application error

PGError at /

could not connect to server: Connection refused Is the server running on host "dns17.i.zerigo.net" and accepting TCP/IP connections on port 5432?

- **file:** `postgresql_adapter.rb`
- **location:** `initialize`
- **line:** 941

BACKTRACE

(expand)

JUMP TO: GET POST COOKIES ENV

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/postgresql_adapter.rb` in **`initialize`**

-
  ```
  934. private
  935. # The internal PostgreSQL identifier of the money data type.
  936. MONEY_COLUMN_TYPE_OID = 790 #:nodoc:
  937.
  938. # Connects to a PostgreSQL server and sets up the adapter depending on the
  939. # connected server's characteristics.
  940. def connect
  941. @connection = PGconn.connect(*@connection_parameters)
  942. PGconn.translate_results = false if PGconn.respond_to?(:translate_results=)
  943.
  944. # Ignore async_exec and async_query when using postgres-pr.
  945. @async = @config[:allow_concurrency] && @connection.respond_to?(:async_exec)
  946.
  947. # Money type has a fixed precision of 10 in PostgreSQL 8.2 and below, and as of
  948. # PostgreSQL 8.3 it has a fixed precision of 19.
       PostgreSQLColumn.extract_precision
  ```
- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/postgresql_adapter.rb` in **`connect`**

-
  ```
  934. private
  935. # The internal PostgreSQL identifier of the money data type.
  936. MONEY_COLUMN_TYPE_OID = 790 #:nodoc:
  ```

```
# Connects to a PostgreSQL server and sets up the adapter depending on the
# connected server's characteristics.
def connect
@connection = PGconn.connect(*@connection_parameters)
PGconn.translate_results = false if PGconn.respond_to?(:translate_results=)

# Ignore async_exec and async_query when using postgres-pr.
@async = @config[:allow_concurrency] && @connection.respond_to?(:async_exec)

# Money type has a fixed precision of 10 in PostgreSQL 8.2 and below, and as of
# PostgreSQL 8.3 it has a fixed precision of 19.
PostgreSQLColumn.extract_precision
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/postgresql_adapter.rb` in **connect**
-
```
private
# The internal PostgreSQL identifier of the money data type.
MONEY_COLUMN_TYPE_OID = 790 #:nodoc:

# Connects to a PostgreSQL server and sets up the adapter depending on the
# connected server's characteristics.
def connect
@connection = PGconn.connect(*@connection_parameters)
PGconn.translate_results = false if PGconn.respond_to?(:translate_results=)

# Ignore async_exec and async_query when using postgres-pr.
@async = @config[:allow_concurrency] && @connection.respond_to?(:async_exec)

# Money type has a fixed precision of 10 in PostgreSQL 8.2 and below, and as of
# PostgreSQL 8.3 it has a fixed precision of 19.
PostgreSQLColumn.extract_precision
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/postgresql_adapter.rb` in **initialize**
-
```
end

# Initializes and connects a PostgreSQL adapter.
def initialize(connection, logger, connection_parameters, config)
super(connection, logger)
@connection_parameters, @config = connection_parameters, config

connect
end

# Is this connection alive and ready for queries?
def active?
if @connection.respond_to?(:status)
@connection.status == PGconn::CONNECTION_OK
else
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/postgresql_adapter.rb` in **new**
-
```
database = config[:database]
else
raise ArgumentError, "No database specified. Missing argument: database."
```

```
end

# The postgres drivers don't allow the creation of an unconnected PGconn object,
# so just pass a nil connection object for the time being.
ConnectionAdapters::PostgreSQLAdapter.new(nil, logger, [host, port, nil, nil,
database, username, password], config)
end
end

module ConnectionAdapters
class TableDefinition
def xml(*args)
options = args.extract_options!
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/postgresql_adapter.rb` in **`postgresql_connection`**

-
  ```
  database = config[:database]
  else
  raise ArgumentError, "No database specified. Missing argument: database."
  end

  # The postgres drivers don't allow the creation of an unconnected PGconn object,
  # so just pass a nil connection object for the time being.
  ConnectionAdapters::PostgreSQLAdapter.new(nil, logger, [host, port, nil, nil,
  database, username, password], config)
  end
  end

  module ConnectionAdapters
  class TableDefinition
  def xml(*args)
  options = args.extract_options!
  ```
- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb` in **`send`**

-
  ```
  end

  synchronize :clear_reloadable_connections!, :verify_active_connections!,
  :connected?, :disconnect!, :with => :@connection_mutex

  private
  def new_connection
  ActiveRecord::Base.send(spec.adapter_method, spec.config)
  end

  def current_connection_id #:nodoc:
  Thread.current.object_id
  end

  # Remove stale threads from the cache.
  ```
- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb` in **`new_connection`**

-
  ```
  end
  ```

```

synchronize :clear_reloadable_connections!, :verify_active_connections!,
:connected?, :disconnect!, :with => :@connection_mutex

private
def new_connection
ActiveRecord::Base.send(spec.adapter_method, spec.config)
end

def current_connection_id #:nodoc:
Thread.current.object_id
end

# Remove stale threads from the cache.
```

- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb in **checkout_new_connection**

-
  ```
  next unless cache.has_key?(key)
  block.call(key, cache[key])
  cache.delete(key)
  end
  end
  
  def checkout_new_connection
  c = new_connection
  @connections << c
  checkout_and_verify(c)
  end
  
  def checkout_existing_connection
  c = (@connections - @checked_out).first
  checkout_and_verify(c)
  ```
- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb in **checkout**

-
  ```
  def checkout
  # Checkout an available connection
  @connection_mutex.synchronize do
  loop do
  conn = if @checked_out.size < @connections.size
  checkout_existing_connection
  elsif @connections.size < @size
  checkout_new_connection
  end
  return conn if conn
  # No connections available; wait for one
  if @queue.wait(@timeout)
  next
  else
  # try looting dead threads
  ```
- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb in **loop**

-
  ```
  #
  # Raises:
  ```

```
179. # - ConnectionTimeoutError: no connection can be obtained from the pool
180. # within the timeout period.
181. def checkout
182. # Checkout an available connection
183. @connection_mutex.synchronize do
184. loop do
185. conn = if @checked_out.size < @connections.size
186. checkout_existing_connection
187. elsif @connections.size < @size
188. checkout_new_connection
189. end
190. return conn if conn
191. # No connections available; wait for one
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb` in **checkout**

-
  ```
  177. #
  178. # Raises:
  179. # - ConnectionTimeoutError: no connection can be obtained from the pool
  180. # within the timeout period.
  181. def checkout
  182. # Checkout an available connection
  183. @connection_mutex.synchronize do
  184. loop do
  185. conn = if @checked_out.size < @connections.size
  186. checkout_existing_connection
  187. elsif @connections.size < @size
  188. checkout_new_connection
  189. end
  190. return conn if conn
  191. # No connections available; wait for one
  ```
- `/usr/lib/ruby/1.8/monitor.rb` in **synchronize**

-
  ```
  235. # Enters exclusive section and executes the block. Leaves the exclusive
  236. # section automatically when the block exits. See example under
  237. # +MonitorMixin+.
  238. #
  239. def mon_synchronize
  240. mon_enter
  241. begin
  242. yield
  243. ensure
  244. mon_exit
  245. end
  246. end
  247. alias synchronize mon_synchronize
  248.
  249. #
  ```
- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb` in **checkout**

-
  ```
  176. # Returns: an AbstractAdapter object.
  177. #
  178. # Raises:
  179. # - ConnectionTimeoutError: no connection can be obtained from the pool
  180. # within the timeout period.
  ```

```
def checkout
# Checkout an available connection
@connection_mutex.synchronize do
loop do
conn = if @checked_out.size < @connections.size
checkout_existing_connection
elsif @connections.size < @size
checkout_new_connection
end
return conn if conn
```

- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb in **connection**

-
  ```
  #
  # #connection can be called any number of times; the connection is
  # held in a hash keyed by the thread id.
  def connection
  if conn = @reserved_connections[current_connection_id]
  conn
  else
  @reserved_connections[current_connection_id] = checkout
  end
  end

  # Signal that the thread is finished with the current connection.
  # #release_connection releases the connection-thread association
  # and returns the connection to the pool.
  def release_connection
  ```
- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_pool.rb in **retrieve_connection**

-
  ```

  # Locate the connection of the nearest super class. This can be an
  # active or defined connection: if it is the latter, it will be
  # opened and set as the active connection for the class it was defined
  # for (not necessarily the current class).
  def retrieve_connection(klass) #:nodoc:
  pool = retrieve_connection_pool(klass)
  (pool && pool.connection) or raise ConnectionNotEstablished
  end

  # Returns true if a connection that's accessible to this class has
  # already been opened.
  def connected?(klass)
  conn = retrieve_connection_pool(klass)
  conn ? conn.connected? : false
  ```
- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_specification.rb in **retrieve_connection**

-
  ```
  end

  def connection_pool
  connection_handler.retrieve_connection_pool(self)
  end
  ```

