Ruby on Rails 2.2

What's new?

di Carlos Brando
tradotto da Carlo Pecchia

published on envycasts

Ruby on Rails 2.2

WHAT'S NEW

Prima edizione

Ruby on Rails 2.2

WHAT'S NEW

Prima edizione

di Carlos Brando
traduzione italiana a cura di Carlo Pecchia

Prima edizione: ottobre 2008

Carlos Brando
http://www.nomedojogo.com

Carlo Pecchia
http://carlopecchia.eu

INTRODUZIONE

Ruby on Rails 2.2 è ricco di nuove funzionalità, miglioramenti e correzioni di bug. In questo libro troverete una breve descrizione accompagnata (nella maggior parte dei casi) da esempi per ciascuno dei cambiamenti più significativi introdotti nella nuova versione di Rails.

La realizzazione del presente libro è stata una notevole impresa, ed è nostra speranza che possa aiutarvi a comprendere ogni nuova funzionalità introdotta in Rails.

Con la nuova versione Rails è divenuto poliglotta. Con il nuovo sistema di internazionalizzazione (i18n) potrete creare con facilità applicazioni per i vari utenti sparsi sul globo.

Sforzi notevoli sono stati fatti per rendere Rails thread-safe ed il più possibile pronto per l'imminente rilascio della versione 1.9 di Ruby. C'è stato anche un grande lavoro per renderlo maggiormente compatibile con JRuby. Benché la modalità thread-safe non è ancora disponibile per tutti gli ambienti, dal momento che funziona solo nelle virtual machine con supporto nativo per i thread (es: JRuby), è da considerare come un notevole punto di vantaggio per il framework.

Se nel passato ci sono state lamentele riguardo la mancanza di documentazione in Rails, con questa nuova versione queste svaniscono. E' stato fatto un grande lavoro documentando il codice ed illustrando le funzionalità di Rails. Per un chiaro esempio, provate questo comando nel vostro terminale:

rake doc:guides

Questo task rake creerà la directory doc/guides, nella directory radice del vostro progetto, popolandola con varie guide che vi saranno utili per studiare Rails.

Capitolo 1

ActiveRecord

NUOVE OPZIONI PER LE ASSOCIAZIONI: :VALIDATE

In Rails è stata aggiunta una nuova opzione per le associazioni. L'opzione `:validate` viene utilizzata per abilitare/disabilitare sugli oggetti associati al modello. Esempio:

```
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
```

Come potete vedere nell'esempio, benché l'oggetto associato (`Developer`) non sia valido, l'oggetto principale (`AuditLog`) viene salvato nel database. Questo non costituiva il normale comportamento nelle precedenti versioni di Rails, dove un oggetto padre veniva salvato solo se tutti i suoi oggetti figli erano validi.

Sebbene nel precedente esempio abbiamo disabilitato la validazione per dimostrare il funzionamento della nuova feature, in Rails 2.2 il valore di default per `:validate` è impostato a `false`. Quindi `:validate => false` non sarebbe stato necessario nel codice precedente. D'altra parte, se volete avere il vecchio comportamento, è necessario specificare `:validate => true`.

NUOVI METODI PER INDICARE CONDITIONS CON UN HASH

Nell'utilizzare le query sul database, spesso vi potreste trovare nella condizione di usare l'opzione `:join`, sia per migliorare le performance dell'applicazione, sia per necessità di recuperare informazioni dipendenti da risultati sparsi su più tabelle.

Per esempio, se voleste recuperare tutti gli utenti che hanno acquistato articoli rossi, dovreste utilizzare una query del genere:

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

Questa sintassi è poco gradevole, dal momento che dovete includere il nome di una tabella (**items** nel nostro caso) all'interno di una stringa.

In Rails 2.2 esiste un nuovo metodo per fare ciò, ovvero utilizzando una chiave di un hash per identificare la tabella:

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

# un ulteriore esempio per rendere il codice più comprensibile
User.all :joins => :items, :conditions => {
  :users => { :age => 10 },
  :items => { :color => 'red' }
}
```

E' nostra opinione che in questo modo il codice appaia molto più leggibile, specialmente quando avete bisogno di indicare condizioni su più campi di tabelle differenti.

Non dimenticate che la chiave utilizzata è il nome plurale della tabella (o un alias, se lo avete utilizzato nella query).

NUOVA OPZIONE :FROM PER I METODI DI CALCOLO IN ACTIVERECORD

Una nuova opzione è stata aggiunta nei metodi di calcolo di `ActiveRecord` (`count`, `sum`, `average`, `minimum` e `maximum`).

Quando utilizzate l'opzione `:from` potete fare l'override del nome della tabella generata dalla query `ActiveRecord`, il che non sembra molto utile ad un primo sguardo, ma la cosa interessante è che vi permette di forzare MySQL ad utilizzare uno specifico indice per il calcolo.

Vediamo qualche esempio:

```
# Forza MySQL ad utilizzare un index per un calcolo
Edge.count :all, :from => 'edges USE INDEX(unique_edge_index)',
           :conditions => 'sink_id < 5')

# Effettua il calcolo in una tabella della classe associata
Company.count :all, :from => 'accounts'
```

IL METODO MERGE_CONDITIONS IN ACTIVERECORD È ORA PUBBLICO

Il metodo `merge_conditions` in `ActiveRecord` è ora un metodo pubblico, questo significa che è disponibile in tutti i vostri modelli.

Questo metodo fa esattamente quello che dice — vi permette di specificare più `conditions` che vengono combinate in una singola condizione.

```
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')"
```

Notate che `conditions` effettua l'unione sempre con un **AND**.

DEFINIRE IL COMPORTAMENTO DI VALIDATES_LENGTH_OF

Uno dei tanti metodi di validazione presenti in `ActiveRecord` è `validates_length_of`. Questo particolare metodo ci permette di assicurarci che il valore immagazzinato in una particolare colonna del database abbia una certa lunghezza. Ci permette di indicare una lunghezza massima, una lunghezza minima, una lunghezza esatta, oppure un insieme di lunghezze valide.

"Lunghezza", comunque, è un concetto relativo. Dicendo "lunghezza" (length) ci riferiamo, generalmente, al numero di caratteri presenti in un testo. Ma immaginiamo di avere un campo di un form in cui il limite non è definito dal numero di caratteri, ma dal numero di parole - ad esempio "Scrivi un paragrafo con almeno 100 parole" (un esempio migliore potrebbe essere "Scrivi con commento con non più di 100 parole"). Immaginate una pagina dove un utente deve inserire un breve tema, per esempio.

Prima di Rails 2.2 la nostra unica possibilità era quella di creare un metodo di validazione apposito, ma adesso possiamo personalizzare `validates_length_of` usando l'opzione `:tokenizer`. Il seguente esempio risolve il problema suesposto:

```
validates_length_of :essay,
                    :minimum => 100,
                    :too_short => "Il suo tema deve contenere almeno 100 parole.",
                    :tokenizer => lambda {|str| str.scan(/\w+/) }
```

Questo è solo un esempio di cosa potete fare con questa nuova opzione. Oltre a ciò, potete utilizzarla per contare solamente il numero di cifre inserite, il numero di volte che una certa parola viene ripetuta, etc.

OPZIONE :LIMIT COME DIMENSIONE IN BYTE SULLA COLONNA

A partire da questa versione di Rails l'opzione `:limit` indica la dimesione in byte per colonne di tipo intero, per MySQL e PostgreSQL (su Sqlite già funzionava così).

Il tipo di colonna utilizzata nel database dipende dal numero di byte indicati. Date un'occhiata al seguente codice che determina il tipo di colonna 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}")
```

E per PostgreSQL:

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

INDICARE DIFFERENTI PRIMARY_KEY NELLE ASSOCIAZIONI

Una nuova opzione, chiamata `:primary_key`, è stata agiunta ai metodi `has_many` e `has_one`.

