WeTransfer

Tools to move ideas

🔪 ✂️

Perfection is the enemy of the good

We all succumb to the desire of crafting *perfect* code

It is perfect as long as we wrote it

...i.e. given the constraints of

— Style that was available to us in a *personal development state*

— Specific purpose of a *project*

— A specific *deadline*

Ruby is an open runtime

- Everything is mutable
- Everything can be changed
- Benign intent is assumed

Cultures differ

Example: serving HTTP ranges in Go

Objective: given an HTTP `Range:` request header, return sections of the resource to the client. For example, 3 `File` references glued together in sequence.

```
func ServeContent(w ResponseWriter,
  req *Request,
  name string,
  modtime time.Time,
  content io.ReadSeeker)
```

Customisation points

— The entire HTTP request ❓

— What will be served (exposed resource) ✅

— What will be the `Content-Disposition` filename ✅

— What will be the `Last-Modified` filename ✅

— Where the response will be written to ✅

But wait.

— How much data was served 🚫

— Which ranges were requested 🚫

— What is the `Content-Disposition` intent (`inline/attachment`) 🚫

— Any kind of error handling 🚫

— Any kind of error handling 🚫

`ServeContent` does not return `error`, neither does it propagate it.

Parts of `ServeContent`

The function is multiple hundreds of lines long.

- HTTP `Range` header parsing (depends on the seekable size of `content`)
- Serving multipart byte ranges (`multipart/byte-range` response type)
- Handling errors when doing `io.Copy`

All of the above is private to the `http` package.

- Not possible to reuse.
- Not possible to redefine.
- Not possible to replace with own implementation.

Denying choices is a perfectionist stance

You assume that *exactly* your implementation is *exactly* what will work for your user.

bradfitz commented on 29 Jan 2015 Member

Your minor convenience isn't worth the cost of all other Go programmer's increased cognitive load required by having more stuff in the `net/http` package to read and understand the difference between.

Sorry, we're not adding this. There are ways to do this already. Unless a large number/percentage of people needed this, it's not worth the cost of adding it.

golang locked and limited conversation to collaborators on 25 Jun 2016

Using unexported functions/types from stdlib in Go

Asked 8 months ago Active 8 months ago Viewed 54 times

-1

Disclaimer: yes I know that this is not "supposed to be done" and "use interface composition and delegation" and "the authors of the language know better". However, I am confronted with a choice of either copy-pasting from the standard library and creating my own packages, or doing what I am asking. So please do not reply with "What you want to do is wrong, you are a bad dev and you should feel bad."

Wikileaks To Leak 5000 Open Source Java Projects With All That Private/Final Bullshit Removed

http://steve-yegge.blogspot.com/2010/07/wikileaks-to-leak-5000-open-source-java.html

Agile Java Developer Johnnie Garza of Irvine, CA condemns the move. "They have no right to do this. Open Source does not mean the source is somehow 'open'. That's my code, not theirs. If I make something private, it means that no matter how desperately you need to call it, I should be able to prevent you from doing so, even long after I've gone to the grave."

Open / closed

Should be open for *extension,* closed for *modification.*

...you can't predict extension

— And in your drive for perfection and minimalism you will underdo it.

With open-runtime languages you are treated as a grown up

- With the help of some sharp tools
- Extension is sometimes indistinguishable from modification

Normally considered under 🔪

— Debuggers

— `strace`

— Observability tools/injections

— Metrics

Just touch the code already

It don't bite. It is not "their" code. It is not sacred just because DHH wrote it.

Tool 1: `bundle open`

```
documents-app (master) $
documents-app (master) $ bundle open activerecord
```

will open the ActiveRecord gem in your editor.

Case: bridging the Apartment gem and ActiveStorage

- Apartment is a gem for multitenancy, offering one database to one customer
- ActiveStorage handles file uploads and attachments

- ActiveStorage saves in the default database
- It bypasses the Apartment database switch in some cases
- It is a Rails Engine, so it has its own controllers
- It stores all files in a single, shared namespace

— `ActiveStorage::Attachment` saved in the correct database

— `ActiveStorage::Blob` saved in the *main* database, sometimes.

