Ruby on Rails 2.2

What's new?

by **Carlos Brando** and **Carl Youngblood**

published on envycasts

Ruby on Rails 2.2

WHAT'S NEW

First Edition

Ruby on Rails 2.2

WHAT'S NEW

First Edition

by Carlos Brando
translated by Carl Youngblood

First edition: October 2008

Carlos Brando
http://www.nomedojogo.com

Carl Youngblood
http://youngbloods.org

INTRODUCTION

Ruby on Rails 2.2 is chock full of new features, improvements and bug fixes. In this book, you will find a brief description, along with an example (in most cases) of each significant change in this new version of Rails.

This book was a significant undertaking, and we hope that it helps you to make the most of each new feature in Rails.

With this new version, Rails has turned polyglot. With the new internationalization (i18n) system, you can make applications for users throughout the world with ease.

A lot of work went into making Rails thread-safe and as ready as possible for the upcoming Ruby 1.9. There was also a great deal of effort spent on making it more compatible with JRuby. Although the thread-safe mode is not yet available to everyone, since it only works on virtual machines with support for native threads (such as JRuby), it is a great addition to the framework.

If there were complaints in the past about lack of documentation in Rails, they should be silenced with this new version. A lot of hard work has been done documenting the code and explaining Rails functionality. For an example, run this in your terminal:

rake doc:guides

This rake task will create a doc/guides folder in the root of your project with various guides to help you learn Rails.

Chapter 1

ActiveRecord

NEW OPTIONS FOR ASSOCATIONS: :VALIDATE

A new option for associations has been added to Rails. The `:validate` option can now be used to disable or enable validation of a model's associated objects. For example:

```
class AuditLog < ActiveRecord::Base
  belongs_to :developer, :validate => false
end

log = AuditLog.create(:developer_id => 0 , :message => "")
log.developer = Developer.new

puts log.developer.valid?
# => false
```

```
puts log.valid?
# => true

puts log.save
# => true
```

As you can see in this example, even though the associated object (`Developer`) is not valid, the main object (`AuditLog`) gets saved in the database. This wasn't the normal behavior in earlier versions of Rails, where a parent object could only be saved if all its children were valid.

Although, in this prior example, I disabled validation to demonstrate this new feature, in fact, the default setting for `:validate` has been set to `false` in Rails 2.2, so `:validate => false` would not actually be necessary in the code above. On the other hand, if you wished to keep the old behavior, it would be necessary to use `:validate => true`.

NEW WAY OF SPECIFYING CONDITIONS WITH A HASH

When performing database queries, sometimes you need to use the `:joins` option, either to improve application performance or when you need to retrieve information that depends on results from more than one table.

For example, if you wanted to retrieve all users who bought red items, you could do something like this:

```
User.all :joins => :items, :conditions => ["items.color = ?", 'red']
```

This syntax is a little odd, since you need to include the name of the table (**items** in this case) inside a string.

In Rails 2.2 there is a new way of doing this, using a hash key to identify the table:

```
User.all :joins => :items, :conditions => {
  :age => 10,
```

```
  :items => { :color => 'red' }
}

# another example that perhaps makes the code a little clearer
User.all :joins => :items, :conditions => {
  :users => { :age => 10 },
  :items => { :color => 'red' }
}
```

In my opinion, the code is a lot clearer this way, especially when you need to provide conditions for many fields from various tables.

Just keep in mind that the key you use is the (plural) name of the table or an alias, if you have specified one in the query.

NEW :FROM OPTION FOR CALCULATION METHODS IN ACTIVERECORD

A new option has been included in the calculation methods of `ActiveRecord` (`count`, `sum`, `average`, `minimum` and `maximum`).

When you use the `:from` option, you can override the name of the table in the query that `ActiveRecord` generates, which doesn't seem very useful at first glance, but one interesting thing that this allows you to do is force MySQL to use a specific index for the calculation.

Check out some examples:

```
# Forcing MySQL to use an index for a calculation
Edge.count :all, :from => 'edges USE INDEX(unique_edge_index)',
           :conditions => 'sink_id < 5')
```

```
# Doing the calculation in a different table from the associated class
Company.count :all, :from => 'accounts'
```

THE MERGE_CONDITIONS METHOD IN ACTIVERECORD IS NOW PUBLIC

The `merge_conditions` method in `ActiveRecord` is now a public method, which means that it will be available in all your models.

This method does exactly what its name says—it allows you provide many different `conditions` in your parameters that get combined into a single condition.

```
class Post < ActiveRecord::Base
end

a = { :author => 'Carlos Brando' }
b = [ 'title = ?', 'Edge Rails' ]

Post.merge_conditions(a, b)
# => "(\"posts\".\"author\" = 'Carlos Brando') AND (title = 'Edge Rails')"
```

Note that the `conditions` always get combined with **AND**.

DEFINING HOW VALIDATES_LENGTH_OF SHOULD FUNCTION

The `validates_length_of` is one of many validation methods in `ActiveRecord`. This particular method is for making sure that the value stored in a given database column has a certain length. It lets you specify a maximum length, a minimum length, an exact length, or even a range of lengths.

"Length," however, is relative. When we say "length" these days we are talking about the number of characters in the text. But imagine a case where you have a form field in which the limit is not defined by the number of characters, but by the number of words, such as, "Write a paragraph with at least 100 words." (I think a better example would be "Submit a comment with no more than 100 words.") Imagine a page where the user must submit an essay, for example.

Before Rails 2.2, our only option would be to create a custom validation method, but now you can personalize `validates_length_of` using the `:tokenizer` option. The following example resolves the aforementioned problem:

```
validates_length_of :essay,
                    :minimum => 100,
                    :too_short => "Sua redação deve ter no mínimo %d palavras."),
                    :tokenizer => lambda {|str| str.scan(/\w+/) }
```

This is just one example of what you can do with this new option. In addition to this, you can use it to count only the number of digits, the times a certain word was used, etc.

TREATING THE :LIMIT OPTION AS COLUMN BYTE SIZE

Starting with this version of Rails, the :limit option specifies byte size on integer columns for MySQL and PostgreSQL (on Sqlite it works this way already).

The type of column used in the database depends on the number of bytes specified. Take a look at the code snippet that determines the column type in MySQL:

```
when 1; 'tinyint'
when 2; 'smallint'
when 3; 'mediumint'
when nil, 4, 11; 'int(11)' # compatibility with MySQL default
```

```
  when 5..8; 'bigint'
  else raise(ActiveRecordError, "No integer type has byte size #{limit}")
```

And for PostgreSQL:

```
  when 1..2; 'smallint'
  when 3..4, nil; 'integer'
  when 5..8; 'bigint'
```

INDICATING ANOTHER PRIMARY_KEY IN ASSOCIATIONS

A new option called `:primary_key` has been added to the `has_many` and `has_one` methods.

Using this option, you can define which method and associated model will return the primary key that will be used in the association. Obviously the standard primary key is `id`.

Look at this usage example:

```
has_many :clients_using_primary_key, :class_name => 'Client',
    :primary_key => 'name', :foreign_key => 'firm_name'
```

The `has_one` method functions exactly as explained above.

NEW TEST HELPER (ASSERT_SQL)

Maybe you have already heard of `assert_queries`, which helps to validate tests and the number of queries executed on a database. For example:

In the test below I am specifying that if there are `partial_updates`, a query should be executed on the database; otherwise, no query should be executed.

We have now got another helper to make it easier to test the SQL query that will be run from `ActiveRecord` abstractions. For example:

```
def test_empty_with_counter_sql
  company = Firm.find(:first)
  assert_sql /COUNT/i do
    assert_equal false, company.clients.empty?
  end
end
```

In the example above I am asserting that in the specified block, at least one query should contain the word **COUNT**. Obviously you can be more specific in the regular expression that you are using. Let's looks at another example:

```
assert_sql(/\(\"companies\".\"id\" IN \(1\)\)/) do
  Account.find(1, :include => :firm)
end
```

MIGRATIONPROXY

Imagine that, after running a number of migrations, a certain model gets renamed. Now imagine that, before this takes place, another migration tries to refer to this model. Because all migrations used to get loaded regardless of whether they would be executed or not, this situation would cause an ugly error and halt execution of migrations.

To avoid this problem, a new class called `MigrationProxy` has been created, which stores the name, version and filename of each migration, and uses this information to load migrations only when necessary, so they don't all get loaded at the same time.

CONNECTION POOLS IN RAILS

One of the complaints that a lot of folks make about Rails is that it's slow. We know this isn't exactly true, but we also know that a lot could be done to improve Rails' performance.

And something has been done. A database **connection pool** has been added to Rails.

Every time a database query is made, some time is lost opening a new connection before data can be retrieved or saved. At first glance it may seem trivial, but opening a database connection is not so simple. It is necessary to open a network connection with the database server, perform authentication, and verify a number of things. All of this takes resources and time. After creating this connection, Rails uses it for all applicable database requests, and heavier queries can delay the execution of other requests. This goes to show how the database can become the achilles heel of some large Rails projects.

The solution for this problem is to reuse previously-created database connections, having them perform more tasks before closing. Or in more technical terms, we need a connection pool.

Here's how it works: A database connection is opened so a query can be performed. After this, instead of closing it, it gets saves in a connection pool. When another query is executed, instead of opening a new conneciton, the system takes advantage of the one that has already been created, reducing the time and resources required to get the task done. Many connections can be stored in the pool at the same time, and requests get distributed between them. This means that a slow query can be running on the database while the application still receives requests and executes queries using the other connections in the pool.

In Rails, a new pool gets created every time the `establish_connection` method is called. In other words, every database listed in **database.yml** will have its own connection pool.

The pool starts out empty and grows until it reaches the default limit of five connections, but you can increase this default by adding the `pool` option in the database configuration:

```
development:
  adapter: mysql
  host: localhost
  username: myuser
  password: mypass
  database: somedatabase
  pool: 10
  wait_timeout: 15
```

If no connection is available, a thread will wait five seconds (default) for another connection before it times out. This waiting period can also be configured by adding the `wait_timeout` option to the database configuration.

If you want to use connection pools outside of `ActionPack`, there is a method called `ActiveRecord::Base.connection_pool`, which allows you to manually handle the reserving (checking out) and releasing (checking in) of the connections in the pool. Don't forget to check in a connection after you're done using it so that it can be reused by other requests.

```
connection = ActiveRecord::Base.connection_pool.checkout
# do something in the database
ActiveRecord::Base.connection_pool.checkin(connection)
```

You can also use the `ActiveRecord::Base.connection_pool.with_connection` method, which does the checkout/checkin for you, making your code a little more secure.

```
ActiveRecord::Base.connection_pool.with_connection do |connection|
  # do something in the database
end
```

TRANSACTIONAL MIGRATIONS IN POSTGRESQL

When a migration is executed and an error occurs, only the code before the error actually gets applied to the database. Besides this, the migration will be marked as finished. Fixing this can be a headache.

But if the database you're using supports **DDL rollbacks** in transactions, it becomes possible to use this feature to undo all the changes that were made before the error. The problem is that not all database engines support this feature. MySQL, for example, doesn't. But PostgreSQL, SQL Server and other database engines do.

Rails has been updated to allow the use of transationcs in migrations for databases that support it. Although Rails supports this feature, the database adapter must be updated (by simply causing the `supports_ddl_transactions?` method to return `true`) to be able to use it. As of this book's release, only the PostgreSQL adapter appears to have been updated.

NEW DESTRUCTIVE VERSIONS OF ACTIVERECORD FIND METHODS

The `ActiveRecord` dynamic finders now have destructive counterparts that raise a `RecordNotFound` exception when the result set is empty, instead of returning `nil` like the original behavior.

To use these destructive versions, just add an exclamation point to the end of the method. Here is an example:

```
Topic.find_by_title!("The First Topic!")
# => ActiveRecord::RecordNotFound
```

IMPROVING PERFORMANCE IN ASSOCIATION_ID METHODS.

If you have two models, `Post` and `Comment`, where a post has many (`has_many`) comments, if you execute the following:

```
Post.first.comment_ids
```

Rails will use the following query to retrieve the ids:

```
SELECT * FROM `comments` WHERE (`comments`.post_id = 1)
```

But in this case, we don't need the entire objects. The following query would be more than sufficient for this method to function, and it would also perform better:

```
SELECT `comments`.id FROM `comments` WHERE (`comments`.post_id = 1)
```

For both `has_many` and `has_many :through` associations, the Rails code base has been updated to include this performance enhancement from this version on.

ANOTHER DYNAMIC FINDER

In addition to the existing `ActiveRecord` finders, we now have `find_last_by`. The others are `find_by` and `find_all_by`.