```

def retrieve_connection
connection_handler.retrieve_connection(self)
end

# Returns true if +ActiveRecord+ is connected.
def connected?
connection_handler.connected?(self)
end

```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/connection_adapters/abstract/connection_specification.rb` in **connection**
-
```
ActiveSupport::Deprecation.warn("ActiveRecord::Base.verification_timeout= has been deprecated and no longer has any effect. Please remove all references to verification_timeout=.")
end

# Returns the connection currently associated with the class. This can
# also be used to "borrow" the connection to do database work unrelated
# to any of the specific Active Records.
def connection
retrieve_connection
end

def connection_pool
connection_handler.retrieve_connection_pool(self)
end

def retrieve_connection
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/base.rb` in **quoted_table_name**
-
```
connection = self.class.connection
attributes.keys.collect do |column_name|
connection.quote_column_name(column_name)
end
end

def self.quoted_table_name
self.connection.quote_table_name(self.table_name)
end

def quote_columns(quoter, hash)
hash.inject({}) do |quoted, (name, value)|
quoted[quoter.quote_column_name(name)] = value
quoted
end
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/base.rb` in **construct_finder_sql**
-
```
'*'
end
end

```

```
def construct_finder_sql(options)
scope = scope(:find)
sql = "SELECT #{options[:select] || (scope && scope[:select]) ||
default_select(options[:joins] || (scope && scope[:joins]))} "
sql << "FROM #{options[:from] || (scope && scope[:from]) || quoted_table_name} "

add_joins!(sql, options[:joins], scope)
add_conditions!(sql, options[:conditions], scope)

add_group!(sql, options[:group], options[:having], scope)
add_order!(sql, options[:order], scope)
add_limit!(sql, options, scope)
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/base.rb` in **`find_every`**

```

def find_every(options)
include_associations = merge_includes(scope(:find, :include), options[:include])

if include_associations.any? && references_eager_loaded_tables?(options)
records = find_with_associations(options)
else
records = find_by_sql(construct_finder_sql(options))
if include_associations.any?
preload_associations(records, include_associations)
end
end

records.each { |record| record.readonly! } if options[:readonly]

```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/base.rb` in **`find_initial`**

```

"(#{segments.join(') AND (')})" unless segments.empty?
end

private
def find_initial(options)
options.update(:limit => 1)
find_every(options).first
end

def find_last(options)
order = options[:order]

if order
order = reverse_sql_order(order)
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/base.rb` in **`find`**

```
# end
def find(*args)
options = args.extract_options!
validate_find_options(options)
set_readonly_option!(options)

case args.first
when :first then find_initial(options)
when :last then find_last(options)
when :all then find_every(options)
else find_from_ids(args, options)
end
end

# A convenience wrapper for <tt>find(:first, *args)</tt>. You can pass in all the
```

- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/named_scope.rb in **send**
-
```
with_scope({:find => proxy_options, :create => proxy_options[:conditions].is_a?(Hash) ? proxy_options[:conditions] : {}}, :reverse_merge) do
method = :new if method == :build
if current_scoped_methods_when_defined && !scoped_methods.include?(current_scoped_methods_when_defined)
with_scope current_scoped_methods_when_defined do
proxy_scope.send(method, *args, &block)
end
else
proxy_scope.send(method, *args, &block)
end
end
end
end

def load_found
@found = find(:all)
```

- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/named_scope.rb in **method_missing**
-
```
with_scope({:find => proxy_options, :create => proxy_options[:conditions].is_a?(Hash) ? proxy_options[:conditions] : {}}, :reverse_merge) do
method = :new if method == :build
if current_scoped_methods_when_defined && !scoped_methods.include?(current_scoped_methods_when_defined)
with_scope current_scoped_methods_when_defined do
proxy_scope.send(method, *args, &block)
end
else
proxy_scope.send(method, *args, &block)
end
end
end
end

def load_found
@found = find(:all)
```

- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/base.rb in **with_scope**
-
```
end
hash
end
end

self.scoped_methods << method_scoping
begin
yield
```

```
ensure
self.scoped_methods.pop
end
end

# Works like with_scope, but discards any nested properties.
def with_exclusive_scope(method_scoping = {}, &block)
```

- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/named_scope.rb in **__send__**

-
  ```
  [].methods.each do |m|
  unless m =~ /^__/ || NON_DELEGATE_METHODS.include?(m.to_s)
  delegate m, :to => :proxy_found
  end
  end

  delegate :scopes, :with_scope, :scoped_methods, :to => :proxy_scope

  def initialize(proxy_scope, options, &block)
  options ||= {}
  [options[:extend]].flatten.each { |extension| extend extension } if options[:extend]
  extend Module.new(&block) if block_given?
  unless (Scope === proxy_scope || ActiveRecord::Associations::AssociationCollection === proxy_scope)
  @current_scoped_methods_when_defined = proxy_scope.send(:current_scoped_methods)
  end
  ```
- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/named_scope.rb in **with_scope**