Utilizzando questa opzione potete definire quale metodo e quale modello associato restituisce la chiave primaria che sarà utilizzata nell'associazione. Ovviamente la chiave primaria standard rimane `id`.

Osservate il seguente esempio:

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

Il metodo `has_one` si comporta esattamente allo stesso modo.

NUOVO TEST HELPER (ASSERT_SQL)

Probabilmente avrete già sentito parlare del metodo `assert_queries`, utilizzato per validare test e il numero di query eseguite su un database. Per esempio:

Nel test seguente specifichiamo che, se ci sono `partial_updates`, una query deve essere eseguita sul database; altrimenti nessuna query deve essere eseguita.

Adesso abbiamo un ulteriore helper che facilita il test delle query generate dalle astrazioni di `ActiveRecord`. Vediamo un esempio:

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

Nell'esempio precedente stiamo asserendo che, nel blocco specificato, almeno una query dovrebbe contenere la parola **COUNT**. Ovviamente potete essere più specifici nell'espressione regolare utilizzata. Vediamo un altro esempio:

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

MIGRATIONPROXY

Supponiamo che dopo avere eseguito un certo numero di migrazioni, un determinato modello venga rinominato. Ora supponiamo che, prima che ciò avvenga, un'altra migrazione faccia riferimento a tale modello. Poiché tutte le migrazioni

vengono caricate indipendentemente dal fatto che vengano eseguite o meno, questa situazione causerebbe un errore e bloccherebbe l'esecuzione delle migrazioni stesse.

Per arginare tale problema è stata introdotta una nuova classe chiamata `MigrationProxy`, la quale conserva il nome, la versione ed il nome del file di ciascuna migrazione. Quindi utilizza questa informazione per caricare le migrazioni solamente quando necessario, in tal modo non vengono caricate tutte insieme.

POOL DI CONNESSIONE IN RAILS

Uno dei problemi riscontrati da un certo numero di sviluppatori su Rails è relativo alla sua lentezza. Sappiamo bene che non è oggettivamente vero, ma sappiamo anche che molto potrebbe essere fatto per migliorare le performance di Rails.

E qualcosa è stato fatto. Un **pool di connessione** al database è stato aggiunto a Rails.

Ogni volta che viene effettuata una query al database viene sprecato del tempo per aprire una nuova connessione prima che i dati possano essere letti/salvati. A prima vista ciò potrebbe sembrare banale, ma aprire una connessione verso il database non è semplice. Infatti è necessario aprire una connessione di rete con il database server, autenticarsi, e effettuare un certo numero di verifiche. Tutto ciò richiede tempo e risorse. Dopo aver create tale connessione, Rails la utilizza per ogni richiesta effettuata verso il database, e query pesanti possono rallentare l'esecuzione di altre query. Ciò dimostra come il database possa spesso diventare il tallone di Achille per alcune progetti Rails di grandi dimensioni (o con elevati carichi, NdT).

La soluzione a tale problema consiste nel riutilizzare le connessioni precedentemente create, facendogli eseguire più task prima di essere chiuse. Ovvero, in termini più tecnici, necessitiamo di un connection pool.

In pratica funziona così: una connessione al database viene creata in modo tale da poter eseguire una query. Quindi, anziché chiuderla, viene salvata in un connection pool. Quando deve essere eseguita un'altra query il sistema recupera la connessione precedentemente salvata, riducendo in questo modo il tempo e le risorse richieste per terminare il task in esecuzione. Diverse connessioni possono essere salvate nel pool contemporaneamente, e le richieste vengono distribuite tra di esse. Questo significa che una query lenta può tranquillamente essere in esecuzione sul database mentre l'applicazione continua a ricevere richieste e ad eseguire query utilizzando le altre connessioni presenti nel pool.

In Rails, un nuovo pool viene creato ogni volta che il metodo `establish_connection` viene invocato. In altri termini, ogni database presente in **database.yml** avrà i propri connection pool.

Il pool, inizialmente vuoto, cresce finché non raggiunge il limite di default di cinque connessioni, ma è possibile aumentare tale valore aggiungendo l'opzione `pool` nella configurazione del database:

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

Se non è disponibile nessuna connessione, il thread attenderà una connessione libera per cinque secondi (valore di default) prima di andare in time out. Anche tale periodo di attesa può essere configurato aggiungendo l'opzione `wait_timeout` alla configurazione del database.

Se volete utilizzare i connection pool fuori da `ActionPack`, esiste un altro metodo chiamato `ActiveRecord::Base.connection_pool`, il quale vi permette di gestire manualmente la prenotazione (checking out) ed

il rilascio (cheching in) delle connessioni presenti nel pool. Non dimenticate di rilasciare una connessione una volta terminato il suo uso, affinché possa essere riutilizzata da altre richieste.

```
connection = ActiveRecord::Base.connection_pool.checkout
# eseguire query sul database
ActiveRecord::Base.connection_pool.checkin(connection)
```

Potete anche utilizzare il metodo `ActiveRecord::Base.connection_pool.with_connection`, il quale esegue il checkout/checkin per voi, rendendo il codice leggermente più sicuro.

```
ActiveRecord::Base.connection_pool.with_connection do |connection|
  # eseguire query sul database
end
```

MIGRAZIONI CON TRANSAZIONI IN POSTGRESQL

Quando una migrazione viene eseguita e si manifesta un errore, solamente il codice prima dell'errore viene eseguito sul database. Dopo ciò, la migrazione viene considerata terminata. Risolvere tali errori può causare un autentico mal di pancia.

Ma se il database (DBMS, NdT) che state utilizzando supporta i **rollbacks DDL** nelle transazioni, allora è possibile sfruttare questa feature per "disfare" tutti i cambiamenti fatti prima dell'errore. Il problema è che non tutti i motori di database supportano questa feature. MySQL, ad esempio, non la supporta. Ma PostgreSQL, SQL Server ed alcuni altri sì.

Rails è stato aggiornato per permettere l'uso delle transazioni nelle migrazioni, per i database che le supportano. Sebbene Rails suppporti tale feature, il database adapter deve essere aggiornato (semplicemente facendo in modo che il

metodo `supports_ddl_transactions?` restituisca `true`) per poterla utilizzare. Nel momento in cui tale libro veniva pubblicato, solamente l'adapter per PostgreSQL supportava tale feature.

NUOVA VERSIONE DISTRUTTIVA DEI METODI FIND DI ACTIVERECORD

I finder dinamici di `ActiveRecord` adesso hanno una controparte "distruttiva" che solleva un eccezione `RecordNotFound` quando l'insieme dei risultati è vuoto, anziché restituire `nil` come nel comportamento originale.

Per utilizzare tali versioni distruttive è sufficiente aggiungere un punto esclamativo alla fine del nome del metodo. Ecco un esempio:

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

MIGLIORARE LE PERFORMANCE NEI METODI ASSOCIATION_ID

Se avete due modelli, `Post` e `Comment`, dove un post ha più commenti (`has_many`), se eseguite il codice:

```
Post.first.comment_ids
```

Rails utilizzerà la seguente query per recuperare gli id:

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

Ma in questo caso, non siamo interessati agli oggetti completi. La seguente query sarebbe più che sufficiente per realizzare il precedente metodo, e sarebbe anche più veloce:

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

Per entrambe le associazioni `has_many` e `has_many :through`, il codice base di Rails è stato aggiornato per includere da questa versione in poi tale miglioramento delle performance.