Besides being simpler, this also offers a much more elegant way of retrieving the last comment made by a certain user, for example. Check it out:

```
Comment.find_last_by_author("Carlos Brando")
```

```
    # is the same as

Comment.last(:conditions => { :author => "Carlos Brando" })
```

SUPPORT FOR THE :LIMIT OPTION IN UPDATE_ALL

The `update_all` method now has a new `:limit` option. This is very helpful, because it makes sure that things will still work when you use `update_all` in `has_many` associations that use the `:limit` option.

NEW OPTIONS FOR THE COMPOSED_OF METHOD

The `composed_of` method has two new options: `:constructor` and `:converter`.

The `constructor` option can receive a symbol with the name of a method or a `Proc`. By default, the composite class is created by the `new` method, receiving all the mapped attributes as parameters, exactly in the order that they were mapped. If, for some reason, your class does not follow this convention, you should use the `:constructor` option. It allows you to change the way your class gets created. Take a look at this example, taken from Rails' own documentation:

```
composed_of :ip_address,
  :class_name => 'IPAddr',
  :mapping => %w(ip to_i),
  :constructor => Proc.new { |ip| IPAddr.new(ip, Socket::AF_INET) }
```

As you can see from the example, upon creating a new instance of the `IPAddr` class, it is necessary to supply an additional parameter to the constructor. Using the `:constructor` option makes this easy.

As for the `:converter` option, it also accepts a symbol representing a method from the class in `:class_name`, or a `Proc`, and it is called when the value of the composed field gets assigned from another instance and requires conversion. Here is an example:

```
composed_of :balance,
  :class_name => "Money",
  :mapping => %w(balance amount),
  :converter => Proc.new { |balance| Money.parse(balance) }
```

In this example, the `balance=` method will always be expecting an instance of the `Money` class, but, if another type of object is passed, it should be converted using the `parse` method of the `Money` object.

You should use this new `:converter` option instead of the conversion block that the method used to allow.

UPDATING AN ASSOCIATION USING ITS FOREIGN KEY

I'm not sure if this is a bug or not, but in my opinion, this has been a problem. Check out the following code, in which I try to modify a user's account using its foreign key in a project that uses Rails 2.1 or earlier:

```
class User < ActiveRecord::Base
  belongs_to :account
end

user = User.first
# => #<User id: 1, login: "admin", account_id: 1>

user.account
# => #<Account id: 1, name: "My Account">

user.account_id = 2
```

```
# => 2

user.account
# => #<Account id: 1, name: "My Account">
```

Note that I am modifying the user's account, but the association wasn't updated. Even after saving the `user` object, if it is not reloaded, the association will continue to show the wrong account.

This has been fixed in Rails 2.2. Take a look:

```
class Comment < ActiveRecord::Base
  belongs_to :post
end

comment = Comment.first
# => #<Comment id: 1>

>> comment.post
# => #<Post id: 1>

>> comment.post_id = 2
# => 2

>> comment.post
# => #<Post id: 2>
```

Notice that, upon modifying the `post` using its foreign key, the association gets updated automatically.

ALIAS_ATTRIBUTE WORKS WITH DIRTY OBJECTS

To understand this modification, we will need to analyze a code snippet running in both previous and 2.2 versions of Rails. Let's look at an example model:

```
class Comment < ActiveRecord::Base
  alias_attribute :text, :body
end
```

Note that I am using the `alias_attribute` method to create an alias for the `body` attribute called `text`. In theory, this method should replicate all the various dynamic methods that get created by `ActiveRecord` that involve the `body` attribute. But look at this example code running in Rails 2.1 or earlier:

```
c = Comment.first
# => #<Comment id: 1, body: "my comment">

c.body
# => "my comment"

c.text
# => "my comment"

c.body = "a new message"
# => "a new message"

c.body_changed?
# => true

c.text_changed?
# => NoMethodError: undefined method `text_changed?' ...
```

Upon running the `text_changed?` method, we have an error, because the `alias_attribute` call was replicating the tracking methods, but this has now been corrected. Check out the same code running in a Rails 2.2 project:

```
c = Comment.first
# => #<Comment id: 1, body: "my comment">

c.body
# => "my comment"

c.text
# => "my comment"

c.body = "a new message"
# => "a new message"

c.body_changed?
# => true

c.text_changed?
# => true

c.text_change
# => ["my comment", "a new message"]
```

NEW MODEL#DELETE INSTANCE METHOD

To make `ActiveRecord` more consistent, an additional instance method has been created: `Model#delete`. It is similar to the class method with the same name. The `delete` method, contrary to the `destroy` method, erases the database record without calling any callback methods, such as `before_destroy` and `after_destroy`.

This method also disregards constraint options, such as the `:dependent` option, which specifies what should be done with associated objects when a record is destroyed.

```
client = Client.find(1)
client.delete
```

MAKING ACTIVERECORD ATTRIBUTES PRIVATE

Rails 2.2 allows you to mark `ActiveRecord` objects as `private`. Until now, this was difficult, because these records are created through metaprogramming.

To see how this works, let's make the `name` attribute of the `User` model private:

```
class User < ActiveRecord::Base

  private
  def name
    "I'm private"
  end

end
```

Then let's try to retrieve it:

```
user = User.first
# => #<User id: 1, name: "teste">

user.name
# => NoMethodError: undefined method `NoMethodError' for #<User:0x234df08>
```

You can see that the `NoMethodError` exception was raised when we tried to call a private method. On the other hand, I can modify the user's name, since the `name=` method is still public:

```
user.name = "Carlos"
# => "Carlos"
```

Chapter 2

ActiveSupport

ARRAY#SECOND THROUGH ARRAY#TENTH

In the `Array` class we already had the `first` and `last` methods, so why not add the `second`, `third`, `fourth`, and so on? That's just what they did. These methods return the element at the specified offset of the array.

Here is an example:

```
array = (1..10).to_a

array.second  # => array[1]
array.third   # => array[2]
array.fourth  # => array[3]
array.fifth   # => array[4]
array.sixth   # => array[5]
array.seventh # => array[6]
```

```
array.eighth  # => array[7]
array.ninth   # => array[8]
array.tenth   # => array[9]
```

NEW METHOD: ENUMERABLE#MANY?

A new method called `many?` was added to the `Enumerable` module. And just as its name seems to say, it tells you if the collection contains more than one object; that is to say, if it contains many objects.

This method is an alias to `collection.size > 1`. Let's look at a few examples:

```
>> [].many?
# => false

>> [ 1 ].many?
# => false

>> [ 1, 2 ].many?
# => true
```

Besides the format shown in these examples, this method also got a new implementation that allows it to receive blocks, which work the same way as the `any?` method.

Check out these examples:

```
>> x = %w{ a b c b c }
# => ["a", "b", "c", "b", "c"]

>> x.many?
# => true
```

```
>> x.many? { |y| y == 'a' }
# => false

>> x.many? { |y| y == 'b' }
# => true


# another example...
people.many? { |p| p.age > 26 }
```

Just to remind and reiterate, this method only returns `true` if more than one iteration of the block returns true, and when the collection has more than one object if called without a block.

As an interesting side note, the method was initially going to be called `several?` but it ended up as `many?`.

MAKE RULES FOR THE STRING#HUMANIZE

For some time now Pratik Naik has been trying to get this patch accepted in Rails and it looks like he finally succeeded.

In **config/initializers/inflections.rb** you have the option of adding new inflections to pluralization, singularization and others:

```
Inflector.inflections do |inflect|
  inflect.plural /^(ox)$/i, '\1en'
  inflect.singular /^(ox)en/i, '\1'
  inflect.irregular 'person', 'people'
  inflect.uncountable %w( fish sheep )
end
```

In Rails 2.2 you can also include inflections for the `String` class's `humanize` method. Let's look at some awesome examples:

```
'jargon_cnt'.humanize # => "Jargon cnt"
'nomedojogo'.humanize # => "Nomedojogo"

ActiveSupport::Inflector.inflections do |inflect|
  inflect.human(/_cnt$/i, '\1_count')
  inflect.human('nomedojogo', 'Nome do Jogo')
end

'jargon_cnt'.humanize # => "Jargon count"
'nomedojogo'.humanize # => "Nome do jogo"
```

INTRODUCING MEMOIZABLE FOR ATTRIBUTE CACHING

Performance is important, and one of the most commonly used methods for improving code execution speed is caching. You've probably written code like this before:

```
class Person < ActiveRecord::Base
  def age
    @age ||= a_very_complex_calculation
  end
end
```

In this version of Rails we have a more elegant way of doing this, using the `memoize` method (it is really `memoize` and not **memorize**). Let's change the example above to use this new functionality:

```
class Person < ActiveRecord::Base
  def age
```

```
    a_very_complex_calculation
  end
  memoize :age
end
```

The `age` method will be executed just once and its return value will be stored in memory and returned from memory for future calls to the method.

There is just one difference between the two examples above. In the first, as the method is always executed, if the value stored in `@age` is `nil` or `false`, the complex calculation will be executed again until you get the person's age. In the second example, the `age` method will only be executed once and the return value will always be returned for future calls, even if it's `nil` or `false`.

If you ever need to disable or re-enable caching of memoized properties, you can use the `unmemoize_all` and `memoize_all` methods:

```
@person = Person.first

# To stop caching the age method
@person.unmemoize_all

# To re-enable caching of just the age method
@person.memoize_all
```

NEW METHOD: OBJECT#PRESENT?

A new method was added to the Object base class. The `present?` method is equivalent to `!Object#blank?`.

In other words, an object is present if its not empty. But what exactly is an empty object?

```
class EmptyTrue
  def empty?() true; end
end

a = EmptyTrue.new
b = nil
c = false
d = ''
e = '   '
g = "  \n\t  \r "
g = []
h = {}

a.present? # => false
b.present? # => false
c.present? # => false
d.present? # => false
e.present? # => false
f.present? # => false
g.present? # => false
h.present? # => false
```

All of these objects are empty, or in other words, are not present.

But be very careful; some folks have gotten confused with this. The following are examples of some objects that AREN'T empty, or rather, are present:

```
class EmptyFalse
  def empty?() false; end
end

a = EmptyFalse.new
b = Object.new
```

```
c = true
d = 0
e = 1
f = 'a'
g = [nil]
h = { nil => 0 }

a.present? # => true
b.present? # => true
c.present? # => true
d.present? # => true
e.present? # => true
f.present? # => true
g.present? # => true
h.present? # => true
```

Any object that contains a value is present, and this is true even of an array containing a `nil`, because the array is not empty.

STRINGINQUIRER

A new class called `StringInquirer` has been added to Rails.

To understand how it works, I will need to explain with some terms. Let's create a class called `Client`, which contains a method that returns the `status` of the client:

```
class Client
  def status
    "active"
  end
end
```

```
c = Client.new
c.status
# => "active"

c.status == "active"
# => true

c.status == "inactive"
# => false
```

Ok, to this point everything is as expected. Now I will modify the implementation of the status method using the `StringInquirer` class, keeping in mind that the return value of `status` can come from a database column (of course)—this is only an example:

```
class Client
  def status
    ActiveSupport::StringInquirer.new("active")
  end
end

c = Client.new
c.status
# => "active"

# Here is the big difference:
c.status.active?
# => true

c.status.inactive?
# => false
```

To verify that the `status` of the client is what you expected, instead of comparing `Strings`, I use a method with the status value and an exclamation point.

Of course this usage has already begun in Rails itself. For example, if you need to verify that Rails was loaded in a production environment, you can substitute the old `Rails.env == "production"` for:

```
Rails.env.production?
```

NEW TEST SYNTAX

A new way of declaring tests has been added to Rails, using `test/do`. Take a look:

```
test "verify something" do
  # ...
end
```

This is the new standard for Rails tests. See how a unit test file was generated in this version:

```
require 'test_helper'

class PostTest < ActiveSupport::TestCase
  # Replace this with your real tests.
  test "the truth" do
    assert true
  end
end
```

The old way of doing it using methods will still work, so your old tests will not break.

ENABLING AND DISABLING DEPENDENCY LOADING

A new initialization parameter as been added to Rails that allows you to enable or disable the loading of new classes during a request.

```
config.dependency_loading = true
# or
config.dependency_loading = false
```

If `dependency_loading` is `true`, during a request, Rails will attempt to load into memory any class that wasn't initially loaded during initialization. If it is `false`, these classes are ignored.

If you want to run your project in a threaded environment, you should disable this option and load all classes using eager loading, or by using the `require` method during system initialization.

FILE.ATOMIC_WRITE COPIES THE ORIGINAL FILE'S PERMISSIONS

Some of you may be familiar with the `File.atomic_write` method. Unsurprisingly, it is used to ensure that file writes are atomic. It can be very useful in situations where you don't want other processes or threads to see a half-written file.

```
File.atomic_write("important.file") do |file|
  file.write("hello")
end
```

This method creates a temporary file while you are writing to it, and when you are finished, it replaces the old file with the new one.

A new feature of Rails 2.2 is that this method now ensures that the new file has the same permissions as the old one.