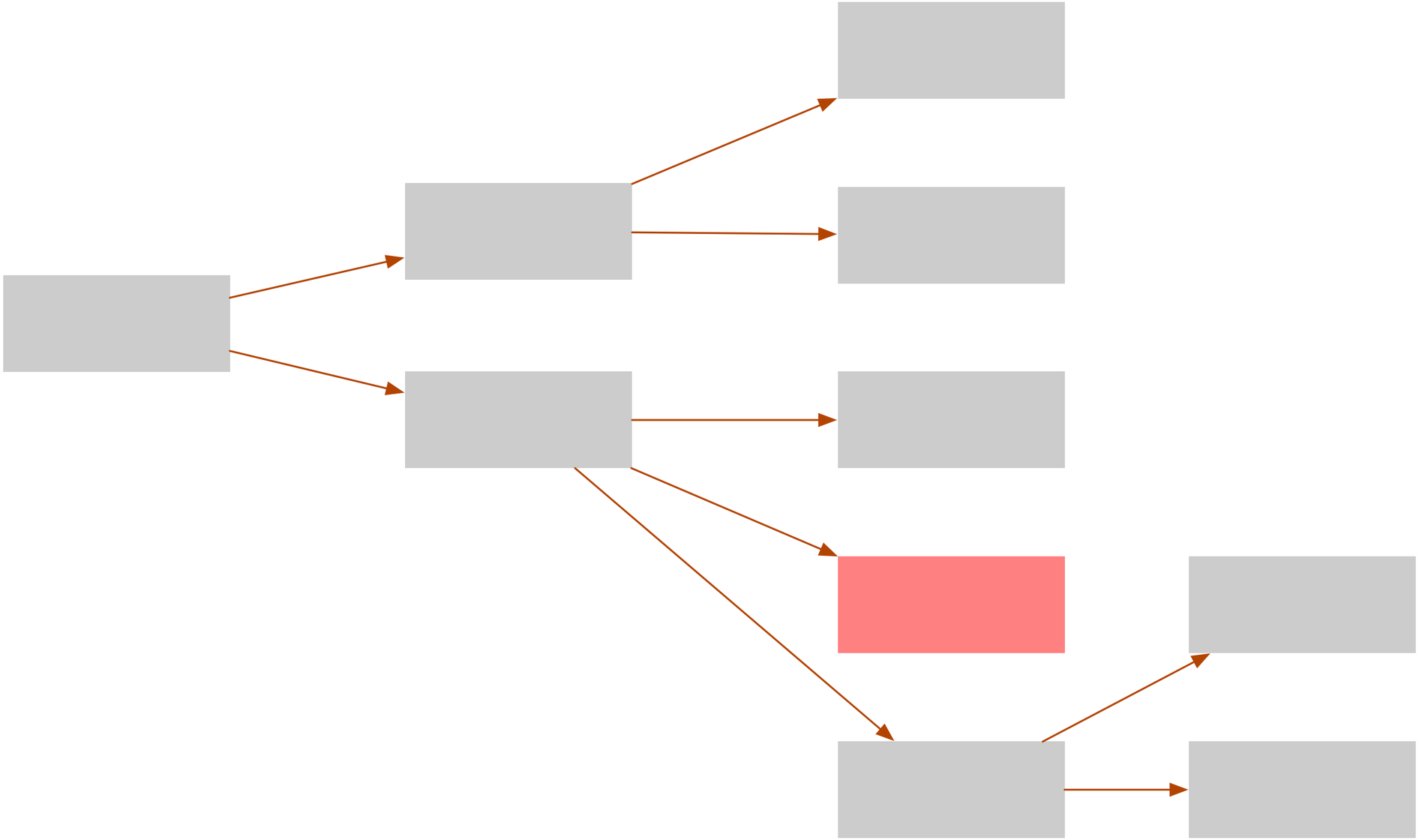
Tool 2: poking at things

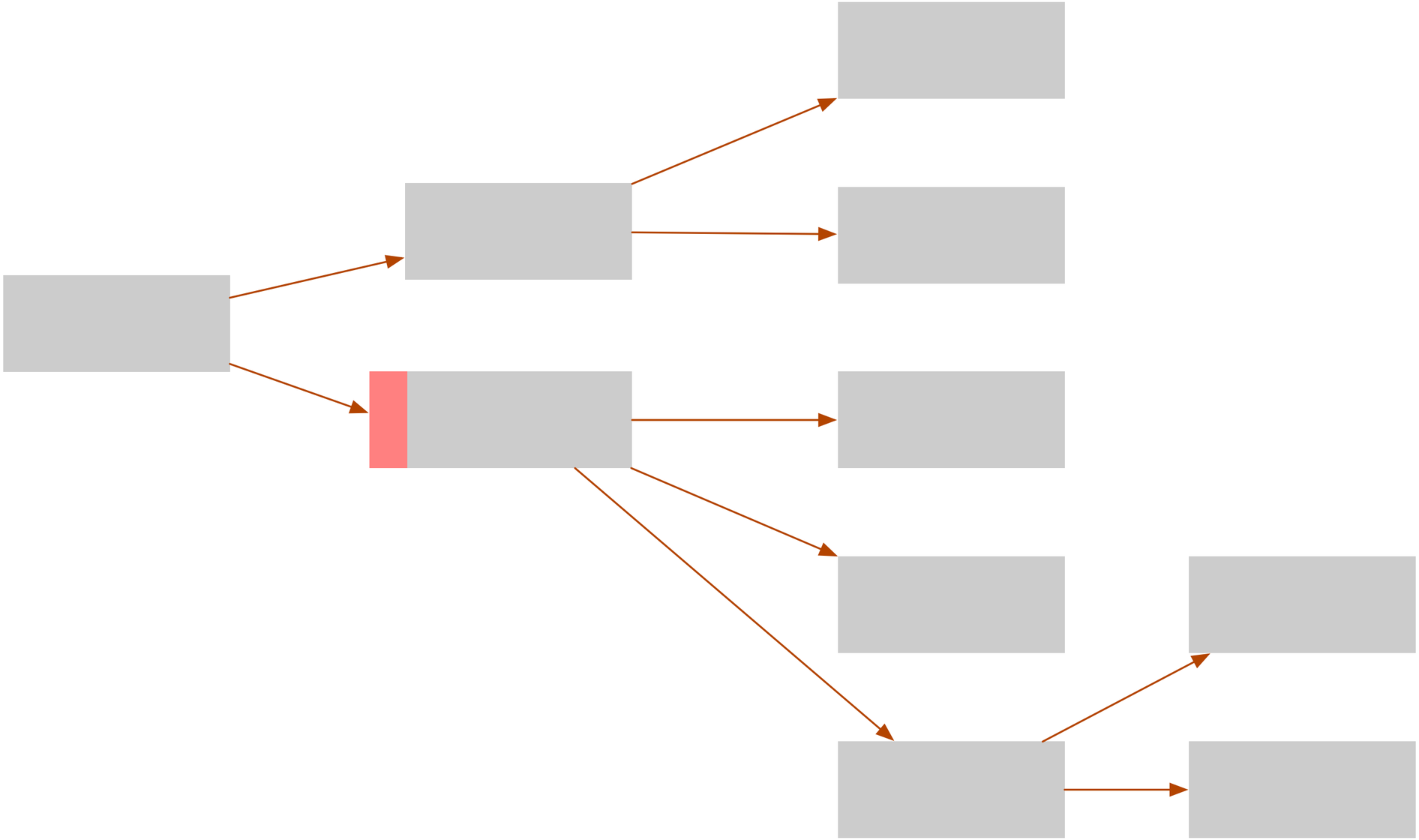
```
[3] pry(main)> ActiveStorage::Blob.public_instance_methods.grep(/signed/)
=> [:signed_id, :to_signed_global_id]
[4] pry(main)> ActiveStorage::Blob.methods.grep(/signed/)
=> [:find_signed]
[5] pry(main)> location = ActiveStorage::Blob.method(:find_signed).source_location
=> [".../activestorage-5.2.3/app/models/active_storage/blob.rb", 46]
[3] pry(main)> `code --goto #{location[0]}:#{location[1]}`
```