-
  ```
  [].methods.each do |m|
  unless m =~ /^__/ || NON_DELEGATE_METHODS.include?(m.to_s)
  delegate m, :to => :proxy_found
  end
  end

  delegate :scopes, :with_scope, :scoped_methods, :to => :proxy_scope

  def initialize(proxy_scope, options, &block)
  options ||= {}
  [options[:extend]].flatten.each { |extension| extend extension } if options[:extend]
  extend Module.new(&block) if block_given?
  unless (Scope === proxy_scope || ActiveRecord::Associations::AssociationCollection === proxy_scope)
  @current_scoped_methods_when_defined = proxy_scope.send(:current_scoped_methods)
  end
  ```
- /var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/named_scope.rb in **method_missing**

-
  ```
  end

  private
  def method_missing(method, *args, &block)
  if scopes.include?(method)
  scopes[method].call(self, *args)
  else
  ```

```
with_scope({:find => proxy_options, :create => proxy_options[:conditions].is_a?
(Hash) ? proxy_options[:conditions] : {}}, :reverse_merge) do
method = :new if method == :build
if current_scoped_methods_when_defined && !scoped_methods.include?
(current_scoped_methods_when_defined)
with_scope current_scoped_methods_when_defined do
proxy_scope.send(method, *args, &block)
end
else
proxy_scope.send(method, *args, &block)
```

- `/var/lib/gems/1.8/gems/activerecord-2.3.11/lib/active_record/named_scope.rb` in **`first`**

-
  ```
  load_found; self
  end

  def first(*args)
  if args.first.kind_of?(Integer) || (@found && !args.first.kind_of?(Hash))
  proxy_found.first(*args)
  else
  find(:first, *args)
  end
  end

  def last(*args)
  if args.first.kind_of?(Integer) || (@found && !args.first.kind_of?(Hash))
  proxy_found.last(*args)
  else
  ```
- `./url_redirector.rb` in **`GET (?-mix:(\/.*))`**

-
  ```
  not_found if c_info[1] == 404

  cached = true
  target_url = c_info[0]
  else
  cached = false
  begin
  unless target = Record.by_url(req_host).first
  # try wildcard lookup
  req2 = req_host.gsub(/^[a-z0-9-]+\./, '*.')
  target = Record.by_url(req2).first
  end
  rescue ActiveRecord::StatementInvalid, RuntimeError => e
  # captures loss of DB connection and forces an exit. unicorn will see this
  # and relaunch the process, which will reestablish the DB connection. DB
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`call`**

-
  ```
  define_method(method_name, &block)
  unbound_method = instance_method method_name
  pattern, keys = compile(path)
  conditions, @conditions = @conditions, []
  remove_method method_name

  [ block.arity != 0 ?
  proc { unbound_method.bind(self).call(*@block_params) } :
  proc { unbound_method.bind(self).call },
  pattern, keys, conditions ]
  ```

```
end

def compile(path)
keys = []
if path.respond_to? :to_str
```

- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`compile!`**

-
  ```
  define_method(method_name, &block)
  unbound_method = instance_method method_name
  pattern, keys = compile(path)
  conditions, @conditions = @conditions, []
  remove_method method_name

  [ block.arity != 0 ?
  proc { unbound_method.bind(self).call(*@block_params) } :
  proc { unbound_method.bind(self).call },
  pattern, keys, conditions ]
  end

  def compile(path)
  keys = []
  if path.respond_to? :to_str
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`instance_eval`**

-
  ```

  route_eval(&pass_block) if pass_block
  route_missing
  end

  # Run a route block and throw :halt with the result.
  def route_eval(&block)
  throw :halt, instance_eval(&block)
  end

  # If the current request matches pattern and conditions, fill params
  # with keys and call the given block.
  # Revert params afterwards.
  #
  # Returns pass block.
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`route_eval`**

-
  ```

  route_eval(&pass_block) if pass_block
  route_missing
  end

  # Run a route block and throw :halt with the result.
  def route_eval(&block)
  throw :halt, instance_eval(&block)
  end

  # If the current request matches pattern and conditions, fill params
  # with keys and call the given block.
  # Revert params afterwards.
  #
  # Returns pass block.
  ```

- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **route!**

-
  ```
  686. end
  687.
  688. # Run routes defined on the class and all superclasses.
  689. def route!(base=self.class, pass_block=nil)
  690. if routes = base.routes[@request.request_method]
  691. routes.each do |pattern, keys, conditions, block|
  692. pass_block = process_route(pattern, keys, conditions) do
  693. route_eval(&block)
  694. end
  695. end
  696. end
  697.
  698. # Run routes defined in superclass.
  699. if base.superclass.respond_to?(:routes)
  700. return route!(base.superclass, pass_block)
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **process_route**

-
  ```
  734. {}
  735. end
  736. @params = @original_params.merge(params)
  737. @block_params = values
  738. catch(:pass) do
  739. conditions.each { |cond|
  740. throw :pass if instance_eval(&cond) == false }
  741. yield
  742. end
  743. end
  744. ensure
  745. @params = @original_params
  746. end
  747.
  748. # No matching route was found or all routes passed. The default
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **catch**

-
  ```
  731. elsif values.any?
  732. {'captures' => values}
  733. else
  734. {}
  735. end
  736. @params = @original_params.merge(params)
  737. @block_params = values
  738. catch(:pass) do
  739. conditions.each { |cond|
  740. throw :pass if instance_eval(&cond) == false }
  741. yield
  742. end
  743. end
  744. ensure
  745. @params = @original_params
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **process_route**

-
  ```
  731. elsif values.any?
  732. {'captures' => values}
  733. else
  734. {}
  ```

```
end
@params = @original_params.merge(params)
@block_params = values
catch(:pass) do
conditions.each { |cond|
throw :pass if instance_eval(&cond) == false }
yield
end
end
ensure
@params = @original_params
```

- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **route!**
-
```
base.filters[type].each { |block| instance_eval(&block) }
end

# Run routes defined on the class and all superclasses.
def route!(base=self.class, pass_block=nil)
if routes = base.routes[@request.request_method]
routes.each do |pattern, keys, conditions, block|
pass_block = process_route(pattern, keys, conditions) do
route_eval(&block)
end
end
end

# Run routes defined in superclass.
if base.superclass.respond_to?(:routes)
```

- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **each**
-
```
filter! type, base.superclass if base.superclass.respond_to?(:filters)
base.filters[type].each { |block| instance_eval(&block) }
end

# Run routes defined on the class and all superclasses.
def route!(base=self.class, pass_block=nil)
if routes = base.routes[@request.request_method]
routes.each do |pattern, keys, conditions, block|
pass_block = process_route(pattern, keys, conditions) do
route_eval(&block)
end
end
end

# Run routes defined in superclass.
```

- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **route!**
-
```
filter! type, base.superclass if base.superclass.respond_to?(:filters)
base.filters[type].each { |block| instance_eval(&block) }
end

# Run routes defined on the class and all superclasses.
def route!(base=self.class, pass_block=nil)
if routes = base.routes[@request.request_method]
routes.each do |pattern, keys, conditions, block|
pass_block = process_route(pattern, keys, conditions) do
```

```
693. route_eval(&block)
694. end
695. end
696. end
697.
698. # Run routes defined in superclass.
```

- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`dispatch!`**

-
  ```
  819. res
  820. end
  821.
  822. # Dispatch a request with error handling.
  823. def dispatch!
  824. static! if settings.static? && (request.get? || request.head?)
  825. filter! :before
  826. route!
  827. rescue NotFound => boom
  828. handle_not_found!(boom)
  829. rescue ::Exception => boom
  830. handle_exception!(boom)
  831. ensure
  832. filter! :after unless env['sinatra.static_file']
  833. end
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`call!`**

-
  ```
  612. @response = Response.new
  613. @params = indifferent_params(@request.params)
  614. template_cache.clear if settings.reload_templates
  615. force_encoding(@request.route)
  616. force_encoding(@params)
  617.
  618. @response['Content-Type'] = nil
  619. invoke { dispatch! }
  620. invoke { error_block!(response.status) }
  621. unless @response['Content-Type']
  622. if body.respond_to?(:to_ary) and body.first.respond_to? :content_type
  623. content_type body.first.content_type
  624. else
  625. content_type :html
  626. end
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **`instance_eval`**

-
  ```
  784. # Creates a Hash with indifferent access.
  785. def indifferent_hash
  786. Hash.new {|hash,key| hash[key.to_s] if Symbol === key }
  787. end
  788.
  789. # Run the block with 'throw :halt' support and apply result to the response.
  790. def invoke(&block)
  791. res = catch(:halt) { instance_eval(&block) }
  792. return if res.nil?
  793.
  794. case
  795. when res.respond_to?(:to_str)
  796. @response.body = [res]
  797. when res.respond_to?(:to_ary)
  ```