UN ULTERIORE FINDER DINAMICO

In aggiunta agli esistenti finder di `ActiveRecord`, adesso abbiamo anche `find_last_by`. Gli altri sono `find_by` e `find_all_by`.

Oltre ad essere semplice, ciò offre un modo molto più elegante per recuperare — ad esempio — l'ultimo commento fatto da un determinato utente. Esempio:

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

    # è equivalente a

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

SUPPORTO PER L'OPZIONE :LIMIT IN UPDATE_ALL

Il metodo `update_all` adesso dispone di una nuova opzione `:limit`. E' molto utile, poiché ci assicura che tutto funzioni correttamente quando utilizziamo `update_all` con l'opzione `:limit` in un'associazione `has_many`.

NUOVE OPZIONI PER IL METODO COMPOSED_OF

Il metodo `composed_of` dispone di due nuove opzioni: `:constructor` e `:converter`.

L'opzione `:constructor` può ricevere un simbolo con il nome di un metodo, oppure un `Proc`. Per default, la classe composta viene creata dal metodo `new`, ricevendo tutti gli attributi mappati come parametri, esattamente con l'ordine con cui vengono mappati. Ma, se per qualsiasi motivo, la vostra classe non seguisse tale convenzione, dovreste usare l'opzione `:constructor`. Questa permette di modificare la modalità con cui la classe viene creata. Osserviamo il seguente esempio, tratto dalla documentazione di Rails:

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

Come potete vedere, prima di creare una nuova istanza della classe `IPAddr`, è necessario fornire un parametro aggiuntivo al costruttore. Utilizzando l'opzione `:constructor` ciò risulta semplice.

Discorso analogo per l'opzione `:converter`, anch'essa accetta un simbolo indicante un metodo nella classe `:class_name`; oppure un `Proc`, e questo viene chiamato quando il valore del campo composto viene assegnato da un'altra istanza e pertanto è richiesta una conversione. Ecco un esempio:

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

In questo esempio, il metodo `balance=` si aspetterà sempre un'istanza della classe `Money`, ma, se viene passato un altro tipo di oggetto, questo verrà convertito utilizzando il metodo `parse` dell'oggetto `Money`.

Dovreste utilizzare la nuova opzione `:converter` al posto del blocco di conversione previsto dal metodo.

AGGIORNARE UN'ASSOCIAZIONE UTILIZZANDO LE SUE FOREIGN KEY

Non siamo sicuri che questo sia un bug ma, a nostro parere, ha causato dei problemi. Osservate il seguente codice, dove cerco di modificare un account utente utilizzando le sue foreign key in un progetto su Rails 2.1 (o precedente):

```
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">
```

Notate come stiamo modificando l'account dell'utente (user), ma l'associazione non viene alterata. Anche dopo aver salvato l'oggetto `user`, se non viene ricaricato, l'associazione continuerà a mostrare l'account sbagliato.

Questo problema è stato risolto in Rails 2.2. Osserviamo:

```
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>
```

Notate che modificando il `post` utilizzando le sue foreign key, l'associazione viene aggiornata automaticamente.

ORA ALIAS_ATTRIBUTE FUNZIONA CON I DIRTY OBJECTS

Per comprendere questa modifica dobbiamo analizzare uno spezzone di codice eseguito su due versioni di Rails: 2.2 e precedente. Vediamo un esempio di modello:

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

Notate che stiamo utilizzando il metodo `alias_attribute` per creare un alias per l'attributo `body` chiamato `text`. In teoria questo metodo dovrebbe replicare i vari metodi dinamici creati da `ActiveRecord`, correlati all'attributo `body`. Ma osservate il seguente esempio eseguito su Rails 2.1 (o precedente):

```
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?' ...
```

Nell'eseguire il metodo `text_changed?` compare un errore, dal momento che la chiamata `alias_attribute` sta replicando il metodo tracciato. Ma ciò adesso è stato corretto.

Osserviamo il comportamento del medesimo codice su Rails 2.2:

```
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"]
```

NUOVO METODO DI ISTANZA MODEL#DELETE

Al fine di rendere `ActiveRecord` più consistente, un ulteriore metodo di istanza è stato creato: `Model#delete`. E' simile all'omonimo metodo di classe. Il metodo `delete`, al differenza del metodo `destroy`, elimina il record dal database senza invocare i metodi di callback, come `before_destroy` e `after_destroy`.

Questo metodo, inoltre, non rispetta le opzioni di vincolo, come `:dependent`, che specifica cosa debba essere fatto con gli oggetti associati quando il record viene eliminato.

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

RENDERE GLI ATTRIBUTI ACTIVERECORD PRIVATI

Rails 2.2 vi permette di marcare attributi di `ActiveRecord` come `private`. Fino ad ora, questo era difficoltoso, dal momento che tali attributi sono creati attraverso tecniche di metaprogrammazione.

Per capire come funziona, rendiamo privato l'attributo `name` del modello `User`:

```
class User < ActiveRecord::Base

  private
  def name
    "I'm private"
  end

end
```

Quindi, proviamo ad accedervi:

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

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

Potete vedere che l'eccezione `NoMethodError` viene sollevata quando cerchiamo di chiamare un metodo privato. Tuttavia, possiamo modificare il nome dell'utente, poiché il metodo `name=` è ancora pubblico:

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

Capitolo 2

ActiveSupport

DA ARRAY#SECOND FINO A ARRAY#TENTH

Nella classe `Array` già sono presenti i metodi `first` e `last`, quindi perché non aggiungere anche `second`, `third`, `fourth` e così via? Questo è proprio ciò che è stato fatto. Questi metodi restituiscono l'elemento dell'array che si trova all'offset specificato.

Esempio:

```
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]
```

NUOVO METODO: ENUMERABLE#MANY?

Un nuovo metodo, chiamato `many?`, è stato aggiunto al modulo `Enumerable`. E come il suo nome suggerisce, vi informa se una collezione contiene più di un oggetto; ovvero, se contiene più oggetti.

Questo metodo è un alias per `collection.size > 1`. Vediamo alcuni esempi:

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

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

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

In aggiunta al formato mostrato nei precedenti esempi, questo metodo beneficia di una nuova implementazione che gli permette di ricevere anche dei blocchi, esibendo quindi un comportamento simile al metodo `any?`.

Osserviamo i seguenti esempi:

```
>> 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

# un altro esempio...
people.many? { |p| p.age > 26 }
```

Per inciso, questo metodo restituisce `true` solo se più di una iterazione del blocco restituisce true, oppure laddove la collezione ha più di un oggetto quando invocato senza blocco.

Un'interessante nota a margine: il metodo fu inizialmente chiamato `several?` ma poi venne cambiato in `many?`.

DESCRIVERE REGOLE PER STRING#HUMANIZE

Per diverso tempo Pratik Naik ha chiesto che tale patch fosse accettata in Rails, e sembra proprio che alla fine ci sia riuscito.