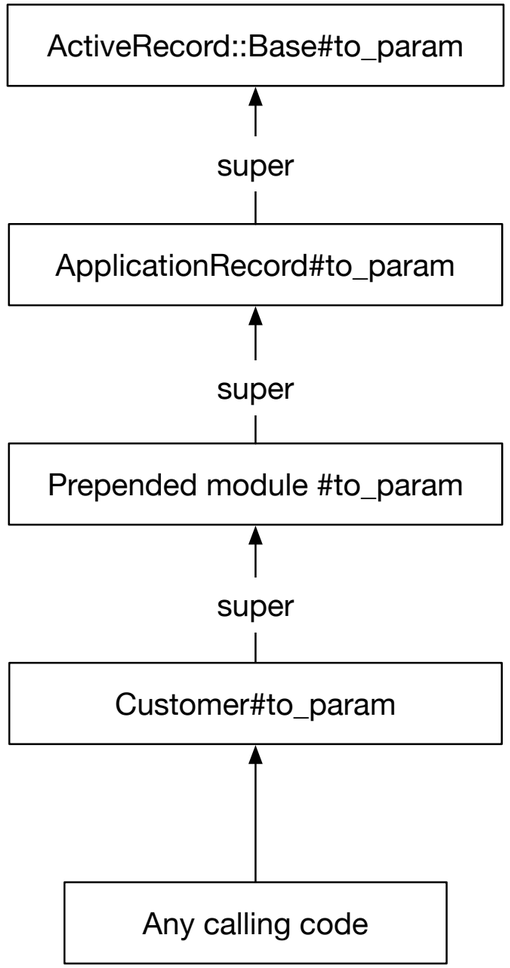
Let's see

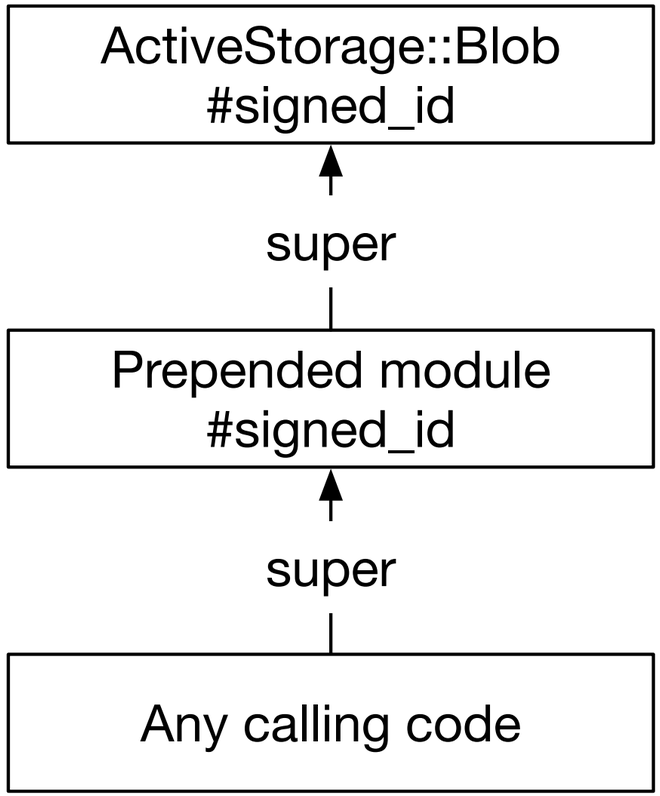
All lookups of the `Blob` go through signed identifiers.

```
class << self
  ...

  # You can used the signed ID of a blob to refer to it on the client side without fear of tampering.
  # This is particularly helpful for direct uploads where the client-side needs to refer to the blob
  # that was created ahead of the upload itself on form submission.
  #
  # The signed ID is also used to create stable URLs for the blob through the BlobsController.
  def find_signed(id)
    find ActiveStorage.verifier.verify(id, purpose: :blob_id)
  end

  ...
end
```






ActiveRecord::Base#to_param
super
ApplicationRecord#to_param
super
Prepended module #to_param
super
Customer#to_param
Any calling code


ActiveStorage::Blob
#signed_id
super
Prepended module
#signed_id
super
Any calling code

Tool 3: `Module#prepend`

Allows you to insert *your functionality* between the caller and the callee

```
module ApartmentAwareSignedId
  def find_signed(id)
    tenant, id = ActiveStorage.verifier.verify(signed_id_with_tenant_prefix, purpose: :blob_id_with_tenant)
    Apartment::Tenant.switch(tenant) { find(id) }
  end

  def signed_id
    ActiveStorage.verifier.generate([Apartment::Tenant.current, id], purpose: :blob_id_with_tenant)
  end
end

ActiveStorage::Blob.singleton_class.prepend(ApartmentAwareSignedId)
ActiveStorage::Blob.prepend(ApartmentAwareSignedId)
```

— `tkjhug789r` exists in `tenant-1`

— `tkjhug789r` does not exist in `tenant-2` and gets created...

overwriting the one that belongs to `tenant-1`.

```ruby
module PrefixKeyWithTenantToken
  # Prefix all generated blob keys with the tenant. Do not
  # use slash as a delimiter because it needs different escaping
  # depending on the storage service adapter - in some cases it
  # might be significant, in other cases it might get escaped as a path
  # component etc.
  def generate_unique_secure_token
    tenant_slug = Apartment::Tenant.current.split('_').last
    "#{tenant_slug}-#{super}" #=> "tenant1-cbf781tr"
  end
end

ActiveStorage::Blob.singleton_class.prepend(PrefixKeyWithTenantToken)
```

