case value
-          when TrueClass; "1"
-          when FalseClass; "0"
-          else
-            value.to_s
-          end
-        when :integer              then klass.value_to_integer(value)
-        when :float                then value.to_f
-        when :decimal              then klass.value_to_decimal(value)
-        when :datetime             then klass.string_to_time(value)
-        when :time                 then klass.string_to_dummy_time(value)
-        when :date                 then klass.value_to_date(value)
-        when :binary               then klass.binary_to_string(value)
-        when :boolean              then klass.value_to_boolean(value)
-        else value
+        if encoded?
+          coder.load(value)
+        else
+          cast_type.type_cast(value)
         end
       end
```

```
diff --git a/activerecord/lib/active_record/connection_adapters/column.rb b/activerecord/lib/active_record/connection_adapters/column.rb
index f46f9af..0f0aa91 100644
--- a/activerecord/lib/active_record/connection_adapters/column.rb
+++ b/activerecord/lib/active_record/connection_adapters/column.rb
@@ -18,7 +18,7 @@ module ActiveRecord

       alias :encoded? :coder

-      delegate :type, to: :cast_type
+      delegate :type, :text?, :number?, :binary?, to: :cast_type

       # Instantiates a new column in the table.
       #
@@ -43,16 +43,6 @@ module ActiveRecord
         @coder            = nil
       end

-      # Returns +true+ if the column is either of type string or text.
-      def text?
-        type == :string || type == :text
-      end
-
-      # Returns +true+ if the column is either of type integer, float or decimal.
-      def number?
-        type == :integer || type == :float || type == :decimal
-      end
-
       def has_default?
         !default.nil?
       end
@@ -70,10 +60,6 @@ module ActiveRecord
         end
       end

-      def binary?
-        type == :binary
-      end
-
       # Casts a Ruby value to something appropriate for writing to the database.
       # Numeric columns will typecast boolean and string to appropriate numeric
       # values.
```

```
diff --git a/activerecord/lib/active_record/connection_adapters/column.rb b/activerecord/lib/active_record/connection_adapters/column.rb
index 107b18f..a23d2bd 100644
--- a/activerecord/lib/active_record/connection_adapters/column.rb
+++ b/activerecord/lib/active_record/connection_adapters/column.rb
@@ -18,7 +18,7 @@ module ActiveRecord

       alias :encoded? :coder

-      delegate :type, :text?, :number?, :binary?, :type_cast_for_write, to: :cast_type
+      delegate :type, :klass, :text?, :number?, :binary?, :type_cast_for_write, to: :cast_type

       # Instantiates a new column in the table.
       #
@@ -47,19 +47,6 @@ module ActiveRecord
         !default.nil?
       end

-      # Returns the Ruby class that corresponds to the abstract data type.
-      def klass
-        case type
-        when :integer                    then Fixnum
-        when :float                      then Float
-        when :decimal                    then BigDecimal
-        when :datetime, :time            then Time
-        when :date                       then Date
-        when :text, :string, :binary     then String
-        when :boolean                    then Object
-        end
-      end
-
       # Casts value to an appropriate instance.
       def type_cast(value)
         if encoded?
```

```ruby
module ActiveRecord
  module Type
    class String < Value # :nodoc:
      def type
        :string
      end

      def type_cast(value)
        if value
          value.to_s
        end
      end

      def serialize(value)
        case value
        when ::Numeric, ActiveSupport::Duration then value.to_s
        when true then "t"
        when false then "f"
        else super
        end
      end
    end
  end
end
```

```ruby
class OverloadedType < ActiveRecord::Base
  create_table :overloaded_types do |t|
    t.float :overloaded_float
    t.float :unoverloaded_float
  end

  attribute :overloaded_float, Type::Integer.new
end

test "overloading types" do
  data = OverloadedType.new

  data.overloaded_float = "1.1"
  data.unoverloaded_float = "1.1"

  assert_equal 1, data.overloaded_float
  assert_equal 1.1, data.unoverloaded_float
end
```