CAMELIZE(:LOWER)

By default, the `camelize` method in Rails is used to convert a string to the **UpperCamelCase** format, meaning that the first word of the string will be capitalized along with every other word. It is now possible to convert a string to **lowerCamelCase** (first word lowercase and other words capitalized) using the `:lower` argument. If, however, you try to execute the following code:

```
'Capital'.camelize(:lower)
# => "Capital"
```

The first letter is not rendered in lowercase. This has been corrected. Here is the same code being executed under Rails 2.2:

```
'Capital'.camelize(:lower)
# => "capital"
```

REPLACEMENT LIBRARY FOR GENERATING SECRET KEYS

The `Rails::SecretKeyGenerator` class, used to generate random secret keys for encrypting user session cookies, has been deprecated.

In its place, Rails now uses the new `ActiveSupport::SecureRandom` class, which has been added in Ruby 1.9. The **SecureRandom** library does the same thing as before but a little better. It supports the following random number generators:

- openssl
- /dev/urandom
- Win32

Let's examine some of the keys generated by this new library:

```
# random hexadecimal string.
ActiveSupport::SecureRandom.hex(10) #=> "52750b30ffbc7de3b362"
ActiveSupport::SecureRandom.hex(10) #=> "92b15d6c8dc4beb5f559"
ActiveSupport::SecureRandom.hex(11) #=> "6aca1b5c58e4863e6b81b8"
ActiveSupport::SecureRandom.hex(12) #=> "94b2fff3e7fd9b9c391a2306"
ActiveSupport::SecureRandom.hex(13) #=> "39b290146bea6ce975c37cfc23"

# random base64 string.
ActiveSupport::SecureRandom.base64(10) #=> "EcmTPZwWRAozdA=="
ActiveSupport::SecureRandom.base64(10) #=> "9b0nsevdwNuM/w=="
ActiveSupport::SecureRandom.base64(10) #=> "KO1nIU+p9DKxGg=="
ActiveSupport::SecureRandom.base64(11) #=> "l7XEiFja+8EKEtY="
ActiveSupport::SecureRandom.base64(12) #=> "7kJSM/MzBJI+75j8"
ActiveSupport::SecureRandom.base64(13) #=> "vKLJ0tXBHqQOuIcSIg=="

# random binary string.
ActiveSupport::SecureRandom.random_bytes(10) #=> "\016\t{\370g\310pbr\301"
ActiveSupport::SecureRandom.random_bytes(10) #=> "\323U\030TO\234\357\020\a\337"
```

NEW METHOD: EACH_WITH_OBJECT

The `each_with_object` method from Ruby 1.9 has been added to Rails, in case you don't happen to be running Ruby 1.9. This method is quite interesting. It functions like the well-known `inject` method, with a slight difference. Each iteration, besides receiving an element from the collection, also receives an object called **memo**.

For example, let's say that I intend to fill a hash with the values from a collection:

```
%w(first second third).each_with_object({}) do |str, hash|
  hash[str.to_sym] = str
end
# => {:second=>"second", :first=>"first", :third=>"third"}
```

Note that in the example above, the memo is an empty hash (`{}`). Inside the block, I fill this hash with the items from my collection.

Just be aware that you can't use unchangeable objects as your memo variable, such as numbers, `true`, and `false`. The example below always returns 1, since the number one can't be modified.

```
(1..5).each_with_object(1) { |value, memo| memo *= value } # => 1
```

INFLECTOR#PARAMETERIZE MAKES IT EASIER TO CREATE PRETTY URLS

A new inflector has been added to Rails that I find particularly useful. The `parameterize` method makes any text URL-friendly. Take a look at this example.

Model:

```
class User
  def to_param
    "#{id}-#{name.parameterize}"
  end
end
```

Controller:

```
@user = User.find(1)
# => #<User id: 1, name: "Carlos E. Brando">
```

View:

```
link_to @user.name, user_path
# => <a href="/users/1-carlos-e-brando">Carlos E. Brando</a>
```

One interesting thing to note about this feature is that it initially didn't work on accented characters, which was a significant problem for many people around the world. One day after the initial implementation, Michael Koziarski came to the rescue by adding this support. In spite of this, the code still wasn't perfect, so the code of the excellent slugalizer plugin, created by Henrik Nyh, was adapted to this purpose. Now it really is perfect!

For those who are not using Rails 2.2 yet, the slugalizer plugin can be used.

THREE NEW METHODS FOR CLASSES THAT WORK WITH DATES AND TIMES

The `Time`, `Date`, `DateTime` and `TimeWithZone` classes have been given three new methods that are quite convenient. They now have `today?`, `past?` and `future?`, to make our lives a little easier in some situations.

It will help to show how each one works. Here are the methods in action:

```
date = Date.current
# => Sat, 04 Oct 2008

date.today?
# => true
```

```
date.past?
# => false

date.future?
# => false

time = Time.now
# => Sat Oct 04 18:36:43 -0300 2008

time.today?
# => true
```

MODIFICATION IN THE ASSERT_DIFFERENCE METHOD

When you used the `assert_difference` method with more than one expression, and an error occurred, it was difficult to know which expression was the one that failed the assertion, since the error message didn't include this information.

In Rails 2.2, the error message explains exactly which method failed the assertion. For example:

```
assert_difference ['Post.count', 'current_user.posts.count'] do
  Post.create(params)
end
```

The code above displays the following message, in the event that the second expression doesn't pass:

<current_user.posts.count> was expression that failed. expected but was .

ADDING A PREFIX TO DELEGATES

The `Module#delegate` now has a new `:prefix` option that can be used when you want the target class's name to be prefixed to the method name. Let's take a look at two examples. First, here is the way you usually work with delegates:

```
class Invoice < ActiveRecord::Base
  delegate :street, :city, :name, :to => :client
end

Invoice.new.street
Invoice.new.city
Invoice.new.name
```

And here is an example that uses the new `:prefix` option:

```
class Invoice < ActiveRecord::Base
  delegate :street, :city, :name, :to => :client, :prefix => true
end

Invoice.new.client_street
Invoice.new.client_city
Invoice.new.client_name
```

You can also use a custom name for the prefix:

```
class Invoice < ActiveRecord::Base
    delegate :street, :city, :name, :to => :client, :prefix => :customer
end

Invoice.new.customer_street
```

```
Invoice.new.customer_city
Invoice.new.customer_name
```

Chapter 3

ActiveResource

RECOVERING THE LAST RECORD FROM ACTIVERESOURCE

Following the `ActiveRecord` convention, `ActiveResource`'s find method now accepts the `:last` option:

```
Person.find(:last, :from => :managers)
# => GET /people/managers.xml
```

You can use this to recover the last item in a collection of resources.

ACTIVERESOURCE::BASE #TO_XML AND #TO_JSON

Two new methods have been added to `ActiveResource::Base`: `to_xml` and `to_json`. The first converts the resource into an XML string, and the second returns a string in JSON format.

The `to_json` method can be configured using the optional hash-based parameters `:only` and `:except`, which allow you to restrict or remove certain attributes. For example:

```
person = Person.new(:first_name => "Carlos", :last_name => "Brando")
person.to_json
# => {"first_name": "Carlos", "last_name": "Brando"}

person.to_json(:only => ["first_name"])
# => {"first_name": "Carlos"}

person.to_json(:except => ["first_name"])
# => {"last_name": "Brando"}
```

Chapter 4

ActionPack

NEW :LAYOUT OPTION FOR THE CACHES_ACTION METHOD

An option called :layout has been added to the `caches_action` method.

```
class ListsController < ApplicationController
  ...

  caches_action :index, :layout => false

  ...
end
```

In the example above I specified `:layout => false`, which means that only the content of the action and not the layout will be stored in the cache. This is very useful for when you have dynamic content in your layout (which is usually the case).

If you don't specify anything, it will continue to function the way it did in the past, which is to cache everything.

TEMPLATES IN THE CACHE

To improve Rails performance, template pages will be automatically stored in the cache for the production environment.

This means that if you alter a template in production, you will need to restart the server for it to take effect.

CHANGES IN THE CONCAT METHOD

If you are in the habit of avoiding repetition in your views by making helpers, you have no doubt used the `concat` method. If you have never used this method, it is basically the equivalent of `puts` for views.

The current implementation of this method accepts two parameters, a `String` with the text that will be displayed in the view and another parameter called `binding`. Due to certain improvements that have been made in the code, while it still accepts these two parameters, the `binding` argument is no longer necessary. This parameter has been deprecated, which means that starting with version 2.2, you will be warned if you use it, and it will be altogether removed at some future date.

The warning method you are shown is as follows: "The binding argument of #concat is no longer needed. Please remove it from your views and helpers."

LINK_TO METHOD NOW ACCEPTS BLOCKS

The `link_to` method has been updated to work with blocks. This is useful when you have really long strings in your links. For example, while before you might have done something like this:

```
<%= link_to "<strong>#{@profile.name}</strong> -- <span>Check it out!!</span>", @profile %>
```

You can now do something like this, with the same result:

```
<% link_to(@profile) do %>
  <strong><%= @profile.name %></strong> -- <span>Check it out!!</span>
<% end %>
```

It is not a significant change in functionality, but it does allow you to write more legible code, which is also important.

RJS#PAGE.RELOAD

The `reload` method has been included in `ActionView::Helpers::PrototypeHelper` so that it can be used in **.rjs** templates or `render(:update)` blocks. This method forces the current page to be reloaded in the browser using javascript. In other words, it is a shortcut for the widely-used `window.location.reload();` technique.

Here's how you use it:

```
respond_to do |format|
  format.js do
    render(:update) { |page| page.reload }
  end
end
```

GIVING A NAME TO THE LOCAL VARIABLE IN A COLLECTION OF PARTIALS

In the code below, a `partial` is being used with a collection of objects:

```
render :partial => "admin_person", :collection => @winners
```

Inside the `partial` it then becomes possible to use the `admin_person` variable to access the elements of the collection. But this variable naming convention is kind of lousy.

You now have the option of choosing a custom name for this variable using the `:as` option. Let's change that last example:

```
render :partial => "admin_person", :collection => @winners, :as => :person
```

Now each item in the collection can be accessed using the more intuitive `person` variable.

PARTIALS NO LONGER DEFINE LOCAL VARIABLES IMPLICITLY

In the example below, I am rendering a partial, and I'm not indicating which variable it should use to fill in the content. Previously, Rails would assume that it should use the instance variable of the same name as the partial.

```
@customer = Customer.new("Carlos Brando")
render :partial => "customer"
```

This works, but it is a little risky. Starting with Rails 2.2, this functionality still works but now emits a deprecation warning:

"@customer will no longer be implicitly assigned to customer"

You can expect this feature to be removed completely in the future.

POLYMORPHIC_URL NOW WORKS WITH SINGLETON RESOURCES

For more information about singular resource routes, see the "Additional Information" chapter at the end of this book.

Until now, the `polymorphic_url` helper did not work properly with singleton resources.

A new patch has been made to allow a singleton resource to be specified with symbols, like you can with namespaces. For example:

```
# This code
polymorphic_url([:admin, @user, :blog, @post])

# is the same as
admin_user_blog_post_url(@user, @post)
```

SUPPORT FOR REGULAR EXPRESSIONS IN TIME_ZONE_SELECT

In `ActiveSupport`'s `TimeZone` class there is a `us_zones` method that conveniently returns a dynamically-generated list of all US time zones.

The problem is that sometimes you want to develop software for people in other countries, but there is no such convenient method that lists these other countries' time zones.

There was a long debate over whether or not to create methods like `african_zones`, `american_zones`, etc. Eventually, the following solution prevailed.

Support for =~ was added to `TimeZone` so that it would be possible to generate a list of time zones from a regular expression. The `time_zone_select` method has also been updated to work with these changes.

You can now do something like this:

```
<%= time_zone_select( "user", 'time_zone', /Australia/) %>
```

The code above returns all time zones, but puts the following time zones at the top of the list:

```
(GMT +08:00) Perth
(GMT +09:30) Adelaide
(GMT +09:30) Darwin
(GMT +10:00) Brisbane
(GMT +10:00) Canberra
(GMT +10:00) Hobart
(GMT +10:00) Melbourne
(GMT +10:00) Sydney
```

To learn more about `TimeZones`, I suggest that you watch Episode 106 of RailsCasts (http://railscasts.com/episodes/106) and also take a look at my last book.

RENDER :TEMPLATE NOW ACCEPTS :LOCALS

The `render :action` and `render :partial` methods allow you to pass a `Hash` containing the data you want them to display using the `:locals` option, but `render :template` used to not allow this.

In this version of Rails it works now:

```
render :template => "weblog/show", :locals => {:customer => Customer.new}
```

:USE_FULL_PATH IN THE RENDER METHOD IS NO LONGER AVAILABLE

```
render :file => "some/template", :use_full_path => true
```

The `:use_full_path` option on the `render` method is no longer available in Rails 2.2. This is not anything serious, since it was not necessary.