```
res = res.to_ary
```

- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **invoke**

-
  ```
  # Creates a Hash with indifferent access.
  def indifferent_hash
  Hash.new {|hash,key| hash[key.to_s] if Symbol === key }
  end

  # Run the block with 'throw :halt' support and apply result to the response.
  def invoke(&block)
  res = catch(:halt) { instance_eval(&block) }
  return if res.nil?

  case
  when res.respond_to?(:to_str)
  @response.body = [res]
  when res.respond_to?(:to_ary)
  res = res.to_ary
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **catch**

-
  ```
  # Creates a Hash with indifferent access.
  def indifferent_hash
  Hash.new {|hash,key| hash[key.to_s] if Symbol === key }
  end

  # Run the block with 'throw :halt' support and apply result to the response.
  def invoke(&block)
  res = catch(:halt) { instance_eval(&block) }
  return if res.nil?

  case
  when res.respond_to?(:to_str)
  @response.body = [res]
  when res.respond_to?(:to_ary)
  res = res.to_ary
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **invoke**

-
  ```
  # Creates a Hash with indifferent access.
  def indifferent_hash
  Hash.new {|hash,key| hash[key.to_s] if Symbol === key }
  end

  # Run the block with 'throw :halt' support and apply result to the response.
  def invoke(&block)
  res = catch(:halt) { instance_eval(&block) }
  return if res.nil?

  case
  when res.respond_to?(:to_str)
  @response.body = [res]
  when res.respond_to?(:to_ary)
  res = res.to_ary
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **call!**

-
  ```
  @response = Response.new
  @params = indifferent_params(@request.params)
  template_cache.clear if settings.reload_templates
  ```

```
force_encoding(@request.route)
force_encoding(@params)

@response['Content-Type'] = nil
invoke { dispatch! }
invoke { error_block!(response.status) }
unless @response['Content-Type']
if body.respond_to?(:to_ary) and body.first.respond_to? :content_type
content_type body.first.content_type
else
content_type :html
end
```

- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb in **call**

-
  ```
  @app = app
  @template_cache = Tilt::Cache.new
  yield self if block_given?
  end

  # Rack call interface.
  def call(env)
  dup.call!(env)
  end

  attr_accessor :env, :request, :response, :params

  def call!(env) # :nodoc:
  @env = env
  @request = Request.new(env)
  ```
- /var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/showexceptions.rb in **call**

-
  ```

  def initialize(app)
  @app = app
  @template = ERB.new(TEMPLATE)
  end

  def call(env)
  @app.call(env)
  rescue Exception => e
  errors, env["rack.errors"] = env["rack.errors"], @@eats_errors

  if respond_to?(:prefers_plain_text?) and prefers_plain_text?(env)
  content_type = "text/plain"
  body = [dump_exception(e)]
  else
  ```
- /var/lib/gems/1.8/gems/rack-1.2.1/lib/rack/methodoverride.rb in **call**

-
  ```
  method = method.to_s.upcase
  if HTTP_METHODS.include?(method)
  env["rack.methodoverride.original_method"] = env["REQUEST_METHOD"]
  env["REQUEST_METHOD"] = method
  end
  end

  @app.call(env)
  ```

```
end
end
end
```

- `/var/lib/gems/1.8/gems/rack-1.2.1/lib/rack/commonlogger.rb` in **call**

-
  ```
  def initialize(app, logger=nil)
  @app = app
  @logger = logger
  end

  def call(env)
  began_at = Time.now
  status, header, body = @app.call(env)
  header = Utils::HeaderHash.new(header)
  log(env, status, header, began_at)
  [status, header, body]
  end

  private

  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **call**

-
  ```
  middleware.each { |c,a,b| builder.use(c, *a, &b) }

  builder.run super
  builder.to_app
  end

  def call(env)
  synchronize { prototype.call(env) }
  end

  private
  def detect_rack_handler
  servers = Array(server)
  servers.each do |server_name|
  begin
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **synchronize**

-
  ```
  end

  @@mutex = Mutex.new
  def synchronize(&block)
  if lock?
  @@mutex.synchronize(&block)
  else
  yield
  end
  end

  def metadef(message, &block)
  (class << self; self; end).
  send :define_method, message, &block
  end
  ```
- `/var/lib/gems/1.8/gems/sinatra-1.2.0/lib/sinatra/base.rb` in **call**

-
  ```
  middleware.each { |c,a,b| builder.use(c, *a, &b) }
  ```

```

builder.run super
builder.to_app
end

def call(env)
synchronize { prototype.call(env) }
end

private
def detect_rack_handler
servers = Array(server)
servers.each do |server_name|
begin
```

- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **process_client**

-
  ```
  nil
  end

  # once a client is accepted, it is processed in its entirety here
  # in 3 easy steps: read request, call app, write app response
  def process_client(client)
  client.fcntl(Fcntl::F_SETFD, Fcntl::FD_CLOEXEC)
  response = app.call(env = REQUEST.read(client))

  if 100 == response[0].to_i
  client.write(Const::EXPECT_100_RESPONSE)
  env.delete(Const::HTTP_EXPECT)
  response = app.call(env)
  end
  HttpResponse.write(client, response, HttpRequest::PARSER.headers?)
  ```
- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **worker_loop**

-
  ```
  # changes with chmod doesn't update ctime on all filesystems; so
  # we change our counter each and every time (after process_client
  # and before IO.select).
  alive.chmod(m = 0 == m ? 1 : 0)

  ready.each do |sock|
  begin
  process_client(sock.accept_nonblock)
  nr += 1
  alive.chmod(m = 0 == m ? 1 : 0)
  rescue Errno::EAGAIN, Errno::ECONNABORTED
  end
  break if nr < 0
  end

  ```
- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **each**

-
  ```
  # prefer temporary files to be unlinked for security,
  # performance and reliability reasons, so utime is out. No-op
  # changes with chmod doesn't update ctime on all filesystems; so
  # we change our counter each and every time (after process_client
  # and before IO.select).
  alive.chmod(m = 0 == m ? 1 : 0)
  ```

```

ready.each do |sock|
begin
process_client(sock.accept_nonblock)
nr += 1
alive.chmod(m = 0 == m ? 1 : 0)
rescue Errno::EAGAIN, Errno::ECONNABORTED
end
break if nr < 0
```

- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`worker_loop`**

-
  ```
  # prefer temporary files to be unlinked for security,
  # performance and reliability reasons, so utime is out. No-op
  # changes with chmod doesn't update ctime on all filesystems; so
  # we change our counter each and every time (after process_client
  # and before IO.select).
  alive.chmod(m = 0 == m ? 1 : 0)

  ready.each do |sock|
  begin
  process_client(sock.accept_nonblock)
  nr += 1
  alive.chmod(m = 0 == m ? 1 : 0)
  rescue Errno::EAGAIN, Errno::ECONNABORTED
  end
  break if nr < 0
  ```
- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`spawn_missing_workers`**

-
  ```
  (0...worker_processes).each do |worker_nr|
  WORKERS.values.include?(worker_nr) and next
  worker = Worker.new(worker_nr, Unicorn::Util.tmpio)
  before_fork.call(self, worker)
  WORKERS[fork {
  ready_pipe.close if ready_pipe
  self.ready_pipe = nil
  worker_loop(worker)
  }] = worker
  end
  end

  def maintain_worker_count
  (off = WORKERS.size - worker_processes) == 0 and return
  off < 0 and return spawn_missing_workers
  ```
- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`fork`**

-
  ```
  end

  def spawn_missing_workers
  (0...worker_processes).each do |worker_nr|
  WORKERS.values.include?(worker_nr) and next
  worker = Worker.new(worker_nr, Unicorn::Util.tmpio)
  before_fork.call(self, worker)
  WORKERS[fork {
  ready_pipe.close if ready_pipe
  self.ready_pipe = nil
  worker_loop(worker)
  ```

```
}] = worker
end
end