```
attribute :overloaded_float, Type::Integer.new
```

```
attribute :overloaded_float, Type::Integer.new
```

Every DSL has a cost

Implement in small steps

```ruby
def columns
  @columns ||= connection.schema_cache.columns(table_name).map do |col|
    col = col.dup
    col.primary = (col.name == primary_key)
    col
  end
end

def columns_hash
  @columns_hash ||= Hash[columns.map { |c| [c.name, c] }]
end
```

Separate lazy from strict

```ruby
def attribute(name, cast_type)
  name = name.to_s

  self.attributes_to_define_after_schema_loads =
    attributes_to_define_after_schema_loads.merge(
      name => cast_type
    )
end

def define_attribute(name, cast_type)
  clear_caches_calculated_from_columns

  @columns = columns.map do |column|
    if column.name == name
      column.with_type(cast_type)
    else
      column
    end
  end
end

def load_schema! # :nodoc:
  super
  attributes_to_define_after_schema_loads.each do |name, type|
    define_attribute(name, type)
  end
end
```

```
diff --git a/activerecord/lib/active_record/enum.rb b/activerecord/lib/active_record/enum.rb
index f053372..b70d52a 100644
--- a/activerecord/lib/active_record/enum.rb
+++ b/activerecord/lib/active_record/enum.rb
@@ -79,6 +79,37 @@ module ActiveRecord
       super
     end

+    class EnumType < Type::Value
+      # ...
+    end
+
     def enum(definitions)
       klass = self
       definitions.each do |name, values|
@@ -90,37 +121,19 @@ module ActiveRecord
         detect_enum_conflict!(name, name.to_s.pluralize, true)
         klass.singleton_class.send(:define_method, name.to_s.pluralize) { enum_values }

-        _enum_methods_module.module_eval do
-          # def status=(value) self[:status] = statuses[value] end
-          klass.send(:detect_enum_conflict!, name, "#{name}=")
-          define_method("#{name}=") { |value|
-            # writer implementation...
-          }
-
-          # def status() statuses.key self[:status] end
-          klass.send(:detect_enum_conflict!, name, name)
-          define_method(name) { enum_values.key self[name] }
+        detect_enum_conflict!(name, name)
+        detect_enum_conflict!(name, "#{name}=")

-          # def status_before_type_cast() statuses.key self[:status] end
-          klass.send(:detect_enum_conflict!, name, "#{name}_before_type_cast")
-          define_method("#{name}_before_type_cast") { enum_values.key self[name] }
+        attribute name, EnumType.new(name, enum_values)
@@ -138,25 +151,7 @@ module ActiveRecord
     private
       def _enum_methods_module
         @_enum_methods_module ||= begin
-          mod = Module.new do
-            private
-              def save_changed_attribute(attr_name, old)
-                # significant source of bugs
-              end
-          end
+          mod = Module.new
           include mod
           mod
         end
```

Make your internal APIs as pleasant to use as your external APIs

```ruby
def decorate_matching_attribute_types(matcher, decorator_name, &block)
  self.attribute_type_decorations = attribute_type_decorations.merge(decorator_name => [matcher, block])
end

private

def load_schema!
  super
  attribute_types.each do |name, type|
    decorated_type = attribute_type_decorations.apply(name, type)
    define_attribute(name, decorated_type)
  end
end
```

```ruby
matcher = ->(name, type) { create_time_zone_conversion_attribute?(name, type) }
decorate_matching_attribute_types(matcher, :_time_zone_conversion) do |type|
  TimeZoneConverter.new(type)
end
```

```ruby
matcher = ->(name, _) { name == attr_name }
decorate_matching_attribute_types(matcher, :"_serialize_#{attr_name}") do |type|
  Type::Serialized.new(type, coder)
end
```

```ruby
def decorate_attribute_type(attr_name, decorator_name, &block)
  matcher = ->(name, _) { name == attr_name.to_s }
  key = "_#{column_name}_#{decorator_name}"
  decorate_matching_attribute_types(matcher, key, &block)
end
```