DEFINE_JAVASCRIPT_FUNCTIONS

Yet another method is no longer available in Rails: `define_javascript_functions`.

This makes sense, since `javascript_include_tag` and others accomplish the same thing in a better way.

DISABLING THE ACCEPT HEADER IN HTTP REQUESTS

Let's start with some example code:

```
def index
  @people = Person.find(:all)

  respond_to do |format|
    format.html
    format.xml { render :xml => @people.to_xml }
```

```
    end
  end
```

In the example above, Rails has two ways of identifying which format should be used for the `respond_to` block. The first and most common is through the format indicated in the URL (/index.xml, for example), and the second is when the format is not specified, in which case it consults the HTTP request's **Accept** header.

For those who may not know, the **Accept** header is used to indicate what types of documents (often called MIME Types: http://en.wikipedia.org/wiki/MIME) the browser prefers using strings like:

```
Accept: text/plain, text/html
Accept: text/x-dvi; q=.8; mxb=100000; mxt=5.0, text/x-c

# recuperando esta informação via código
@request.env["HTTP_ACCEPT"] = "text/javascript"
```

Consult this URL to see a list of some of the most common MIME types: http://www.developershome.com/wap/detection/detection.asp?page=httpHeaders

This header is implemented inefficiently on many browsers, and when it is used on public web sites, sometimes strange errors occur when indexing robots perform their HTTP requests.

Thus, the decision was made to disable this header by default. It's always better to indicate the desired format in your URLs, but if you need to enable this header, just include the following line in your **environment.rb**:

```
config.action_controller.use_accept_header = false
```

When disabled, if the format is not indicated by the URL, Rails will assume that it should use **.html** format.

There is a special case when you make ajax requests using the **X-Requested-With** header. In this case, the **.js** format will be used even if the format was not specified (/people/1, for example).

BUTTON_TO_REMOTE

If you have been using Rails for a while, you probably are familiar with the `submit_to_remote` method. This method returns an HTML button that will send a form using **XMLHttpRequest**.

In Rails 2.2 this method has been renamed to `button_to_remote`, to maintain consistency with the name of its sibling method, `link_to_remote`.

Those who plan to update their projects don't need to worry, since the old name (`submit_to_remote`) will be an alias to the new one.

RECEIVING WARNINGS FOR IMPROVING PERFORMANCE

Rails has gotten a configuration parameter that makes it emit warnings if it renders a template outside of the specified views directory. This is great because files outside of the specified views directories are not stored in the cache, which results in more disk accesses.

To start receiving warnings about this, just include the following line in your project's **environment.rb** file:

```
config.action_view.warn_cache_misses = true
```

Doing this will result in the following warning if a file outside the configured directories is rendered:

```
[PERFORMANCE] Rendering a template that was not found in view path. Templates outside the view path
are not cached and result in expensive disk operations. Move this file into /Users/user/project/app/
views or add the folder to your view path list
```

This option is disabled by default.

:INDEX OPTION NOW WORKS IN FIELDS_FOR

Often the `:index` option of the `select` method is helpful, such as when you need to create many different dynamic select controls on a single page.

Until now, the `fields_for` method didn't re-pass this option to methods like `select`, `collection_select`, `country_select` and `time_zone_select`, which limited its usefulness in certain circumstances.

This has been corrected in this version of Rails. For example, check out this code:

```
fields_for :post, @post, :index => 108 do |f|
  concat f.select(:category, %w( abe <mus> hest))
end
```

This will return:

```
<select id=\"post_108_category\" name=\"post[108][category]\">
  <option value=\"abe\">abe</option>
  <option value=\"&lt;mus&gt;\" selected=\"selected\">&lt;mus&gt;</option>
  <option value=\"hest\">hest</option>
</select>
```

Note that I am using the `:index => 108` option in the `fields_for` method. Take a look at the `id` and `name` properties of the tag generated by the `select` method. Although no options were passed to this method, the index gets added to its output too.

IMPROVING PERFORMANCE WITH ETAGS

Before discussing this, allow me to explain what ETags (Entity Tags) are. A crude explanation of ETags is that they are identifiers that get associated with each resource to determine if a file that is on the server is the same as one that is in a browser's cache. In many cases it is an HTML page, but it could also be XML or JSON.

The server is in charge of verifying that the requested resource is the same on both sides. If the server can verify this, instead of returning the resource's entire content, it only needs to return a **304** (Not Modified) status, which tells the browser to use the file in its cache.

Web servers like Apache and IIS already know how to do this for static pages, but when the content is dynamic, as is the case for most pages in a Ruby on Rails project, this is your responsibility.

The `response` object has two new attributes: `last_modified` and `etag`. When values are assigned to these attributes, they are automatically passed to the `HTTP_IF_MODIFIED_SINCE` and `HTTP_IF_NONE_MATCH` headers, respectively. When a new request for a resource is made, the `request` object will return with these headers set, allowing you to compare them with the current value and determine if the user's cache is as recent as the resource's content. If the user's cached version is the most recent one, instead of rendering the resource again, you can simply return a "304 Not Modified" status, which causes the browser to display the cached version.

To do this, two methods are available, depending on the situation: `stale?` and `fresh_when`.

Let's look at an example:

```
class ArticlesController < ApplicationController
  def show
    @article = Article.find(params[:id])

    if stale?(:last_modified => @article.published_at.utc, :etag => @article)
      respond_to do |wants|
        wants.html
        wants.xml { render :xml => @article }
      end
    end
  end
end
```

In the example above, if the `request` headers' values are different from the ones referred to in the call to `stale?`, this means that the user's cache is not up-to-date, so the `respond_to` block is called and the values that were passed to `stale?` are assigned to the `response` object's `last_modified` and `etag` attributes.

If the values are equal, this means that the user's cache is up-to-date. The `respond_to` block is not called, and just a "304 Not Modified" status is returned.

We also have the `fresh_when` method, which is a simpler version of the `stale?` method. Check out this example:

```
def show
  @article = Article.find(params[:id])
  fresh_when(:etag => @article, :last_modified => @article.created_at.utc)
end
```

Basically, this method assigns the values it receives to their respective attributes in the `response` object and verifies that they are equal to those in the `request` object. If they are different (stale), it renders the appropriate template. If they are equal (fresh), then it returns a "304 Not Modified" status instead of rendering the template.

In some situations it may be necessary to pass an `Array` to the `:etag` option, such as in the following example:

```
fresh_when(:etag => [@article, current_user], :last_modified => @article.created_at.utc)

# or

if stale?(:last_modified => @article.published_at.utc, :etag => [@article, current_user])
    # ...
end
```

CHANGES TO THE NUMBER_WITH_DELIMETER METHOD'S SIGNATURE

The `number_with_delimiter` method has been reimplemented. Besides making the code cleaner, the method's signature also changed. Check out how it was before:

```
def number_with_delimiter(number, delimiter=",", separator=".")

# usage examples
number_with_delimiter(12345678) # => 12,345,678
number_with_delimiter(12345678.05) # => 12,345,678.05
number_with_delimiter(12345678, ".") # => 12.345.678
number_with_delimiter(98765432.98, " ", ",")
```

Here is the new version:

```
def number_with_delimiter(number, *args)

# usage examples
number_with_delimiter(12345678) # => 12,345,678
number_with_delimiter(12345678.05) # => 12,345,678.05
number_with_delimiter(12345678, :delimiter => ".") # => 12.345.678
```

```
number_with_delimiter(12345678, :seperator => ",") # => 12,345,678
number_with_delimiter(98765432.98, :delimiter => " ", :separator => ",")
```

So make sure, when you use the `number_with_delimiter` method, that you specify the options with symbolized hash keys.

EXECUTING MULTIPLE INSTANCES OF A PROJECT IN SUBDIRECTORIES

Sometimes you need to run multiple copies of the same project. Maybe you have a product that will be used by various clients, or maybe you just want to run a test version and a production version of your app at the same time.

The easiest way of doing this is to have multiple (sub)domains, each with its own instance of the app. But if this isn't possible, you can put your application in a subdirectory and use a prefix on its URL to distinguish each instance of your app. For example, you could run blogs for different users with URLs like:

- http://www.nomedojogo.com/johndoe/blog
- http://www.nomedojogo.com/jilldoe/blog
- http://www.nomedojogo.com/joedoe/blog

In these cases, the prefixes **johndoe**, **jilldoe** and **joedoe** identify the instances of the app that are running in corresponding subdirectories of the same name. The application routing happens after this. You can tell Rails to ignore this part of the URLs when a request is made, but still put it on the URLs that it generates. This can be configured using the `RAILS_RELATIVE_URL_ROOT` constant or the `AbstractRequest.relative_url_root` method.

If your Rails project is running on Apache, however, this feature is already activated automatically, so in many cases it is not necessary to worry about configuring this. Once again, this is only if you are using Apache.

In Rails 2.2, however, the `relative_url_root` will not be automatically configured by the HTTP header. This will need to be done manually, putting a line in each of your apps' **environment.rb** files that looks something like this:

```
config.action_controller.relative_url_root = "/johndoe"
```

That done, your application will ignore the "johndoe" prefix that occurs after the domain. But when it generate URLs it will always put this prefix in to guarantee that you will be accessing the project in the right subdirectory.

CHANGES IN ERROR_MESSAGE_ON

The `error_message_on` message is very useful. You can use it to easily display error messages returned by certain methods in an object.

```
<%= error_message_on "post", "title" %>

<!-- or -->

<%= error_message_on @post, "title" %>
```

This will cause an error message to be displayed within a DIV tag if an error is associated with the title field of the Post model.

But the most interesting thing about the `error_message_on` method is that we can personalize it to display friendlier messages. Here is where the changes in Rails 2.2 come in.

In the current version, the personalization parameters are passed directly to the method, but in Rails 2.2 they will be passed as an option Hash:

```
<%= error_message_on "post", "title",
      :prepend_text => "Title simply ",
      :append_text => " (or it won't work).",
      :css_class => "inputError" %>
```

Don't worry about having to port your current projects for these changes, because the code still works in the old way too (at least for a while), but with a warning that you need to update your code.

MORE METHODS UPDATED TO ACCEPT OPTION HASHES

The following methods have also been modified to accept their arguments in `Hash` format, making your code more readable and easy to maintain.

truncate

```
truncate("Once upon a time in a world far far away")
# => Once upon a time in a world f...

truncate("Once upon a time in a world far far away", :length => 14)
# => Once upon a...

truncate("And they found that many people were sleeping better.", :omission => "... (continued)",
    :length => 15)
# => And they found... (continued)
```

highlight

```
highlight('You searched for: rails', ['for', 'rails'], :highlighter => '<em>\1</em>')
# => You searched <em>for</em>: <em>rails</em>
```

```
highlight('You searched for: rails', 'rails', :highlighter => '<a href="search?q=\1">\1</a>')
# => You searched for: <a href="search?q=rails">rails</a>
```

excerpt

```
excerpt('This is an example', 'an', :radius => 5)
# => ...s is an exam...

excerpt('This is an example', 'is', :radius => 5)
# => This is a...

excerpt('This next thing is an example', 'ex', :radius => 2)
# => ...next...

excerpt('This is also an example', 'an', :radius => 8, :omission => '<chop> ')
# => <chop> is also an example
```

word_wrap

```
word_wrap('Once upon a time', :line_width => 8)
# => Once upon\na time

word_wrap('Once upon a time', :line_width => 1)
# => Once\nupon\na\ntime
```

auto_link

```
post_body = "Welcome to my blog at http://www.nomedojogo.com/. Please e-mail me at me@email.com."

auto_link(post_body, :urls)
```

```
# => "Welcome to my blog at
    <a href=\"http://www.nomedojogo.com/\">http://www.nomedojogo.com</a>.
    Please e-mail me at me@email.com."

auto_link(post_body, :all, :target => "_blank")
# => "Welcome to my blog at
    <a href=\"http://www.nomedojogo.com/\" target=\"_blank\">http://www.nomedojogo.com</a>.
    Please e-mail me at <a href=\"mailto:me@email.com\">me@email.com</a>."

auto_link(post_body, :link => :all, :html => {:target => "_blank"})
# => "Welcome to my blog at
    <a href=\"http://www.nomedojogo.com/\" target=\"_blank\">http://www.nomedojogo.com</a>.
    Please e-mail me at <a href=\"mailto:me@email.com\">me@email.com</a>."
```

All these methods continue working in the old way for now, but display warnings in the application log (and terminal, in development mode) to remind you to update your code as soon as possible.

MORE FEATURES FOR PARTIALS

Take a look at the following example:

```
<!-- _layout.html.erb -->
beginning
<%= yield %>
ending

<!-- some view -->
<%= render :layout => 'layout' do %>
middle
<% end %>
```

The output of this code would be:

```
beginning
middle
ending
```

In the example above I'm including a partial inside my view and using the `yield` method to customize the content, which is specified inside a block that gets passed to the `render` method.