Tool 4: Rails reloader

```
# ActiveStorage can get reloaded as well, and once it does get reloaded
# our injected modules will be lost. We need to ensure the patch is applied
# on every reload.
ActiveSupport::Reloader.to_prepare do
  # Install the prefixed key patch and the prefixed signed ID patches.
  ActiveStorage::Blob.singleton_class.send(:prepend, PrefixKeyWithToken)
  ActiveStorage::Blob.singleton_class.send(:prepend, FindSignedWithTenant)
  ActiveStorage::Blob.prepend(PrefixedSignedId)
end
```

initializers

- accept_header_fix.rb
- app_revision_environment_variable.rb
- assets.rb
- backtrace_silencers.rb
- client_ip_injection.rb
- cookies_serializer.rb
- cors.rb
- delay_method.rb
- departure.rb
- devise.rb
- disable_nullable_mysql_primary_keys.rb
- domain_name_uniqueness_validator.rb
- ensure_redis_available.rb
- filter_parameter_logging.rb
- host_authorization_configuration.rb
- host_ident.rb
- i18n.rb
- image_vise_configuration.rb
- inflections.rb
- instrument_worker.rb
- measurometer_init.rb
- migrator_notifications.rb
- mime_types.rb
- notifications.rb
- nu_billing_events_listeners.rb
- patron_default_timeouts.rb
- patron_instrumentation.rb
- payment_provider_api_config.rb
- prawn_document.rb
- quiet_assets.rb
- rails_email_preview.rb
- raindrops_middleware.rb
- record_mailer_class_and_method_in_headers.rb
- redirect_invalid_subdomains.rb
- request_uri_cleanup.rb
- sanitize_varchar_in_records.rb
- secret_token.rb
- server_side_validations.rb
- setup_mail.rb
- smtp_error_capture.rb
- sqlite_sqewer_queue.rb
- stripe.rb
- strong_parameters.rb
- suppress_unacceptable_format.rb
- tag_appsignal_txn_with_rails_version.rb
- trust_aws_ip_as_proxy.rb
- vat_number_validator.rb
- webpack.rb
- wrap_parameters.rb

Database Migrations APP 11:34

Starting `frontend` DB migrations **up** on **production** towards version 20191001141821
This corresponds to the migration `DropResetTransfers`

Migrations that will be applied along the way:

```
20191001141821 Drop reset transfers
```

The tables have turned. Applied `frontend` DB migrations **up** on **production** nearly instantaneously:

```
20191001141821 Drop reset transfers
```

How does it work?

```
ActiveSupport.on_load(:active_record) do
  ActiveRecord::Migrator.prepend(MigrationNotification::MigratorMixin)
end
```

Your perfection is not the same as other people's perfection

— Clever tricks

— Monkey patches

— Spooky action at a distance

— You never know what will actually run

— The user will always break what was so carefully built

— Juniors will never understand this code

Rails is built on this, using sharp tools.

Other people's code is not sacred

Randomized with seed 22579

Kelder

ActiveStorage::DirectUploadsController overrides

performs the original action if there is no "signed_tenant_name"	1.17847s
switches into the tenant before returning the direct upload URL	0.90825s

ActiveStorage::Blob overrides

stay intact after app reload (PENDING: Requires Rails.application.config.cache_classes to be false)	
returns an altered payload from #signed_id and can use it to find itself later via .find_signed	0.29484s
prefixes the "key" attribute with the last component of the current tenant database name	0.74835s

ActiveStorage::Service::DiskService override

stores the files in a prefixed subdirectory	1.26323s
does not tenant-prefix a key which does not contain a "-"	0.21359s

ActiveStorage::BlobsController overrides

with the main tenant, does not suffix the blob key	0.89166s
returns the correct blob even though the blob URL does not contain the query string parameter	0.26229s

With dynamic languages, use sharp tools.

https://www.youtube.com/watch?v=3DEA8njVTIc&t=2158

Bonus 1: Inline Bundler

```ruby
require "bundler/inline"

gemfile do
  source 'https://rubygems.org'
  gem 'most_perfect_library', '1.1.13'
  gem 'rspec', '~> 4`
end

RSpec.describe 'Buggy library' do
  it 'wargs when asked to' do
    expect(MostPerfect.warg).to eq("WARG!!")
    #=> This clearly does not warg...
  end
end
```

Bonus 2: Inline Rails

```
# ...
  gem 'rails'
  gem 'rspec'
end

MY_RACK_APP = ->(env) { [200, {}, ["Ohai there!"]]}

class RailsAppDemonstratingIssue < Rails::Application
  secrets.secret_token    = "secret_token"
  secrets.secret_key_base = "secret_key_base"

  config.logger = Logger.new($stderr)
  Rails.logger = config.logger

  routes.draw do
    mount MY_RACK_APP => '/'
  end
end

# Then use rack-test to perform requests
RailsAppDemonstratingIssue.to_app.call(your_rack_env)
```

🙏