```ruby
def serialize(attr_name, class_name_or_coder = Object)
  coder = if [:load, :dump].all? { |x| class_name_or_coder.respond_to?(x) }
            class_name_or_coder
          else
            Coders::YAMLColumn.new(class_name_or_coder)
          end

  decorate_attribute_type(attr_name, :serialize) do |type|
    Type::Serialized.new(type, coder)
  end
end
```

```ruby
module ActiveRecord
  module Type
    class Serialized < DelegateClass(Type::Value) # :nodoc:
      attr_reader :subtype, :coder

      def initialize(subtype, coder)
        @subtype = subtype
        @coder = coder
        super(subtype)
      end

      def deserialize(value)
        if default_value?(value)
          value
        else
          coder.load(super)
        end
      end

      def serialize(value)
        return if value.nil?
        unless default_value?(value)
          super coder.dump(value)
        end
      end

      private

      def default_value?(value)
        value == coder.load(nil)
      end
    end
  end
end
```

Make your API universal

There should be one canonical way to access things

```ruby
def load_schema!
  @columns_hash = connection.schema_cache.columns_hash(table_name)
  @columns_hash.each do |name, column|
    define_attribute(
      name,
      connection.lookup_cast_type_from_column(column),
      default: column.default,
      user_provided_default: false
    )
  end
end
```

```ruby
class Attribute # :nodoc:
  attr_reader :name, :value_before_type_cast, :type

  def initialize(name, value_before_type_cast, type)
    @name = name
    @value_before_type_cast = value_before_type_cast
    @type = type
  end

  def value
    # `defined?` is cheaper than `||=` when we get back falsy values
    @value = original_value unless defined?(@value)
    @value
  end

  def original_value
    type_cast(value_before_type_cast)
  end

  def value_for_database
    type.serialize(value)
  end

  def type_cast(value)
    type.cast(value)
  end
end
```

```ruby
class AttributeSet # :nodoc:
  def initialize(attributes)
    @attributes = attributes
  end

  def [](name)
    attributes[name] || Attribute.null(name)
  end

  def []=(name, value)
    attributes[name] = value
  end

  def values_before_type_cast
    attributes.transform_values(&:value_before_type_cast)
  end

  def to_hash
    initialized_attributes.transform_values(&:value)
  end
  alias_method :to_h, :to_hash

  def key?(name)
    attributes.key?(name) && self[name].initialized?
  end

  def keys
    attributes.each_key.select { |name| self[name].initialized? }
  end

  def fetch_value(name, &block)
    self[name].value(&block)
  end

  def write_from_database(name, value)
    attributes[name] = self[name].with_value_from_database(value)
  end

  def write_from_user(name, value)
    attributes[name] = self[name].with_value_from_user(value)
  end

  protected

  attr_reader :attributes

  private

  def initialized_attributes
    attributes.select { |_, attr| attr.initialized? }
  end
end
```

```ruby
def read_attribute(attr_name, &block)
  @attributes.fetch_value(attr_name.to_s, &block)
end

def attributes
  @attributes.to_hash
end

def _field_changed?(attr, old_value)
  @attributes[attr].changed_from?(old_value)
end
```

Prefer Composition over Inheritance

Objects have an interface, which lets you infer what behavior can be affected.

Given that

```
Product.belongs_to :user
```

when I call

```
product.user.name = "Changed"
product.save
```

Did the user's name change in the database?

Have a contract

```
assert_equals model.attribute, model.tap(&:save).reload.attribute

model.attribute = model.attribute
refute model.changed?

refute Model.new.changed?

assert_equal model, Model.find_by(attribute: model.attribute)
```

I don't know how to end this talk

Conclusions

Integrated Systems

Synergy

Please ask me questions now

Sean Griffin

- Twitter: @sgrif
- Github: sgrif
- Email: sean@thoughtbot.com
- Podcast: bikeshed.fm