Until now, however, you couldn't pass any arguments to the block. In Rails 2.2 this is now possible. You can do some pretty cool stuff. Check out this example involving a collection of books:

```
<!-- app/views/books/_book.html.erb -->
<div class="book">
  Price: $<%= book.price %>
  <%= yield book %>
</div>

<!-- app/views/books/index.html.erb -->
<% render :layout => @books do |book| %>
  Title: <%= book.title %>
<% end %>
```

This would return something like:

```
<div class="book">
  Price: $29.74
  Title: Advanced Rails
</div>
```

Notice how, inside the block, I'm specifying the book's title, but in another view I can also add the remaining stock or other relevant information, while always using the same view.

You can also use the same partial many times in the same view and use blocks to differentiate between page sections, for example:

```
<!-- app/views/books/_book.html.erb -->
<div class="book">
  <%= yield book, :header %>
  Price: $<%= book.price %>
  <%= yield book, :footer %>
</div>

<!-- app/views/books/index.html.erb -->
<% render :layout => @books do |book, section| %>
  <%- case section when :header -%>
    Title: <%= book.title %>
  <%- when :footer -%>
    (<%= book.reviews.count %> customer reviews)
  <%- end -%>
<% end %>
```

RETRIEVING THE CURRENT STRING FROM THE CYCLE METHOD

You probably are already familiar with the `cycle` method. It is used to alternate the colors of table rows, changing each row's CSS class.

```
@items = [1,2,3,4]
<table>
  <% @items.each do |item| %>
  <tr class="<%= cycle("even", "odd") -%>">
    <td>item</td>
  </tr>
```

```
  <% end %>
</table>
```

A new auxiliary method has been created to help when using `cycle` in more complex tables or other situations where it may be necessary to retrieve the class string for the current iteration. Look at this similar example to the one before, which uses the new `current_cycle` method:

```
@items = [1,2,3,4]
<% @items.each do |item| %>
<div style="background-color:<%= cycle("red", "white", "blue") %>">
  <span style="background-color:<%= current_cycle %>"><%= item %></span>
</div>
<% end %>
```

AVOIDING DUPLICATE ARTICLES IN FEEDS

Sometimes you subscribe to a blog feed and suddenly face a bunch of posts that have already been read but look new to your feed reader. Has this ever happened to you? There are a number of reasons for this, but of course, you wouldn't want it to happen to readers of your own blog, would you?

To help us avoid this annoyance, each entry in a feed generated by the `atom_feed` builder now has a new `:id` option, which allows you to customize the id.

```
atom_feed({ :id => 'tag:nomedojogo.com,2008:test/' }) do |feed|
  feed.title("My great blog!")
  feed.updated((@posts.first.created_at))

  for post in @posts
    feed.entry(post, :id => "tag:nomedojogo.com,2008:" + post.id.to_s) do |entry|
      entry.title(post.title)
```

```
      entry.content(post.body, :type => 'html')

      entry.author do |author|
        author.name("Carlos Brando")
      end
    end
  end
end
```

If you do it this way, even if you have to rewrite some portion of the code that generates your feeds, or make some major alteration in your site's content, the `id` created for each entry will remain the same, causing users' feed readers not to duplicate old entries.

Your readers will thank you.

ADDING PROCESSING INSTRUCTIONS TO XML DOCUMENTS

A new option has been included in the `atom_feed` method. You can now include XML processing instructions. Here is an example:

```
atom_feed(:schema_date => '2008', :instruct => {
    'xml-stylesheet' => {
        :href=> 't.css', :type => 'text/css'
    }
}) do |feed|
    # ...
end
```

XML processing instructions are read by the application that requested the file. They are usually used to inform the application how it should manipulate the XML data.

In the example above, I'm telling the application receiving the XML that it should display the XML with a specific CSS stylesheet. Here is the result:

```
<?xml-stylesheet type="text/css" href="t.css"?>
```

SIMPLIFYING RESOURCE ACCESS

Nested routes have been around for a while. As you build your routes, it's not uncommon to do something like this:

```
map.resources :users do |user|
  user.resources :posts do |post|
    post.resources :comments
  end
end
```

In the code above, I am making it clear that my users have posts, and these posts have comments. In accordance with the way my routes have been configured, I can retrieve a user's posts using the URL '**/users/1/posts**', or retrieve a particular post with the URL '**/users/1/posts/5/comments**'.

With the new `:shallow => true` option, we gain more flexibility. Note that when you add this option to the first resource, all other resources within it will inherit this characteristic.

```
map.resources :users, :shallow => true do |user|
  user.resources :posts do |post|
    post.resources :comments
  end
end
```

With this option enabled, I can keep retrieving data like I did before. The advantage is that I can now also retrieve all the comments for a post without specifying a user, with the following URL: '**/posts/2/comments**'. Or I can retrieve a certain post using only '**/posts/2**'.

The `:shallow` option also works with resources that use `has_many` or `has_one`, as follows:

```
map.resources :users, :has_many => { :posts => :comments }, :shallow => true
```

All the helpers for accessing the routes directly also get created, such as:

```
user_posts_path(@user) # => '/users/1/posts'
posts_path # => '/posts'
post_comments_path(@post) # => /posts/5/comments
```

DEFINING ROUTES WITH ARRAYS OF ROUTING VERBS

You can now pass an array of methods to new members or collections of routes. This eliminates the need for defining a route that will accept any verb (`:any`) when you actually only want it to respond to more than one. In Rails 2.2 you can declare a route like this:

```
map.resources :photos, :collection => { :search => [:get, :post] }
```

POLYMORPHIC ROUTES

The `*_polymorphic_url` and `*_polymorphic_path` methods, which are highly used to generate URLs from database records, now have an optional parameter. In addition to the usual parameters, they new also accept an option hash, making it possible to generate new routes with additional parameters in the URL.

Let's check out the examples. Under each example is a commented-out equivalent method

```
edit_polymorphic_url(@article, :param1 => '10')
# => edit_article_url(@article, :param1 => '10')

polymorphic_url(@article, :param1 => '10')
# => article_url(@article, :param1 => '10')

polymorphic_url(@article, :format => :pdf, :param1 => '10')
# => formatted_article_url(@article, :pdf, :param1 => '10')
```

COUNTRY_SELECT REMOVED FROM RAILS

The `country_select` helper has been removed from Rails. For those who don't remember, this method returns a list with all the countries in the world.

The reason for this removal is that Rails has been updated to use the ISO 3166 standard for country names. The whole problem is that, according to the ISO 3166 standard, its actual name is "Taiwan, Province of China." And that was exactly how Michael Koziarski put it in the method.

Then, Jamis Buck asked if it might not be possible to render it simply as "Taiwan," since the "Province of China" part seemed politically aggressive. A series of comments ensued on Github that got more and more political and set the technical issues aside.

But Michael Koziarski has been categoric in affirming that these political issues are way beyond what we could hope to resolve with a simple code change. And if he accepted this alteration, soon others might be requested for countries like Kosovo, South Ossetia, Abecassia, Transnistria and a long list of others.

The better solution, or at least the one that would be less controversial, would be to remove the helper from Rails and make it available in the form of a plugin. This way, people can easily fork the code and create their own lists in whatever way they prefer.

To install the official plugin:

```
./script/plugin install git://github.com/rails/country_select.git
```

BENCHMARKING REPORTS IN MILLISECONDS

All log messages that reported the time that a process took to be executed have been modified to display milliseconds.

For example, the message:

```
"Completed in 0.10000 (4 reqs/sec) | Rendering: 0.04000 (40%) | DB: 0.00400 (4%) | 200 OK"
```

will now be displayed as follows:

```
"Completed in 100ms (View: 40, DB: 4) | 200 OK"
```

:CONFIRM OPTION ON THE IMAGE_SUBMIT_TAG METHOD

The `:confirm` option that is used in many helpers, like `link_to`, will now be available in the `image_submit_tag` method.

This option causes a confirmation box with a personalized message to be displayed when you click on the image. If the user responds in the affirmative, the form is submitted normally. If not, nothing happens.

```
image_submit_tag("delete.gif", :confirm => "Are you sure?")
# => <input type="image" src="/images/delete.gif"
#                        onclick="return confirm('Are you sure?');"/>
```

SESSION COOKIES ARE NOW HTTP ONLY

Rails has an option for setting cookies that is often overlooked. The `:http_only` option causes a cookie to be accessible only through HTTP, preventing it from being accessed by javascript code. The default value for this is `false`.

In Rails 2.2, session cookies will have the `http_only` option set by default. The purpose of this feature is to increase the security of your projects. Of course, this option can be disabled if necessary. If you wish to do this, just include the following line in your `ApplicationController` or in a specific controller:

```
session :session_http_only => false
```

COMPLETE VIEW PATH DURING EXCEPTIONS

When an exception is thrown in development mode, your view shows a message like this:

```
NoMethodError in Administration/groups#show
Showing app/views//_list.erb where line ...
```

But it would be nice to get a messsage with the complete path of the file that threw the exception, like this:

```
NoMethodError in Administration/groups#show
Showing app/views/administration/reports/_list.erb where line ...
```

This problem has been corrected in this version of Rails, making your work easier.

OPTIONS IN THE FIELD_SET_TAG METHOD

The `field_set_tag` method has gotten a new option parameter that makes HTML formmating easier. This parameter accepts all the options that the `tag` method accepts. For example:

```
<% field_set_tag nil, :class => 'format' do %>
  <p>Some text</p>
<% end %>
```

This code will return the following:

```
<fieldset class="format">
  <p>Some text</p>
</fieldset>
```

XHTML SUPPORT IN ATOM_FEED

The `atom_feed` helper now has an internal builder that allows you to create XHTML by simply adding `:type=>"xhtml"` to any `content`, `rights`, `title`, `subtitle` or `summary` element. For example:

```
entry.summary(:type => 'xhtml') do |xhtml|
  xhtml.p "A XHTML summary"
  xhtml.p post.body
end
```

Take a look at how this block fits inside the `atom_feed` call:

```
atom_feed do |feed|
  feed.title("My great blog!")
  feed.updated((@posts.first.created_at))
```

```
  for post in @posts
    feed.entry(post) do |entry|
      entry.title(post.title)

      entry.summary(:type => 'xhtml') do |xhtml|
        xhtml.p "A XHTML summary"
        xhtml.p post.body
      end

      entry.author do |author|
        author.name("DHH")
      end
    end
  end
end
```

This way, `atom_feed`'s internal builder will include the generated XHTML inside a **div** tag.

Of course, you can still use the old way of passing all the HTML inside a string, but this way your code is cleaner.

AVOIDING RESPONSE SPLITTING ATTACKS

Until now, the URLs passed to the `redirect_to` method in Rails did not go through a sanitization process. This was dangerous, because it made it possible for malicious users to conduct **response splitting** and **header injection** attacks in your application.

An example of this vulnerability is when your app receives a URL through a query string and redirects the user to this URL using the `redirect_to` method. This breach allows malicious users to store cookies on your users' machines and create fake responses to your users if your project uses these parameters to construct HTTP headers.

To avoid these kinds of problems, Rails has been updated to sanitize all URLs passed to the `redirect_to` method. This doesn't mean, however, that you don't need to worry about these issues any more. It's always good to double-check things and stay alert.

A BETTER WAY OF CATCHING ERRORS

The worst thing that can happen in your application is that horrible error page. It's always good to be prepared for this situation. Now you can easily display a custom error page when your app raises an exception, using the `rescue_from` method. Check out this example:

```
class ApplicationController < ActionController::Base
  rescue_from User::NotAuthorized, :with => :deny_access
  rescue_from ActiveRecord::RecordInvalid, :with => :show_errors

  protected
  def deny_access
    …
  end

  def show_errors(exception)
    exception.record.new_record? ? …
  end
end
```

The addition of `ActiveSupport:Rescuable` allows any class to mix in the `rescue_from` functionality.

NEW :RECURSIVE OPTION FOR JAVASCRIPT_INCLUDE_TAG AND STYLESHEET_LINK_TAG

The `javascript_include_tag` and `stylesheet_link_tag` helpers have a new `:recursive` option that can be used together with `:all`, which allows them to load an entire file tree in one line of code. For example, let's say I have the following files:

```
public/javascripts/super_calendar/calendar.js
public/stylesheets/super_calendar/calendar.css
```

Both get included, even if they are in subdirectories, when I execute the following code:

```
javascript_include_tag :all, :recursive => true
stylesheet_link_tag :all, :recursive => true
```

Chapter 5

ActionMailer

USING LAYOUTS IN EMAILS

With this new Rails release we will have a new feature that is sure to delight the spammers!

Joking aside, just as we have layouts for controllers, we now have them for mailers too. The only thing you need to do is add `_mailer` to the end of your filenames, which will cause the layout to be applied automatically.