In **config/initializers/inflections.rb** possiamo aggiungere nuove inflessioni per la pluralizzazione, singolarizzazione, et similia:

```
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 possiamo anche aggiungere inflessioni per il metodo `humanize` della classe `String`. Vediamo qualche interessante esempio:

```
'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"
```

AGGIUNTA DI MEMOIZABLE PER IL CACHING DEGLI ATTRIBUTI

Le performance sono importanti, e uno dei metodi più usati per migliorare la velocità di esecuzione del codice è il caching. Avrete probabilmente scritto codice del genere:

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

Con la nuova versione di Rails abbiamo un metodo più elegante per ottenere ciò, ovvero utilizzando il metodo `memoize` (è effettivamente `memoize` e non **memorize**). Riscriviamo l'esempio precedente utilizzando la nuova funzionalità:

```
class Person < ActiveRecord::Base
  def age
    a_very_complex_calculation
  end
  memoize :age
end
```

Il metodod *age* verrà eseguito solo la prima volta, e il suo valore di ritorno sarà salvato in memoria e successivamente recuperato per le future chiamate allo stesso metodo.

C'è solo una differenza tra i due esempi precedenti. Nel primo caso, dal momento che il metodo viene sempre eseguito se il valore presente in `@age` è `nil` o `false`, il complesso calcolo (indicato con `a_very_complex_computation`) viene eseguito nuovamente. Nel secondo esempio il metodo *age* viene eseguito solamente una volta, ed il valore di ritorno viene restituito sempre, per ogni chiamata futura (anche se è `nil` o `false`).

Se avete bisogno di disabilitare o abilitare nuovamente il caching degli attributi memoized, potete utilizzare i metodi `unmemoize_all` a `memoize_all`:

```
@person = Person.first

# Per disabilitare il caching del metodo age
@person.unmemoize_all

# Per abilitare nuovamente il cachind del metodo age
@person.memoize_all
```

NUOVO METODO: OBJECT#PRESENT?

E' stato aggiunto un nuovo metodo alla class base Object. Il metodo `present?` è equivalente all'espressione `!Object#blank?`.

In altri termini, un oggetto è "presente" se non è vuoto. Ma cosa è esattamente un oggetto vuoto?

```
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
```

Tutti questi oggetti sono vuoti, ovvero non sono "presenti".

Ma occorre fare attenzione, ciò ha generato una certa confusione. Negli esempi seguenti vengono illustrati alcuni oggetti che NON sono vuoti, ovvero sono "presenti":

```
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
```

Ogni oggetto che contiene un valore è "presente", e ciò è vero anche per un array che contiene solo `nil` perché tale array è non vuoto.

STRINGINQUIRER

Una nuova classe, `StringInquirer`, è stata aggiunta a Rails.

Per comprenderne il funzionamento, è necessario fare un esempio. Creiamo una classe `Client`, che contiene un metodo che restituisce lo `status` di un client:

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

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

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

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

Fin qui nulla di strano. Adesso modifichiamo l'implementazione del metodo `status` utilizzando la classe `StringInquirer`. Ricordiamo che il valore restituito da `status` generalmente proviene da una colonna del database (ovviamente) — questo è solo un esempio:

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

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

```
# Notate la differenza:
c.status.active?
# => true

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

Per verificare che lo `status` del client, anziché confrontare `String` utilizziamo un metodo con il valore dello status ed un punto interrogativo.

L'uso di questa tecnica è già stato introdotto in Rails stesso. Ad esempio, se occorre verificare che Rails sia attivo sull'ambiente di produzione, è sufficiente rimpiazzare il vecchio `Rails.env == "production"` con:

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

NUOVA SINTASSI PER I TEST

In Rails è stato introdotto un nuovo modo per dichiarare i test: `test/do`. Vediamo il seguente esempio:

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

Questo è il nuovo standard per il test in Rails. Vediamo, nella nuova versione, come viene generato un nuovo file per uno "unit test":

```
require 'test_helper'

class PostTest < ActiveSupport::TestCase
  # Replace this with your real tests.
```

```
  test "the truth" do
    assert true
  end
end
```

Ovviamente il vecchio modo continua ad essere supportato: i vostri vecchi test non vengono invalidati.

ABILITARE E DISABILITARE IL CARICAMENTO DELLE DIPENDENZE

Un nuovo parametro di inizializzazione è stato aggiunto per permettervi di abilitare/disabilitare il caricamento di nuove classi durante una richiesta.

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

Se `dependency_loading` è `true`, durante una richiesta Rails caricherà in memoria ogni classe che non è stata già caricata durante l'inizializzazione. Se è `false`, queste classi vengono ignorate.

Se volete eseguire il vostro progetto in un ambiente multithread dovreste disabilitare tale opzione e caricare tutte le classi attraverso l'"eager loading", oppure utilizzare il metodo `require` durante l'inizializzazione del sistema.

FILE.ATOMIC_WRITE COPIA I PERMESSI DEL FILE ORIGINE

Qualcuno di voi potrebbe avere familiarità con il metodo `File.atomic_write`. Come indica il nome, tale metodo ci assicura che la scrittura di un file sia atomica. E' molto utile in tutte quelle situazioni in cui vogliamo essere certi che altri processi o thread non possano accedere ad un file mentre viene scritto.

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

Tale metodo crea un file temporaneo durante la scrittura, terminata la quale sostituisce il contenuto del vecchio file con il nuovo.

La nuova feature di Rails 2.2 è costituita dal fatto che il nuovo file ha i medesimi permessi del file di origine.

CAMELIZE(:LOWER)

Per default, il metodo `camelize` viene utilizzato in Rails per convertire una stringa nel formato **UpperCamelCase**, ovvero in cui la prima parola della stringa - al pari delle successive parole - ha la prima lettera maiuscola. Ora è possibile convertire una stringa nel formato **lowerCamelCase** (la prima parola in minuscolo, le successive hanno solamente la prima lettera in maiuscolo) utilizzando l'argomento `:lower`. In ogni caso, se nella precedente versione di Rails provaste ad eseguire il seguente codice:

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

la prima lettera non sarebbe minuscola. Ciò è stato corretto. Il medesimo codice eseguito in Rails 2.2 produrrebbe:

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

SOSTITUZIONE DELLA LIBRERIA PER LA GENERAZIONE DELLE CHIAVI SEGRETE

Ora la classe `Rails::SecretKeyGenerator`, utilizzata per generare chiavi segrete random (per la cifratura dei cookie di sessione), è deprecata.

Adesso al suo posto Rails usa la nuova classe `ActiveSupport::SecureRandom` introdotta con Ruby 1.9. La libreria **SecureRandom** esibisce il medesimo comportamento, ma produce risultati migliori. Sono supportati i seguenti generatori di numeri casuali:

- openssl
- /dev/urandom
- Win32

Vediamo alcune chiavi generate dalla nuova libreria:

```
# 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"
```

NUOVO METODO: EACH_WITH_OBJECT

Il nuovo metodo `each_with_object`, presente in Ruby 1.9, è stato aggiunto a Rails nel caso in cui non venisse eseguito in Ruby 1.9. Questo metodo è molto interessante. E' molto simile al ben noto metodo `inject`, ma con una sottile differenza. Ogni iterazione, oltre a ricevere un elemento dalla collezione, riceve anche un oggetto chiamato **memo**.

Ad esempio, supponiamo di voler riempire un hash con i valori provenienti da una collezione:

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

Notiamo che nel precedente esempio l'oggetto memo è un hash vuoto (`{}`). All'interno del blocco popoliamo l'hash con gli item provenienti dalla collezione.

Fate attenzione al fatto che non dovreste utilizzare oggetti immutabili per la variabile memo, come numeri, `true` e `false`. Il codice seguente restituisce sempre 1, poiché il numero 1 (oggetto *memo*) non può essere modificato:

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

PRETTY URLS FACILI CON INFLECTOR#PARAMETERIZE

E' stato aggiunto un nuovo inflector, che troviamo particolarmente utile. Il metodo `parametrize` rende ogni testo "URL friendly". Osserviamo il seguente esempio:

Modello:

```
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>
```

E' interessante notare come l'implementazione iniziale di questa feature non funzionava correttamente con le lettere accentate, un bel problema per la maggior parte delle persone sparse sul globo. Un giorno Michael Koziarski ha aggiunto tale supporto. Nonostante ciò, il codice non era ancora perfezionato, così l'eccellente plugin slugalizer, creato da Henrik Nyh, è stato utilizzato a tale scopo. Ora funziona correttamente!