```

- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **spawn_missing_workers**

-
  ```
  end
  
  def spawn_missing_workers
  (0...worker_processes).each do |worker_nr|
  WORKERS.values.include?(worker_nr) and next
  worker = Worker.new(worker_nr, Unicorn::Util.tmpio)
  before_fork.call(self, worker)
  WORKERS[fork {
  ready_pipe.close if ready_pipe
  self.ready_pipe = nil
  worker_loop(worker)
  }] = worker
  end
  end
  
  ```
- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **each**

-
  ```
  logger.error "worker=#{worker.nr} PID:#{wpid} timeout " \
  "(#{diff}s > #{timeout}s), killing"
  kill_worker(:KILL, wpid) # take no prisoners for timeout violations
  end
  end
  
  def spawn_missing_workers
  (0...worker_processes).each do |worker_nr|
  WORKERS.values.include?(worker_nr) and next
  worker = Worker.new(worker_nr, Unicorn::Util.tmpio)
  before_fork.call(self, worker)
  WORKERS[fork {
  ready_pipe.close if ready_pipe
  self.ready_pipe = nil
  worker_loop(worker)
  ```
- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **spawn_missing_workers**

-
  ```
  logger.error "worker=#{worker.nr} PID:#{wpid} timeout " \
  "(#{diff}s > #{timeout}s), killing"
  kill_worker(:KILL, wpid) # take no prisoners for timeout violations
  end
  end
  
  def spawn_missing_workers
  (0...worker_processes).each do |worker_nr|
  WORKERS.values.include?(worker_nr) and next
  worker = Worker.new(worker_nr, Unicorn::Util.tmpio)
  before_fork.call(self, worker)
  WORKERS[fork {
  ready_pipe.close if ready_pipe
  self.ready_pipe = nil
  worker_loop(worker)
  ```
- /var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb in **maintain_worker_count**

-
  ```
  worker_loop(worker)
  }] = worker
  end
  end

  def maintain_worker_count
  (off = WORKERS.size - worker_processes) == 0 and return
  off < 0 and return spawn_missing_workers
  WORKERS.dup.each_pair { |wpid,w|
  w.nr >= worker_processes and kill_worker(:QUIT, wpid) rescue nil
  }
  end

  # if we get any error, try to write something back to the client
  # assuming we haven't closed the socket, but don't get hung up
  ```
- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`join`**

-
  ```
  # machine) comes out of suspend/hibernation
  if (last_check + timeout) >= (last_check = Time.now)
  murder_lazy_workers
  else
  # wait for workers to wakeup on suspend
  master_sleep(timeout/2.0 + 1)
  end
  maintain_worker_count if respawn
  master_sleep(1)
  when :QUIT # graceful shutdown
  break
  when :TERM, :INT # immediate shutdown
  stop(false)
  break
  when :USR1 # rotate logs
  ```
- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`loop`**

-
  ```
  logger.info "master process ready" # test_exec.rb relies on this message
  if ready_pipe
  ready_pipe.syswrite($$.to_s)
  ready_pipe.close rescue nil
  self.ready_pipe = nil
  end
  begin
  loop do
  reap_all_workers
  case SIG_QUEUE.shift
  when nil
  # avoid murdering workers after our master process (or the
  # machine) comes out of suspend/hibernation
  if (last_check + timeout) >= (last_check = Time.now)
  murder_lazy_workers
  ```
- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`join`**

-
  ```
  logger.info "master process ready" # test_exec.rb relies on this message
  if ready_pipe
  ready_pipe.syswrite($$.to_s)
  ready_pipe.close rescue nil
  self.ready_pipe = nil
  ```

```
end
begin
loop do
reap_all_workers
case SIG_QUEUE.shift
when nil
# avoid murdering workers after our master process (or the
# machine) comes out of suspend/hibernation
if (last_check + timeout) >= (last_check = Time.now)
murder_lazy_workers
```

- `/var/lib/gems/1.8/gems/unicorn-1.1.6/lib/unicorn.rb` in **`run`**

-
  ```
  # since there is nothing in the application stack that is responsible
  # for client shutdowns/disconnects.
  class ClientShutdown < EOFError
  end

  class << self
  def run(app, options = {})
  HttpServer.new(app, options).start.join
  end

  # This returns a lambda to pass in as the app, this does not "build" the
  # app (which we defer based on the outcome of "preload_app" in the
  # Unicorn config). The returned lambda will be called when it is
  # time to build the app.
  def builder(ru, opts)
  ```
- `/var/lib/gems/1.8/gems/unicorn-1.1.6/bin/unicorn_rails` in **`nil`**

-
  ```
  FileUtils.mkdir_p(%w(cache pids sessions sockets).map! { |d| "tmp/#{d}" })
  end

  if daemonize
  options[:pid] = "tmp/pids/unicorn.pid"
  Unicorn::Launcher.daemonize!(options)
  end
  Unicorn.run(app, options)
  ```
- `/usr/local/bin/unicorn_rails` in **`load`**

-
  ```

  if ARGV.first =~ /^_(.*)_$/ and Gem::Version.correct? $1 then
  version = $1
  ARGV.shift
  end

  gem 'unicorn', version
  load Gem.bin_path('unicorn', 'unicorn_rails', version)
  ```
- `/usr/local/bin/unicorn_rails` in **`nil`**

-
  ```

  if ARGV.first =~ /^_(.*)_$/ and Gem::Version.correct? $1 then
  version = $1
  ARGV.shift
  end

  gem 'unicorn', version
  load Gem.bin_path('unicorn', 'unicorn_rails', version)
  ```