Let's take a look at the examples. First, the views:

```
<!-- layouts/auto_layout_mailer.html.erb -->
Hello from layout <%= yield %>

<!-- auto_layout_mailer/hello.html.erb  -->
Inside
```

Now let's look at `ActionMailer`:

```
# auto_layout_mailer.rb
class AutoLayoutMailer < ActionMailer::Base
  def hello(recipient)
    recipients recipient
    subject "You have a mail"
    from "tester@example.com"
  end
end
```

Doing it this way causes any new mailer to use this layout.

To avoid having the mailer use the default layout, all you have to do is include the `:layout => false` clause in the email body.

```
def nolayout(recipient)
  recipients recipient
  subject "You have a mail"
  from "tester@example.com"
  body render(:inline => "Hello, <%= @world %>", :layout => false,
        :body => { :world => "Earth" })
end
```

You can also specify a different layout for the mailer by changing the `:layout` option. In this case, the layout you use does not need to have the `_mailer` suffix.

```
def spam(recipient)
  recipients recipient
  subject "You have a mail"
  from "tester@example.com"
  body render(:inline => "Hello, <%= @world %>", :layout => 'spam',
```

```
        :body => { :world => "Earth" })
end
```

You can also tell `ActionMailer` which layout it should use like this:

```
class ExplicitLayoutMailer < ActionMailer::Base
  layout 'spam', :except => [:logout]
```

Chapter 6

Railties

ARE PLUGINS A THING OF THE PAST?

In Rails 2.1, it became possible to use gems as plugins in your projects. To do this, you just had to create a folder called **rails** inside of the gem's project and include a file called **init.rb**.

This brought with it a bunch of new things like `config.gem` and `rake:gems`. But it makes you wonder, since it's now possible to load gems inside your Rails application, could it only be a matter of time before plugins disappear?

And it looks like this might actually happen. In this version of Rails, for example, a change has been made that allows you to initialize plugins either using **init.rb** in plugin's root directory, or in **rails/init.rb** (the same way you do with gems). This second option is the preferred method.

So, I could for example make a gem (which would function as a plugin) and install it in two ways:

```
./script/plugin install git://github.com/user/plugin.git

sudo gem install user-plugin --source=http://gems.github.com
```

This also prevents you from having to maintain two **init.rb** files (one in the plugin's root and the other in the rails directory).

IMPROVED THIN SUPPORT

`script/server` now checks to see if **Thin** is available and, if so, attempts to use it. This is very convenient if you are using **Thin** in your production environment (and wish to use the same thing for development). You will need to add the following line in your **environment.rb** file for this to work:

```
config.gem 'thin'
```

TESTING ONLY UNCOMMITTED FILES WHEN USING GIT

There is a very useful **rake** task in Rails that few people know about:

```
rake test:uncommitted
```

As is probably obvious from the name, it runs only those tests whose files haven't yet been committed to the subversion repository, instead of running all project tests.

I used this feature all the time, but when I switched to Git it wasn't available, as it was only supported for subversion. A recent patch, however, guarantees that from here on out we'll also have it when using Git.

RAILS.INITIALIZED?

I new method called `Rails.initialized?` has been added to Rails. It lets you know if all the initialization-related routines have finished executing.

CACHE_CLASSES NOW ENABLED BY DEFAULT

In your project's configuration files there is probably the following line:

```
config.cache_classes = false
```

This tells Rails that it shouldn't cache your project's code; that is, for each request that comes in, it shouldn't reload all the code again. Although this slows down execution, it's ideal for development because it keeps you from having to restart the server every time you change something.

In production mode it's critical that you keep this enabled.

In Rails 2.1 projects, if the line above is not in your configuration file, Rails will assume that it should not cache classes—this is the default behavior.

In Rails 2.2 this has been inverted; if no configuration option is found, it will assume that it should cache classes. This helps inexperienced Rails users not to deploy their projects in development mode.

MAKING CONFIG.GEM NOT LOAD THE GEM

One of the new features introduced in Rails 2.1 was `config.gem`, which allows you to specify which gems your project depends on.

With this feature we gain a number of tasks that make our work easier, such as `rake gems:install`, which install all dependencies automatically.

But you had to be careful when configuring your dependencies, because in some cases the name of the gem isn't the name of the file that should be passed to `require`. For example, the **aws-s3** gem should be loaded with the name **aws/s3**, replacing the dash with a slash. Anticipating this situation, a `:lib` option has been added to `config.gem`. This is how it would look for the example above:

```
config.gem 'aws-s3', :lib => 'aws/s3'
```

After a while another problem came up: "My project depends on this gem, but I don't want it to be loaded right now. What do I do?"

This new version allows us to set the `:lib` option to `false`, which causes Rails not to load the gem.

```
config.gem 'aws-s3', :lib => false
```

Even without loading the gem, it will still be installed when you run `rake gems:install`, and it will be included in any other task related to gems.

THREADSAFE!

Rails now has a new configuration method to enable threaded mode:

```
config.threadsafe!
```

When this method is run in your configuration file (usually in **environments/production.rb**), you will be causing controller actions to be able to accept concurrent requests and connect to multiple database connections. Depending on your application and deployment environment, this may also allow your application to handle more requests, since fewer copies of Rails will be loaded in memory.

This also disables post-initialization dependency loading. For more details about this, look up the "Enabling and disabling dependency loading" in the **ActiveSupport** chapter.

FIXTURES_PATH

The `rake db:fixtures:load` and `rake db:fixtures:identify` tasks got a new optional parameter: `FIXTURES_PATH`.

```
rake db:fixtures:load FIXTURES_PATH=spec/fixtures
```

This way you can specify an alternate path for your fixtures (for example: spec/fixtures).

AUTOMATED BELONGS_TO ASSOCIATIONS IN RAILS SCAFFOLD GENERATOR

If you are using Rails 2.2, try to execute the following command to create a new model:

```
./script/generate scaffold comment author:string body:text post:references
```

Note that this specifies that comments reference the `posts` table. Or, in other words, my comments belong (`belongs_to`) a post. Now take a look at the **app/models/comment.rb** that got generated:b** gerado pelo script:

```
class Comment < ActiveRecord::Base
  belongs_to :post
end
```

The relationship between the two tables has automatically been added to the model. This is a new feature found in this version of Rails.

HELP MESSAGE FOR RAILS NEWBIES

A new message has been added to the 500.html file to help programmers who are just starting to use Rails:

(If you're the administrator of this website, then please read the log file "development.log" to find out what went wrong.)

That's Rails, looking out for the new folks out there.

DEBUGGING IN THE RAILS CONSOLE

Just like you can run Rails with the debugger option using `script/server --debugger`, you can now do it in the console, using `script/console --debugger`. This option pretty much just loads the **ruby-debug** library when the console is initialized.

It's easier to use this option than to have to run `require 'ruby-debug'` in the console every time you need this feature.

DB:MIGRATE:REDO NOW ACCEPTS A MIGRATION VERSION

The `rake db:migrate:redo` task has been very useful for testing and re-executing the most recent migration. This task is now even more useful, allowing you to specify a `VERSION` option and tell it which migration you want to run again:

```
rake db:migrate:redo VERSION=20080725004631
```

SUPPORT FOR IBM DB IN THE RAILS GENERATOR

An option has been added to Rails that allows you to generate a project with IBM DB support. To create a Rails project with this option, just run the following command in your terminal:

```
rails app_name -d ibm_db
```

This will create your project with the following text in your **database.yml** file:

```
# IBM Dataservers
#
# Home Page
# http://rubyforge.org/projects/rubyibm/
#
# To install the ibm_db gem:
# On Linux:
# Source the db2profile file and set the necessary environment variables:
#
```

```
# . /home/db2inst1/sqllib/db2profile
# export IBM_DB_DIR=/opt/ibm/db2/V9.1
# export IBM_DB_LIB=/opt/ibm/db2/V9.1/lib32
#
# Then issue the command: gem install ibm_db
#
# On Windows:
# Issue the command: gem install ibm_db
# If prompted, select the mswin32 option
#
# For more details on the installation refer to http://rubyforge.org/docman/view.php/2361/7682/IBM_DB_GEM.pd
#
# For more details on the connection parameters below refer to:
# http://rubyibm.rubyforge.org/docs/adapter/0.9.0/rdoc/classes/ActiveRecord/ConnectionAdapters/IBM_DBAdapter

development:
  adapter: ibm_db
  username: db2inst1
  password:
  database: app_name_dev
  #schema: db2inst1
  #host: localhost
  #port: 50000
  #account: my_account
  #app_user: my_app_user
  #application: my_application
  #workstation: my_workstation

test:
  adapter: ibm_db
  username: db2inst1
  password:
  database: app_name_tst
  #schema: db2inst1
```

```
  #host: localhost
  #port: 50000
  #account: my_account
  #app_user: my_app_user
  #application: my_application
  #workstation: my_workstation

production:
  adapter: ibm_db
  username: db2inst1
  password:
  database: app_name_prd
  #schema: db2inst1
  #host: localhost
  #port: 50000
  #account: my_account
  #app_user: my_app_user
  #application: my_application
  #workstation: my_workstation
```

Chapter 7

Internationalization (i18n)

Everything started in September of 2007, when a group of developers began building a Rails plugin called **rails-i18n**, whose purpose was to eliminate the necessity of monkey patching Rails in order to internationalize an application.

WHAT IS I18N?

First, to understand the name, it is necessary to have a deep knowledge of mathematics. Count with me how many letters there are between the I and the N in "internationalization"? 18. Very good: i18n.

The same goes for localization and l10n.

Have you already seen a site with little flags at the top that allow you to choose which language you wish to browse the site in? When you click on one of them, all the site's content changes to the corresponding country's language. This is called internationalization, or, as we just learned, i18n.

Of course I'm being a little simplistic here. Most of the time it's not just the text that changes from one country to another. We also can't forget data formats, timezones, units of measurement, and perhaps most importantly, currency.

I18N E L10N, WHAT'S THE DIFFERENCE?

Internationalization is when you prepare your software so that people from other countries and languages can use it.

Localization (l10n) is when you adapt your product to the needs of a certain country, such as, for example, when you take a US site that only accepts Paypal and change it to accept bank payments.

WHAT'S NEW IN RAILS?

In Rails 2.2 this internationalization plugin has been add to the core.

This doesn't mean that Rails has undergone radical changes. Actually, almost nothing has changed. It continues to be the same framework, with all the usual validations in English, etc.

The difference is that if you wanted these messages to be in Portuguese, or in some other language, you would have to create a monkey patch for this. I can't fail to mention the famous Brazilian Rails, which does exactly this to translate ActiveRecord messages.

The main improvement is that in Rails 2.2 we now have a standardized and simpler way of doing this, using a common interface.

HOW DOES THIS WORK?

Basically, this gem is split into two parts:

- The first adds a bunch of new methods to the Rails API. These methods basically allow you to access the second part of the gem.
- The second part is a simple backend that implements all the logic required to make Rails work like it did before, using en-US as the default localization configuration.

The big difference is that this second module can be replaced with another that supports the internationalization that you require. Even better, a series of plugins are coming out that will do just this.

The foundation of this implementation is a module call `i18n`, which comes from a gem built into `ActiveSupport`. This module adds the following features to Rails:

- The `translate` method, which is used to retrieve translations.
- The `localize` method, which is used to "translate" `Date`, `DateTime`, and `Time` objects to the current locale.

Besides these methods, this module contains the code necessary to store and load the localization backends. And it comes with an standard exception handler that catches exceptions raised in the backend.

Both the backend and the exception handler can (and should) be extended. There are also aliases for `#translate` and `#localize` that save you a little time: `#t` and `#l`, respectively. These methods work in both views and controllers.

PRACTICAL EXAMPLES

The best way to understand how to use this internationalization support in Rails is seeing how it works in practice. I recommend taking a look at a project created by Sven Fuchs and others at: **http://i18n-demo.phusion.nl/**.

PLURALIZATION SUPPORT IN INTERNATIONALIZATION METHODS

Sometimes certain translations depend on a number or quantity. This is a pretty common scenario this the internationalization package has taken into account.

For example, in the method `distance_in_words`, the phrase "1 second" works fine when the time is less than or equal to one, but must be pluralized when the time something takes is longer than this.

In the localization file, you can internationalize phrases in your app that depend on a number or quantity like this:

```
datetime:
  distance_in_words:
    x_seconds:
      one: "1 segundo"
      other: "{{count}} segundos"
```

This is a useful feature that is used by many of Rails native methods, and one that you can use in your own methods.

Chapter 8

Performance

IMPROVING RAILS PERFORMANCE

Jeremy Kemper has been working on a number of performance improvements in Rails. For one thing, he has improved the efficiency of **ERb**, as well as some methods like `concat` and `capture`, which are used by many **helpers** in Rails.