Tutti coloro che ancora non utilizzano Rails 2.2 possono avvalersi del plugin slugalizer.

TRE NUOVI METODI PER LE CLASSI CHE INTERAGISCONO CON DATE E ORARI

Le classi `Time`, `Date`, `DateTime` e `TimeWithZone` hanno ricevuto tre nuovi metodi molto utili. Questi sono `today?`, `past?` e `future?`, che talvolta ci possono rendere la vita meno difficile.

E' utile vedere come ciascuno di essi si comporta. Ecco i metodi in azione:

```
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
```

CAMBIAMENTO NEL METODO ASSERT_DIFFERENCE

Quanto utilizziamo il metodo `assert_difference` con più di una espressione e viene generato un errore, diventa difficile capire quale espressione causi il fallimento dell'asserzione, dal momento che il messaggio di errore non include tale informazione.

In Rails 2.2 il messaggio di errore indica chiaramente quale metodo fa fallire l'asserzione. Esempio:

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

Il precedente codice mostra il seguente messaggio di errore, nell'ipotesi che la seconda espressione non passi:

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

AGGIUNGERE UN PREFISSO AGLI ATTRIBUTI DELEGATI

Ora `Module#delegate` ha una nuova opzione `:prefix` che può essere utilizzata per fare in modo che il nome della classe target venga prefisso al nome del metodo. Vediamo due esempi. Nel primo caso, il classico uso degli attributi delegati:

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

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

Ed ora un esempio che illustra l'uso della nuova opzione `:prefix`;

```
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
```

E' anche possibile customizzare il prefisso:

```
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
```

Capitolo 3

ActiveResource

RECUPERARE L'ULTIMO RECORD DA ACTIVERESOURCE

Seguendo le convenzioni di `ActiveRecord` anche il metofo find di `ActiveResource` ora accetta l'opzione `:last`:

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

Potete utilizzarla per recuperae l'ultimo oggetto presente in una collezione di resource.

ACTIVERESOURCE::BASE #TO_XML E #TO_JSON

Sono stati aggiunti due nuovi metodi ad `ActiveResource::Base`: `to_xml` e `to_json`. Il primo fornisce una rappresentazione XML di una risorsa, mentre il secondo ne fornisce una rappresentazione in formato JSON.

Il metodo `to_json` può essere configurato utilizzando i parametri opzionali `:only` ed `:except`, i quali vi permettono di selezionare solamente determinati attributi. Esempio:

```
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"}
```

Capitolo 4

ActionPack

NUOVA OPZIONE :LAYOUT PER IL METODO CACHES_ACTION

L'opzione `:layout` è stata aggiunta al metodo `caches_action`.

```
class ListsController < ApplicationController
  ...

  caches_action :index, :layout => false

  ...
end
```

Nel precedente esempio abbiamo specificato `:layout => false`, indicante che il solo contenuto dell'azione deve essere posto nella cache, escludendo quindi il layout. Ciò è molto utile quando avete del contenuto dinamico nel vostro layout (come spesso avviene).

Se non indicate nulla, funziona come prima, ovvero viene posto in cache tutto (contenuto e layout).

TEMPLATE NELLA CACHE

Per migliorare le performance di Rails, i template delle pagine vengono automaticamente posti in cache per l'ambiente di produzione (production).

Ciò significa che se cambiate un template in produzione, è necessario riavviare il server (Rails, NdT) per vederne gli effetti.

MODIFICHE NEL METODO CONCAT

Se avete l'abitudine di evitare le ripetizioni nelle vostre view facendo ricorso agli helper, senza dubbio avrete usato il metodo `concat`. Se non l'avete mai utilizzato, vi basta sapere che è l'equivalente del metodo `puts` nelle view.

L'implementazione corrente di tale metodo accetta due parametri, una `String` con il testo che verrà mostrato nella view, e un altro parametro chiamato `binding`. A causa di alcuni miglioramenti introdotti nel codice, il parametro `binding` — benché accettato — non viene più utilizzato. Quindi l'uso di tale parametro è deprecato a partire dalla versione 2.2 di Rails.

Viene generato un messaggio di warning del tipo: "The binding argument of #concat is no longer needed. Please remove it from your views and helpers."

IL METODO LINK_TO ORA ACCETTA BLOCCHI

Il metodo `link_to` è stato aggiornato affinché accetti i blocchi. Ciò è molto utile quando avete stringhe di testo molto lunghe nei vostri link. Ad esempio, se prima era necessario qualcosa del genere:

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

Ora potete ottenere il medesimo risultato con:

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

Benché non sia un cambiamento significativo, vi permette di scrivere codice più leggibile e manutenibile, il che è molto importante.

RJS#PAGE.RELOAD

Il metodo `relaod` è stato incluso in `ActionView::Helpers::PrototypeHelper` affinché possa essere utilizzato nei template **.rjs** o nei blocchi `render(:update)`. Tale metodo forza la pagina corrente ad essere ricaricata tramite l'uso di javascript. In altri termini, è una scorciatoia per il ben noto `window.location.reload();`.

Esempio:

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

ASSEGNARE, NEI PARTIALS, UN NOME ALLE VARIABILI LOCALI DI UNA COLLEZIONE

Nel codice seguente un `partial` viene utilizzato con una collezione di oggetti:

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

All'interno del `partial` è possibile utilizzare la variabile `admin_person` per accedere agli elementi della collezione. Ma tale convenzione sui nomi è decisamente poco intuitiva.

Adesso è possibile scegliere un nome customizzato per tale variabile, attraverso l'opzione `:as`. Modifichiamo il precedente esempio:

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

Ora è possibile accedere ad ogni item della collezione attraverso la (più intuitiva) variabile `person`.

I PARTIAL NON DEFINISCONO PIÙ VARIABILI LOCALI IMPLICITAMENTE

Nell'esempio seguente, facciamo il rendering in un partial senza indicare quale variabile debba essere utilizzata per riempire il suo contenuto. Precedentemente, Rails assumeva che dovesse utilizzare una variabile di istanza avente lo stesso nome del partial.

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

A partire da Rails 2.2 tale comportamento è deprecato, e viene emesso un relativo warning:

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

Aspettatevi che tale funzionalità venga completamente rimossa in futuro.

ORA POLYMORPHIC_URL FUNZIONA CON LE SINGLETON RESOURCES

Fino ad ora l'helper `polymorphic_url` non funzionava correttamente con le singleton resource.

E' stata aggiunta una nuova patch per permettere di indicare risorse singleton attraverso i simboli, analogamente ai namespace. Esempio:

```
# Questo codice
polymorphic_url([:admin, @user, :blog, @post])

# è analogo al seguente
admin_user_blog_post_url(@user, @post)
```

SUPPORTO PER LE ESPRESSIONI REGOLARI IN TIME_ZONE_SELECT

Nella classe `TimeZone` di `ActiveSupport` c'è il metodo `us_zones` che restituisce una lista (generata dinamicamente) con tutti i fusi orari degli Stati Uniti.

Il problema è che talvolta vorreste sviluppare software per altri paesi, ma non esiste un metodo analogo per generare i fusi orari.

C'è stato un lungo dibattito sulla possibilità di create o meno metodi tipo `african_zones`, `american_zones`, etc. In ogni caso, è prevalsa la soluzione seguente.

`TimeZone` adesso supporta anche il metodo =~ in modo tale da poter generare una lista di fusi orari a partire da una espressione regolare. Anche il metodo `time_zone_select` è stato aggiornato per funzionare con tali modifiche.

Adesso potete fare qualcosa del genere:

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

Il precedente codice restituisce tutti i fusi orari, ma pone i seguenti in cima alla lista:

```
(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
```

Per approfondimenti su `TimeZone` suggeriamo di vedere l'episodio 106 di RailsCasts (http://railscasts.com/episodes/106).