Jeremy has also attacked the **partial** initialization process and optimized various helpers that generated **javascript** code.

The `RecordIdentifier` class has also been improved by using caching. `RecordIdentifier` employs a series of conventions to deal with `ActiveRecord` and `ActiveResource` records, or pretty much any other kind of model that has an `id`.

It's interesting to see this kind of work. Rails is getting pretty hefty, so optimization efforts will need to become an ongoing part of future Rails enhancements.

CREATING PERFORMANCE TESTS

Rails 2.2 has a new **generator** that allows you to create performance tests.

Executing the following command in the terminal:

```
[carlosbrando:edge]$ ./script/generate performance_test Login
      exists  test/performance/
      create  test/performance/login_test.rb
```

causes a file called **test/performance/login_test.rb** to be created. Check out the code that gets generated:

```
require 'performance/test_helper'

class LoginTest < ActionController::PerformanceTest
  # Replace this with your real tests.
  def test_homepage
    get '/'
  end
end
```

In this file, you can put all the performance-related tests you want to execute, and when they get executed, you will get a report with important information about each test, including execution time, memory usage and other info. To do this, execute the following command in your terminal:

```
[carlosbrando:edge]$ ruby test/performance/login_test.rb
Loaded suite test/performance/login_test
```

```
Started
LoginTest#test_homepage (32 ms warmup)
        process_time: 11 ms
              memory: unsupported
             objects: unsupported
.
Finished in 0.870842 seconds.
```

Chapter 9

Bugs and Fixes

ACTIVERECORD

Correction to a namespace clash between named_scope and :joins.

When you used `with_scope` together with `:joins`, all the secondary table attributes were added to the model of the main table.

find_all method not working in named_scope

When you executed the `find_all` method in a `named_scope`, the method was not being redirected to the `proxy_found` method as expected. This caused the `NoMethodError` exception to be raised.

```
Topic.base.find_all(&:approved)
# => NoMethodError: undefined method `find_all' for #<Class:0x19a0fb4>
```

This problem could be sidestepped using the `to_a` method:

```
Topic.base.to_a.find_all(&:approved)
# => [#<Reply:0x179e720>#<Topic:0x179e388>#<Reply:0x179e20c>]
```

In this version of Ruby on Rails, this has already been resolved.

Partial updates did not update the lock_version if nothing was altered

When you used optimistic locking with partial updates, unecessary extra queries used to be executed. This has been fixed.

CORRECTIONS IN THE TIME#END_OF_QUARTER AND DATE#END_OF_QUARTER

Rails 2.1 had hardly been released when a serious error was found. If you still have a project that was created by this version, go into **irb** and try running this:

```
Date.new(2008, 5, 31).end_of_quarter
```

ERROR!

Why? The implementation of `end_of_quarter` was faulty. It advances to the end of the last month of the trimester and then grabs the last day. The problem is that it only advances the month, and since this example starts with the 31st of

May, it tries to create an instance of `Date` for June 31st, which doesn't exist. In the case of the `Time` object, an exception is not raised, but it returns the wrong date: July 31st.

This error has been corrected in this version, but if you are still using Rails 2.1 on some project, be careful, because this error only shows up when you use the `end_of_quarter` method on the last day of May, July or August.

CORRECTION TO THE DB:MIGRATE:DOWN AND :UP TASKS

In previous versions of Rails, when you ran `rake db:migrate:down VERSION=some_version_number`, the records in the **schema_migrations** table were not updated.

This meant that after you ran `rake db:migrate:down` or `up`, if you ran `rake db:migrate`, some **migrations** might not be executed. Allow me to demonstrate the problem:

```
$ ./script/generate migration test_migration
      create  db/migrate
      create  db/migrate/20080608082216_test_migration.rb

$ rake db:migrate
(in /Users/erik/projects/railstest)
== 20080608082216 TestMigration: migrating ====================================
-- create_table("foo")
   -> 0.0137s
== 20080608082216 TestMigration: migrated (0.0140s) ===========================

$ rake db:migrate:down VERSION=20080608082216
(in /Users/erik/projects/railstest)
== 20080608082216 TestMigration: reverting ====================================
-- drop_table("foo")
   -> 0.0139s
```

```
== 20080608082216 TestMigration: reverted (0.0144s) ============================

$ rake db:migrate
(in /Users/erik/projects/railstest)

$
```

This problem has been corrected by making sure that the **schema_migrations** table gets updated after running these tasks.

POSTGRESQL

In PostgreSQL, the **typecast** syntax looks like this: `#bodylt;column>::#bodylt;type>`

`ActiveRecord` interpreted this syntax as a named bind, and complained that the value wasn't being passed in the `Hash`. This problem has now been corrected, allowing you to do something like this:

```
:conditions => [':foo::integer', { :foo => 1 }]
```

BUG FIX IN RENAME_COLUMN

This modification is a bug fix in the `rename_column` method. To understand what the problem is, let's look at an example scenario. First, let's create a **migration**:

```
create_table "users", :force => true do |t|
  t.column :name, :string, :default => ''
end
```

Then we create a second **migration** that renames the **name** column:

```
rename_column :users, :name, :first_name
```

If you run this test on your machine, you will notice that when the `rename_column` method executes, the "new" column `first_name` no longer has the default value defined in the first **migration**, like it should.

This bug has been fixed in this version of Rails.

ACTIVERECORD#COUNT METHOD DOES NOT INCLUDE AN ALIAS FOR ASSOCIATIONS

Let's say you have the following association `has_many :through`:

```
class Author < ActiveRecord::Base
  has_many :authorships
  has_many :books, :through => :authorships
end
```

When you search for a book, you can include author information:

```
author.books.find(:all, :include => :authorships,
                  :conditions => ["authorships.primary = ?", true])
```

This works well, without any problems. But try to do the same thing with the `count` method:

```
author.books.count(:include => :authorships,
                   :conditions => ["authorships.primary = ?", true])
```

We have an error. This happens because the **authorships** table was included twice in the same query.

The `find` method is more clever, because it creates an alias for the table, which the `count` method doesn't. I know that the example shown is not the best, but I use it just to demonstrate the problem with the `count` method.

This bug has been fixed. Now the `count` method behaves exactly like the `find` method as far as `:include` is concerned.

BUG FIX IN ACTIVERECORD#COUNT

There was a bug in `ActiveRecord#count` that happened when you used `has_many` with the `:limit` or `:offset` options. Let's look at an example:

```
class Post < ActiveRecord::Base
  has_many :comments, :limit=> 2
end
```

In the code above, when you try to retrieve the comments for a post, only two comments should be returned.

```
post.comments.length # => 2

# Veja o SQL usado:
# SELECT * AS count_all FROM "comments" WHERE
# ("comments".post_id = 1) LIMIT 2

But, when we use the count method:

post.comments.count # => 3

# Veja o SQL usado:
```

```
# SELECT count(*) AS count_all FROM "comments" WHERE
# ("comments".post_id = 1)
```

As you can see, the error occured because the `LIMIT 2` clause was not included in the SQL query.

Of course, this has already been corrected and is now working in Rails 2.2.

BUG FIX IN SUBMIT_TAG

When you used the `submit_tag` method with the `:disable_with` option enabled, it suppressed the `:commit` parameter when the form was sent to the server. This happened because, after submitting a form, the javascript in the **onclick** event first disabled the button and only afterwards sent the form to the server, and disabled fields aren't sent in an HTML form submit.

This represents a problem for the cases where the form has more than one `submit_tag` and its update/creation logic depends on the value in the `:commit` parameter.

This problem has been corrected by including some code in the beginning of the javascript that copies the value of this parameter to a **hidden** form field and sends it to the server, even if the button is disabled.

BUG WHEN TESTING NAMED ROUTES

There is a specific bug in versions before 2.2 that happens when a functional test tests a named route with parameters before executing a request. To understand what I'm talking about, check out this example:

```
def test_something
  post_url(:id => 1) # Antes do request isto retornará um erro
```

```
    get :edit, :id => 1
    post_url(:id => 1) # Aqui funciona
  end
```

This problem has been corrected in Rails 2.2.

TIME#ADVANCE NOW RECOGNIZES PARTIAL DAYS AND WEEKS

After Rails 2.1 was released, the `Time#advance` method stopped working correctly with partial time periods. For example:

```
>> Time.stubs(:now).returns Time.local(2000)

>> 1.5.days.ago == 36.hours.ago
# => false
```

This bug has been fixed in Rails 2.2.

ERROR CREATING TWO CONTROLLERS WITH THE SAME NAME

In Rails 2.1, in some instances, when you created two controllers with the same name in different namespaces, you got an error, like this:

```
$ rails -v
Rails 2.1.0

$ ruby -v
ruby 1.8.6 (2008-03-03 patchlevel 114) [universal-darwin9.0]
```

```
$ rails test
$ cd test/
$ script/generate scaffold Posts title:string body:text
$ script/generate controller -t Admin::Posts
The name 'Admin::PostsHelper' is either already used in your application or reserved by Ruby on Rails.
  Please choose an alternative and run this generator again.
  ...
```

Yet another bug that has been fixed in this version.

Chapter 10

CHANGELOG

ACTIONMAILER

2.2.0 [RC1] (October 24th, 2008)

- Add layout functionality to mailers [Pratik]

Mailer layouts behaves just like controller layouts, except layout names need to have '_mailer' postfix for them to be automatically picked up.

ACTIONPACK

2.2.0 [RC1] (October 24th, 2008)

- Fix incorrect closing CDATA delimiter and that HTML::Node.parse would blow up on unclosed CDATA sections [packagethief]

- Added stale? and fresh_when methods to provide a layer of abstraction above request.fresh? and friends [DHH]. Example:

  class ArticlesController < ApplicationController def show_with_respond_to_block

  ```
    @article = Article.find(params[:id])

    if stale?(:last_modified => @article.published_at.utc, :etag => @article)
      respond_to do |wants|
        # normal response processing
      end
    end
  ```

  end

  def show_with_implied_render

  ```
    @article = Article.find(params[:id])

    fresh_when(:last_modified => @article.published_at.utc, :etag => @article)
  ```

  end end

- Added inline builder yield to atom_feed_helper tags where appropriate [Sam Ruby]. Example:

  entry.summary :type => 'xhtml' do |xhtml| xhtml.p pluralize(order.line*items.count, "line item") xhtml.p "Shipped to #{order.address}" xhtml.p "Paid by #{order.pay*type}" end

- Make PrototypeHelper#submit_to_remote a wrapper around PrototypeHelper#button_to_remote. [Tarmo Tänav]
- Set HttpOnly for the cookie session store's cookie. #1046
- Added FormTagHelper#image_submit_tag confirm option #784 [Alastair Brunton]
- Fixed FormTagHelper#submit_tag with :disable_with option wouldn't submit the button's value when was clicked #633 [Jose Fernandez]
- Stopped logging template compiles as it only clogs up the log [DHH]
- Changed the X-Runtime header to report in milliseconds [DHH]
- Changed BenchmarkHelper#benchmark to report in milliseconds [DHH]
- Changed logging format to be millisecond based and skip misleading stats [DHH]. Went from:

  Completed in 0.10000 (4 reqs/sec) | Rendering: 0.04000 (40%) | DB: 0.00400 (4%) | 200 OK [http://example.com]

...to:

```
Completed in 100ms (View: 40, DB: 4) | 200 OK [http://example.com]
```

- Add support for shallow nesting of routes. #838 [S. Brent Faulkner]

Example :

map.resources :users, :shallow => true do |user|

```
  user.resources :posts
```

end

- GET /users/1/posts (maps to PostsController#index action as usual) named route "user_posts" is added as usual.
- GET /posts/2 (maps to PostsController#show action as if it were not nested) Additionally, named route "post" is added too.
- Added button_to_remote helper. #3641 [Donald Piret, Tarmo Tänav]
- Deprecate render_component. Please use render_component plugin from http://github.com/rails/render_component/tree/master [Pratik]
- Routes may be restricted to lists of HTTP methods instead of a single method or :any. #407 [Brennan Dunn, Gaius Centus Novus] map.resource :posts, :collection => { :search => [:get, :post] } map.session 'session', :requirements => { :method => [:get, :post, :delete] }
- Deprecated implicit local assignments when rendering partials [Josh Peek]
- Introduce current_cycle helper method to return the current value without bumping the cycle. #417 [Ken Collins]
- Allow polymorphic_url helper to take url options. #880 [Tarmo Tänav]
- Switched integration test runner to use Rack processor instead of CGI [Josh Peek]

- Made AbstractRequest.if_modified_sense return nil if the header could not be parsed [Jamis Buck]
- Added back ActionController::Base.allow_concurrency flag [Josh Peek]
- AbstractRequest.relative_url_root is no longer automatically configured by a HTTP header. It can now be set in your configuration environment with config.action_controller.relative_url_root [Josh Peek]
- Update Prototype to 1.6.0.2 #599 [Patrick Joyce]
- Conditional GET utility methods. [Jeremy Kemper] response.last_modified = @post.updated_at response.etag = [:admin, @post, current_user]

  if request.fresh?(response) head :not_modified else # render ... end
- All 2xx requests are considered successful [Josh Peek]
- Fixed that AssetTagHelper#compute_public_path shouldn't cache the asset_host along with the source or per-request proc's won't run [DHH]
- Removed config.action_view.cache_template_loading, use config.cache_classes instead [Josh Peek]
- Get buffer for fragment cache from template's @output_buffer [Josh Peek]
- Set config.action_view.warn_cache_misses = true to receive a warning if you perform an action that results in an expensive disk operation that could be cached [Josh Peek]
- Refactor template preloading. New abstractions include Renderable mixins and a refactored Template class [Josh Peek]

- Changed ActionView::TemplateHandler#render API method signature to render(template, local_assigns = {}) [Josh Peek]

- Changed PrototypeHelper#submit_to_remote to PrototypeHelper#button_to_remote to stay consistent with link_to_remote (submit_to_remote still works as an alias) #8994 [clemens]

- Add :recursive option to javascript_include_tag and stylesheet_link_tag to be used along with :all. #480 [Damian Janowski]

- Allow users to disable the use of the Accept header [Michael Koziarski]

  The accept header is poorly implemented by browsers and causes strange errors when used on public sites where crawlers make requests too. You can use formatted urls (e.g. /people/1.xml) to support API clients in a much simpler way.

To disable the header you need to set:

config.action_controller.use_accept_header = false

- Do not stat template files in production mode before rendering. You will no longer be able to modify templates in production mode without restarting the server [Josh Peek]

- Deprecated TemplateHandler line offset [Josh Peek]

- Allow caches_action to accept cache store options. #416. [José Valim]. Example:

caches_action :index, :redirected, :if => Proc.new { |c| !c.request.format.json? }, :expires_in => 1.hour

- Remove define_javascript_functions, javascript_include_tag and friends are far superior. [Michael Koziarski]

- Deprecate :use_full_path render option. The supplying the option no longer has an effect [Josh Peek]
- Add :as option to render a collection of partials with a custom local variable name. #509 [Simon Jefford, Pratik Naik]

render :partial => 'other_people', :collection => @people, :as => :person

This will let you access objects of @people as 'person' local variable inside 'other_people' partial template.

- time_zone_select: support for regexp matching of priority zones. Resolves #195 [Ernie Miller]
- Made ActionView::Base#render_file private [Josh Peek]
- Refactor and simplify the implementation of assert_redirected_to. Arguments are now normalised relative to the controller being tested, not the root of the application. [Michael Koziarski]

This could cause some erroneous test failures if you were redirecting between controllers in different namespaces and wrote your assertions relative to the root of the application.

- Remove follow_redirect from controller functional tests.

If you want to follow redirects you can use integration tests. The functional test version was only useful if you were using redirect_to :id=>...

- Fix polymorphic_url with singleton resources. #461 [Tammer Saleh]
- Replaced TemplateFinder abstraction with ViewLoadPaths [Josh Peek]
- Added block-call style to link_to [Sam Stephenson/DHH]. Example:

<% link_to(@profile) do %> **<%= @profile.name %>** -- Check it out!! <% end %>

- Performance: integration test benchmarking and profiling. [Jeremy Kemper]
- Make caching more aware of mime types. Ensure request format is not considered while expiring cache. [Jonathan del Strother]
- Drop ActionController::Base.allow_concurrency flag [Josh Peek]
- More efficient concat and capture helpers. Remove ActionView::Base.erb_variable. [Jeremy Kemper]
- Added page.reload functionality. Resolves #277. [Sean Huber]
- Fixed Request#remote_ip to only raise hell if the HTTP_CLIENT_IP and HTTP_X_FORWARDED_FOR doesn't match (not just if they're both present) [Mark Imbriaco, Bradford Folkens]
- Allow caches_action to accept a layout option [José Valim]
- Added Rack processor [Ezra Zygmuntowicz, Josh Peek]

ACTIVERECORD

2.2.0 [RC1] (October 24th, 2008)

- Skip collection ids reader optimization if using :finder_sql [Jeremy Kemper]
- Add Model#delete instance method, similar to Model.delete class method. #1086 [Hongli Lai]

- MySQL: cope with quirky default values for not-null text columns. #1043 [Frederick Cheung]
- Multiparameter attributes skip time zone conversion for time-only columns [#1030 state:resolved] [Geoff Buesing]
- Base.skip_time_zone_conversion_for_attributes uses class_inheritable_accessor, so that subclasses don't overwrite Base [#346 state:resolved] [miloops]
- Added find_last_by dynamic finder #762 [miloops]
- Internal API: configurable association options and build_association method for reflections so plugins may extend and override. #985 [Hongli Lai]
- Changed benchmarks to be reported in milliseconds [DHH]
- Connection pooling. #936 [Nick Sieger]
- Merge scoped :joins together instead of overwriting them. May expose scoping bugs in your code! #501 [Andrew White]
- before_save, before_validation and before_destroy callbacks that return false will now ROLLBACK the transaction. Previously this would have been committed before the processing was aborted. #891 [Xavier Noria]
- Transactional migrations for databases which support them. #834 [divoxx, Adam Wiggins, Tarmo Tänav]
- Set config.active_record.timestamped_migrations = false to have migrations with numeric prefix instead of UTC timestamp. #446. [Andrew Stone, Nik Wakelin]

- change_column_default preserves the not-null constraint. #617 [Tarmo Tänav]
- Fixed that create database statements would always include "DEFAULT NULL" (Nick Sieger) [#334]
- Add :tokenizer option to validates_length_of to specify how to split up the attribute string. #507. [David Lowenfels] Example :

Ensure essay contains at least 100 words. validates_length_of :essay, :minimum => 100, :too_short => "Your essay must be at least %d words."), :tokenizer => lambda {|str| str.scan(/\w+/) }

- Allow conditions on multiple tables to be specified using hash. [Pratik Naik]. Example:

User.all :joins => :items, :conditions => { :age => 10, :items => { :color => 'black' } } Item.first :conditions => { :items => { :color => 'red' } }

- Always treat integer :limit as byte length. #420 [Tarmo Tänav]
- Partial updates don't update lock_version if nothing changed. #426 [Daniel Morrison]
- Fix column collision with named_scope and :joins. #46 [Duncan Beevers, Mark Catley]
- db:migrate:down and :up update schema_migrations. #369 [Michael Raidel, RaceCondition]
- PostgreSQL: support :conditions => [':foo::integer', { :foo => 1 }] without treating the ::integer typecast as a bind variable. [Tarmo Tänav]
- MySQL: rename_column preserves column defaults. #466 [Diego Algorta]
- Add :from option to calculations. #397 [Ben Munat]

- Add :validate option to associations to enable/disable the automatic validation of associated models. Resolves #301. [Jan De Poorter]
- PostgreSQL: use 'INSERT ... RETURNING id' for 8.2 and later. [Jeremy Kemper]
- Added SQL escaping for :limit and :offset in MySQL [Jonathan Wiess]

ACTIVERESOURCE

2.2.0 [RC1] (October 24th, 2008)

- Add ActiveResource::Base#to_xml and ActiveResource::Base#to_json. #1011 [Rasik Pandey, Cody Fauser]
- Add ActiveResource::Base.find(:last). [#754 state:resolved] (Adrian Mugnolo)
- Fixed problems with the logger used if the logging string included %'s [#840 state:resolved] (Jamis Buck)
- Fixed Base#exists? to check status code as integer [#299 state:resolved] (Wes Oldenbeuving)

ACTIVESUPPORT

2.2.0 [RC1] (October 24th, 2008)

- TimeWithZone#freeze: preload instance variables so that we can actually freeze [Geoff Buesing]
- Fix Brasilia timezone #1180 [Marcus Derencius, Kane]

- Time#advance recognizes fractional days and weeks. Deprecate Durations of fractional months and years #970 [Tom Lea]

- Add ActiveSupport::Rescuable module abstracting ActionController::Base rescue_from features. [Norbert Crombach, Pratik]

- Switch from String#chars to String#mb_chars for the unicode proxy. [Manfred Stienstra]

This helps with 1.8.7 compatibility and also improves performance for some operations by reducing indirection.

- TimeWithZone #wday, #yday and #to_date avoid trip through #method_missing [Geoff Buesing]

- Added Time, Date, DateTime and TimeWithZone #past?, #future? and #today? #720 [Clemens Kofler, Geoff Buesing]

- Fixed Sri Jayawardenepura time zone to map to Asia/Colombo [Jamis Buck]

- Added Inflector#parameterize for easy slug generation ("Donald E. Knuth".parameterize => "donald-e-knuth") #713 [Matt Darby]

- Changed cache benchmarking to be reported in milliseconds [DHH]

- Fix Ruby's Time marshaling bug in pre-1.9 versions of Ruby: utc instances are now correctly unmarshaled with a utc zone instead of the system local zone [#900 state:resolved] [Luca Guidi, Geoff Buesing]

- Add Array#in_groups which splits or iterates over the array in specified number of groups. #579. [Adrian Mugnolo] Example:

a = (1..10).to*a a.in*groups(3) #=> [[1, 2, 3, 4], [5, 6, 7, nil], [8, 9, 10, nil]] a.in_groups(3, false) #=> [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]

- Fix TimeWithZone unmarshaling: coerce unmarshaled Time instances to utc, because Ruby's marshaling of Time instances doesn't respect the zone [Geoff Buesing]
- Added Memoizable mixin for caching simple lazy loaded attributes [Josh Peek]
- Move the test related core_ext stuff out of core_ext so it's only loaded by the test helpers. [Michael Koziarski]
- Add Inflection rules for String#humanize. #535 [dcmanges]

ActiveSupport::Inflector.inflections do |inflect|

```
  inflect.human(/_cnt$/i, '_count')
```

end

'jargon_cnt'.humanize # => 'Jargon count'

- TimeWithZone: when crossing DST boundary, treat Durations of days, months or years as variable-length, and all other values as absolute length. A time + 24.hours will advance exactly 24 hours, but a time + 1.day will advance 23-25 hours, depending on the day. Ensure consistent behavior across all advancing methods [Geoff Buesing]
- Added TimeZone #=~, to support matching zones by regex in time_zone_select. #195 [Ernie Miller]
- Added Array#second through Array#tenth as aliases for Array#[1] through Array#[9] [DHH]

- Added test/do declaration style testing to ActiveSupport::TestCase [DHH via Jay Fields]
- Added Object#present? which is equivalent to !Object#blank? [DHH]
- Added Enumberable#many? to encapsulate collection.size > 1 [DHH/Damian Janowski]
- Add more standard Hash methods to ActiveSupport::OrderedHash [Steve Purcell]
- Namespace Inflector, Dependencies, OrderedOptions, and TimeZone under ActiveSupport [Josh Peek]
- Added StringInquirer for doing things like StringInquirer.new("production").production? # => true and StringInquirer.new("production").development? # => false [DHH]
- Fixed Date#end_of_quarter to not blow up on May 31st #289 state:resolved

RAILTIES

2.2.0 [RC1] (October 24th, 2008)

- Fixed that sqlite would report "db/development.sqlite3 already exists" whether true or not on db:create #614 [Antonio Cangiano]
- Added config.threadsafe! to toggle allow concurrency settings and disable the dependency loader [Josh Peek]
- Turn cache_classes on by default [Josh Peek]
- Added configurable eager load paths. Defaults to app/models, app/controllers, and app/helpers [Josh Peek]

- Introduce simple internationalization support. [Ruby i18n team]
- Make script/plugin install -r option work with git based plugins. #257. [Tim Pope Jakub Kuźma]. Example:

script/plugin install git://github.com/mislav/will_paginate.git -r agnostic # Installs 'agnostic' branch script/plugin install git://github.com/dchelimsky/rspec.git -r 'tag 1.1.4'

- Added Rails.initialized? flag [Josh Peek]
- Make rake test:uncommitted work with Git. [Tim Pope]
- Added Thin support to script/server. #488 [Bob Klosinski]
- Fix script/about in production mode. #370 [Cheah Chu Yeow, Xavier Noria, David Krmpotic]
- Add the gem load paths before the framework is loaded, so certain gems like RedCloth and BlueCloth can be frozen.
- Fix discrepancies with loading rails/init.rb from gems.
- Plugins check for the gem init path (rails/init.rb) before the standard plugin init path (init.rb) [Jacek Becela]
- Changed all generated tests to use the test/do declaration style [DHH]
- Wrapped Rails.env in StringInquirer so you can do Rails.env.development? [DHH]
- Fixed that RailsInfoController wasn't considering all requests local in development mode (Edgard Castro) [#310 state:resolved]