ORA RENDER :TEMPLATE ACCETTA :LOCALS

I metodi `render :action` e `render :partial` accettano un `Hash` contenente i dati che vogliamo mostrare utilizzando l'opzione `:locals`, ma `render :template` non lo permetteva.

Nella corrente versione di Rails è ora possibile:

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

:USE_FULL_PATH NEL METODO RENDER NON È PIÙ DISPONIBILE

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

L'opzione `:use_full_path` del metodo `render` non è più disponibile in Rails 2.2. Nulla di importante, dal momento che non era necessaria.

DEFINE_JAVASCRIPT_FUNCTIONS

Ancora un altro metodo non più disponibile in Rails: `define_javascript_functions`.

Il che è molto sensato, dal momento che `javascript_include_tag` (ed altri) fanno la stessa cosa — e meglio.

DISABILITARE ACCEPT HEADER NELLE RICHIESTE HTTP

Iniziamo con del codice di esempio;

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

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

Nell'esempio Rails ha due modi per capire quale formato debba essere utilizzato nel blocco `respond_to`. Il primo, e più comune, è attraverso il formato indicato nella URL (ad esempio: /index.html); il secondo modo entra in gioco quando il primo non viene indicato, ovvero Rails usa il valore di **Accept** nell'header della richiesta (proveniente dal client, NdT).

Per chi non lo conoscesse, l'header **Accept** è utilizzato per indicare quale tipo di documenti (spesso chiamati MIME Types: http://en.wikipedia.org/wiki/MIME) il browser preferisca. Si utilizzano stringhe del tipo:

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

# recupera questa informazine nel codice
@request.env["HTTP_ACCEPT"] = "text/javascript"
```

Fate riferimento alla seguente URL per una lista dei MIME types più comunemente utilizzati:
http://www.developershome.com/wap/detection/detection.asp?page=httpHeaders

Questo header non è implementato efficientemente su molti browser, e utilizzato su siti web pubblici causa strani errori quando i robot indicizzatori effettuano le loro richieste HTTP.

Da qui la decisione di disabilitare tale header per default. E' sempre consigliabile indicare il formato richiesto attraverso l'URL, ma se aveste bisogno dell'header, è sufficiente includere la seguente linea di configurazione nel vostro **environment.rb**:

```
config.action_controller.use_accept_header = true
```

Quando disabilitato, se il formato non è indicato nell'URL Rails assume comunque che venga richiesto **.html**.

Esiste tuttavia un'eccezione alla regola, ovvero quando effettuate una richiesta ajax utilizzando l'header **X-Requested-With**. In questo caso il formato **.js** viene utilizzato anche se non è stato esplicitamente indicato (esempio: /poeple/1).

BUTTON_TO_REMOTE

Se utilizzate Rails da un certo periodo, probabilmente avrete familiarità con il metodo `submit_to_remote`. Questo metodo restituisce un button HTML che invia un form utilizzando **XMLHttpRequest**.

In Rails 2.2 questo metodo è stato rinominato in `button_to_remote` al fine di renderlo più consistente con il metodo `link_to_remote`.

Nessun problema per chi intende aggiornare i propri progetti: il vecchio nome (`submit_to_remote`) è un alias per il nuovo.

RICEVERE WARNINGS PER MIGLIORARE LE PERFORMANCE

Rails ha un nuovo parametro di configurazione che gli permette di segnalare dei warning se effettua il rendering di un template fuori dalla directory delle view. Il che è un bene, poiché i file che si trovano fuori da tale directory non vengono salvati nella cache, con conseguente maggiore accesso al disco.

Per ricevere tali warning è sufficiente aggiungere la seguente linea al vostro file **environment.rb**:

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

Se viene effettuato il rendering di un file fuori della directory delle views, viene generato il seguente messaggio di warning:

```
[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
```

Questa opzione è disabilitata per default.

OPZIONE :INDEX PER FIELDS_FOR

Spesso, l'opzione `:index` del metodo `select` è molto utile, ad esempio quando occorre creare più select dinamiche su una singola pagina.

Fino ad ora, il metodo `fields_for` non inoltrava questa opzione ai metodi `select`, `collection_select`, `country_select` e `time_zone_select`, limitandone l'utilità in alcuni casi.

Nella corrente versione di Rails questo è stato corretto. Esempio:

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

Restituisce:

```
<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>
```

Notate che abbiamo utilizzato l'opzione `:index => 108` nel metodo `fields_for`. Osservate le proprietà `id` e `name` del tag HTML generato dal metodo `select`: benché nessuna opzione sia stata passata a questo metodo, è stato comunque aggiunto l'opportuno indice all'output.

MIGLIORARE LE PERFORMANCE CON GLI ETAGS

Prima di addentrarci nel presente capitolo, introduciamo il concetto di ETags (Entity Tags). In prima approssimazione, possiamo dire che essi sono degli identificatori associati ad ogni risorsa, e determinano se un dato file sul server è identico a quello presente nella cache del browser. Tali "risorse" spesso sono pagine HTML, ma possono anche essere documenti XML o JSON.

Il server ha l'onere di verificare che le risorse richieste siano le medesime su entrambi i lati. Se il server è in grado di verificare ciò, anziché inviare al client l'intero contenuto della risorsa gli restituisce il codice **304**, che suggerisce al browser di utilizzare il file contenuto nella propria cache.

I web server come Apache o IIS già sono in grado di utilizzare tale meccanismo per le pagine statiche; ma quando il contenuto è dinamico, come in molti casi nei progetti Rails, la gestione di tale meccaniscmo è tutta nelle nostre mani.

L'oggetto `response` ha due nuove attributi: `last_modified` e `etag`. Quando vengono assegnati dei valori a tali attributi, questi vengono assegnati anche agli header HTTP `HTTP_IF_MODIFIED_SINCE` e `HTTP_IF_NONE_MATCH`, rispettivamente. Quando viene effettuata una nuova richiesta per una data risorsa, l'oggetto `request` restituito contiene questi due attributi, il che vi permette di confrontarli con i valori attuali della risorsa ed agire di conseguenza. Se la versione presente nella cache dell'utente è la più recente, anziché inviare nuovamente la risorsa sarà sufficiente inviare un codice "304 Not Modified", che indica al browser di visualizzare la versione presente nella sua cache.

Per fare tutto ciò sono disponibili due metodi, a seconda della situazione: `stale?` e `fresh_when`.

Esempio:

```
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
```

Nel precedente esempio, se i valori dell'header di `request` differiscono da quelli presenti nella chiamata a `stale?` significa che la cache dell'utente non è aggiornata. In tal caso il blocco `respond_to` viene invocato e i valori presenti in `stale?` vengono assegnati agli attributi `last_modified` ed `etag` dell'oggetto `response`.

Se i valori coincidono significa che la cache dell'utente è aggiornata. In questo caso il blocco `respond_to` non viene invocato, e viene restituito lo stato "304 Not Modified".

Abbiamo a disposizione anche il metodo `fresh_when`, che è una versione semplificata del metodo `stale?`. Osservate l'esempio:

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

Questo metodo assegna i valori che riceve ai rispettivi attributi dell'oggetto `response` e verifica che siano uguali a quelli dell'oggetto `request`. Se differiscono (stale) visualizza l'opportuno template. Se sono uguali (fresh) restituisce lo status "304 Not Modified" anziché visualizzare il template.

In alcuni casi potrebbe essere necessario passare un `Array` alle opzioni `:etag`, come nel seguente esempio:

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

# oppure

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

MODIFICA ALLA SIGNATURE DEL METODO NUMBER_WITH_DELIMETER

Il metodo `number_with_delimiter` è stato riscritto. Oltre a rendere il codice più chiaro, la signature del metodo è stata modificata. Prima era così:

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

# esempi d'uso
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, " ", ",")
```

Questa è la nuova versione:

```
def number_with_delimiter(number, *args)

# esempi d'uso
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 => ",")
```

Quindi siate sicuri, prima di utilizzare il metodo `number_with_delimiter`, di aver indicato le opzioni tramite un hash di simboli.

ESEGUIRE PIÙ ISTANZE DI UN PROGETTO IN PATH DIFFERENTI

Talvolta potreste avere l'esigenza di eseguire più copie dello stesso progetto. Magari avete un prodotto utilizzato da diversi clienti, oppure volete eseguire contemporaneamente una versione di test accanto ad una di produzione.

La via più semplice per ottenere ciò consiste nell'avere più (sotto)domini, ognuno con la propria istanza dell'applicazione. Qualora ciò non fosse possibile potreste spostare la vostra applicazione in una sottodirectory e utilizzare un prefisso nell'URL per distinguere ogni istanza della vostra applicazione. Ad esempio, potreste avere più blog per utenti diversi con URL del tipo:

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

In questo caso i prefissi **johndoe**, **jilldoe** e **joedoe** identificano le istanze dell'applicazione che vengono eseguite nelle corrispondenti, omonime, sottodirectory. Il routing dell'applicazione avviene dopo. Possiamo dire a Rails di

ignorare questa parte dell'URL quando viene effettuata una richiesta, ma continuare ad includerla nelle URL che vengono generate. Ciò può essere configurato attraverso la costante `RAILS_RELATIVE_URL_ROOT` oppure il metodo `AbstractRequest.relative_url_root`.

Se il vostro progetto Rails è in esecuzione su Apache, questa feature è già attivata automaticamente, in molti casi non è quindi necessario preoccuparsene. Ma ricordate: solo se state usando Apache.

In ogni caso, in Rails 2.2 `relative_url_root` non viene configurato automaticamente dall'header HTTP. E' necessario farlo automaticamente, inserendo una direttiva del genere in ogni file **environmente.rb**:

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

Fatto ciò la vostra applicazione ignorerà il prefisso "johndoe" che compare dopo il dominio. Ma quando vengono generate le URL, tale prefisso verrà sempre inserito al fine di garantire l'accesso all'applicazione corretta.

MODIFICHE IN ERROR_MESSAGE_ON

Il messaggio `error_message_on` è molto utile. Potete utilizzarlo per accedere ai messaggi di errore restituiti da certi metodi di un oggetto.

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

<!-- oppure -->

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

Questo visualizza un messaggio di errore all'interno di un tag DIV se c'è un errore associato al campo "title" del modello "Post".

Ma l'aspetto più interessante del metodo `error_message_on` è che possiamo personalizzarlo per visualizzare messaggio più "amichevoli". Qui entrano in gioco le modifiche introdotte con Rails 2.2.

Nella precedente versione i parametri di personalizzazione venivano passati direttamente al metodo, mentre il Rails 2.2 vengono passati con un `Hash`:

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

Non preoccupatevi del vostro vecchio codice, poiché continua a funzionare anche la vecchia modalità (almeno per un po') con qualche messaggio di warning che vi invita ad aggiornare il vostro codice.

DIVERSI METODI SONO STATI MODIFICATI PER ACCETTARE OPZIONI TRAMITE HASH

Anche i seguenti metodi sono modificati per accettare i propri argomenti come `Hash`, rendendo il vostro codice più leggibile e facile da manutenere.

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>."
```

Tutti questi metodi continuano a funzionare con il vecchio formato (per il momento), limitandosi a mostrare messaggi di warning nei log dell'applicazione per ricordarvi di aggiornare il vostro codice al più presto.

PIÙ FUNZIONALITÀ PER I PARTIALS

Osservate i seguenti esempi:

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

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

L'output di questo codice è:

```
beginning
middle
ending
```

Nel precedente esempio stiamo includendo un partial all'interno della nostra vista utilizzando il metodo `yield` per modificarne il contenuto, specificato all'interno del blocco passato al metodo `render`.

Fino ad ora, comunque, non potevate passare alcun argomento al blocco. In Rails 2.2 ciò è possibile. Potete realizzare cose carine. Osservate il seguente esempio di una collezione di libri:

```
<!-- 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 %>
```

Restituisce qualcosa del genere:

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

Notate come, all'interno del blocco, abbiamo indicato il title de libro, ma in un'altra vista possiamo anche aggiungere altre informazioni, sempre con la medesima tecnica.

Potete anche utilizzare lo stesso partial più volte nella medesima view, e utilizzare i blocchi per distinguere tra le sezioni della pagina. Esempio:

```
<!-- 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 %>
```

RECUPEARE LA STRINGA CORRENTE DAL METODO CYCLE

Probabilmente avrete già familiarità con il metodo `cycle`. E' utilizzato per alternare i colori delle righe di una table HTML, cambiando la classe CSS di ogni riga.

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

Un nuovo metodo di supporto è stato creato per lavorare insieme a `cycle` in tutte quelle situazioni che lo richiedono, o dove è necessario recuperare il valore della stringa per l'iterazione corrente. Vediamo il seguente esempio che utilizza il nuovo metodo `current_cycle`:

```
@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 %>
```

EVITARE LE DUPLICAZIONI DEGLI ARTICOLI NEI FEEDS

Talvolta ci capita di sottoscriverci ai feed di qualche blog, e subito ci ritroviamo dei post che nonostante siano stati già letti compaiono ancora nel nostro feed reader. Vi è mai successo? Capita per una serie di motivi, ma ovviamente non vogliamo che capiti a chi usa le nostra applicazioni, giusto?

Per aiutarci ad evitare questo inconveniente, ogni entry di un feed generato con il builder `atom_feed` adesso ha una nuova opzione `:id`, che vi permette di customizzare l'id stesso.

```
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
```

Se procedete in questo modo, anche se dovrete riscrivere una porzione del codice che genera i vostri feed, oppure apportare qualche profonda modifica al contenuto del vostro sito, il campo `id` di ciascuna entry non cambierà. In tal modo il feed reader non duplicherà le vecchie entry.

I vostri lettori vi ringrazieranno.

AGGIUNGERE ISTRUZIONI DI PROCESSAMENTO AI VOSTRI DOCUMENTI XML

Una nuova opzione è stata aggiunta al metodo `atom_feed`. Adesso è possibile includere istruzioni per processare l'XML. Esempio:

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

Le istruzioni per processare l'XML vengono lette dall'applicazione che richiede il file. Vengono spesso utilizzate per informare l'applicazione su come dovrebbe manipolare i dati XML.

Nell'esempio precedente, stiamo dicendo all'applicazione ricevente che dovrebbe visualizzare il documento XML con uno specifico foglio di stile. Ecco il risultato:

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

SEMPLIFICARE L'ACCESSO ALLE RISORSE

I route annidati sono ormai un argomento ben consolidato in Rails. Quando create i vostri route, è usuale avere un codice del genere:

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

```
    post.resources :comments
  end
end
```

Nel codice precedente stiamo esplicitando il fatto che i nostri users hanno dei posts, e questi posts hanno dei commenti. In accordo con tale impostazione dei route, possiamo accedere ai post di uno user utilizzando l'URL '**/users/1/posts**', oppure recuperare i commenti di un particolare post con l'URL '**/users/1/posts/5/comments**'.

Con la nuova opzione `:shallow => true` otteniamo maggiore flessibilità. Notate che aggiungendo tale opzione alla risorsa più esterna, tutte le altre risorse (annidate) ereditano questa caratteristica.

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

Con questa opzione abilitata, possiamo comunque continuare a recuperare i dati come di consueto. Il vantaggio risiede nel poter, ad esempio, recuperare tutti commenti di un post senza dover indicare lo user (es: '**/posts/2/comments**'). Oppure possiamo recuperare un determinato post utilizzando solamente '**/posts/2**'.

L'opzione `:shallow` funziona anche con risorse che usano `has_many` oppure `has_one`, come le seguenti:

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

Vengono creati anche tutti gli helper per accedere direttamente ai route:

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

DEFINIRE ROUTE CON ARRAY DI VERBI HTTP VALIDI

Adesso è possibile passare un array di metodi a nuovi membri o collezioni di route. Ciò elimina la necessità di definire un route che accetti ogni verbo HTTP (`:any`) quando vogliamo che risponda solamente a qualche verbo. In Rails 2.2 possiamo dichiarare un route del genere così:

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

ROUTE POLIMORFICI

Adesso i metodi `*_polymorphic_url` e `*_polymorphic_path`, molto utilizzati per generare URL a partire dai record del database, accettano un parametro opzionale. Oltre agli usuali parametri accettano anche un hash di opzioni, rendendo di fatto possibile generare nuovi route con parametri addizionali nell'URL.

Vediamo qualche esempio. Sotto a ciascun esempio viene mostrato un commento contenente il metodo equivalente:

```
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 RIMOSSO DA RAILS

L'helper `country_select` è stato rimosso da Rails. Per chi non lo ricordasse, questo metodo restituiva una lista con tutti i paesi del mondo.

La ragione di tale rimozione è dovuta al fatto che Rails è stato aggiornato per utilizzare lo standard ISO 3166 per i nomi dei paesi. Il principale problema è che, secondo lo standard ISO 3166, Taiwan (ad esempio) viene indicata come "Taiwan, Provincia della Cina". E ciò è esattamente quello che Michael Koziarski ha inserito nel metodo.

Quindi, Jamis Buck si chiese se non fosse possibile inserire semplicemente "Taiwan", dal momento che l'indicazione "Provincia della Cina" sembrava essere politicamente aggressiva. Una pletora di commenti esplosi su Github ha spostato l'attenzione sul lato politico piuttosto che su quello tecnico.

Ma Michael Koziarski è stato categorico nell'affermare che le questioni politiche erano ben lontane dal poter essere risolte con un semplice aggiornamento del codice. E che se avesse accettato tale cambiamento, presto altri sarebbero potuti arrivare (Kosovo, Ossezia del Sud, Abcazia, Transnistria, etc).

La migliore soluzione, o almeno quella meno controversa, consiste nel rimuovere l'helper in questione da Rails e renderlo disponibile come plugin. In tal modo chiunque può fare un fork del codice e modificarlo a proprio piacimento.

Per installare il plugin ufficiale:

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

REPORT DEI BENCHMARK ESPRESSI IN MILLISECONDI

Tutti i messaggi di log che mostrano il tempo speso da un processo per la sua esecuzione sono stati modificati affinché visualizzino il tempo in millisecondi.

Ad esempio, il messaggio:

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

viene adesso mostrato così:

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

OPZIONE :CONFIRM NEL METODO IMAGE_SUBMIT_TAG

L'opzione `:confirm`, utilizzata in molti helper (es: `link_to`), è ora disponibile anche nel metodo `image_submit_tag`.

Questa opzione fa in modo che venga aperto un messagebox (javascript, NdT) contenente un messaggio personalizzato quando l'immagine viene cliccata. Se l'utente conferma, il form viene inviato normalmente, altrimenti non accade nulla.

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

ORA I COOKIE DI SESSIONE SONO SOLO HTTP

Rails ha un opzione per impostare i cookie che viene spesso trascurata. L'opzione `:http_only` fa in modo che un cookie possa essere acceduto solo tramite HTTP, impedendone quindi l'accesso tramite codice javascript. Il valore di default è `false`.

In Rails 2.2, i cookie di session hanno l'opzione `:http_only` impostata per default. Lo scopo di questa funzionalità è di aumentare la sicurezza dei vostri progetti. Ovviamente può essere disabilitata se necessario. Se volete, basta includere la seguente linea di codice in `ApplicationController` oppure in uno specifico controller:

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

PATH DELLE VISTE COMPLETO NELLE ECCEZIONI

Quando un'eccezione viene sollevata nell'ambiente di sviluppo (development), le vostre viste mostrano un messaggio del genere:

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

Ma sarebbe bello avere un messaggio con il path completo del file che ha generato l'eccezione:

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

Per rendervi la vita più semplice, questo problema è stato risolto nella nuova versione di Rails.

OPZIONI NEL METODO FIELD_SET_TAG

Il metodo `field_set_tag` riceve un nuovo parametro che rende la formattazione dell'HTML più semplice. Questo parametro accetta tutte le opzioni del metodo `tag`. Esempio:

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

Questo codice produce:

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

SUPPORTO ALL'XHTML IN ATOM_FEED

Adesso l'helper `atom_feed` ha un builder interno che vi permette di creare XHTML semplicemente aggiungendo `:type=>"xhtml"` agli elementi `content`, `rights`, `title`, `subtitle` o `summary`. Esempio:

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

Osservate come questo blocco si innesta nella chiamata `atom_feed`:

```
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
```

In tal modo, il builder interno di `atom_feed` includerà all'interno di un tag **div** l'XHTML generato.

Ovviamente, potete ancora utilizzare il vecchio modo di passare tutto l'HTML all'interno di una stringa, ma la nuova modalità è ovviamente più pulita.

EVITARE ATTACCHI DEL TIPO "RESPONSE SPLITTING"

Fino ad ora le URL passate al metodo `redirect_to` non venivano "sterilizzate" (sanitized, NdT). Ciò era fonte di potenziali pericoli, poiché permetteva ad utenti malevoli di generare attacchi del tipo **response splitting** e **header injection** nelle vostra applicazione.

Un esempio di questo problema si può presentare quando la vostra applicazione riceve una URL attraverso una query string e reindirizza l'utente verso tale URL usando il metodo `redirect_to`. Questo varco permette ad utenti malevoli

di salvare dei cookie nella macchina dell'utente, e quindi contraffare le risposte inviate ai vostri utenti se la vostra applicazione usa questi parametri per costruire gli header HTTP.

Per arginare questo tipo di problemi Rails è stato aggiornato affinché sterilizzi tutte le URL passate al metodo `redirect_to`. In ogni caso, questo non significa che non dovrete prestare la dovuta attenzione a tali problemi, è sempre consigliabile fare un doppio check e stare all'erta.

MIGLIORIE PER CATTURARE GLI ERRORI

La peggiore cosa che possa capitare alla vostra applicazione è quell'orribile pagina di errore (di Rails, NdT). E' sempre meglio essere pronti per tale evenienza. Adesso è possibile presentare una opportuna pagina di errore quando la vostra applicazione solleva un'eccezione. Si usa il metodo `rescue_from`, esempio:

```
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
```

L'aggiunta di `ActiveSupport:Rescuable` permette ad ogni classe di includere la funzionalità `rescue_from`.

NUOVA OPZIONE :RECURSIZE PER JAVASCRIPT_INCLUDE_TAG E STYLESHEET_LINK_TAG

Ora gli helper `javascript_include_tag` e `stylesheet_link_tag` hanno la nuova opzione `:recursive`, che può essere usata insieme ad `:all`. Questa permette di caricare l'intero albero dei file con una sola linea di codice. Ad esempio, supponiamo di avere i seguenti file:

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

Quando eseguiamo il seguente codice, entrambi i file vengono inclusi (anche se si trovano in sottodirectory):

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

Capitolo 5

ActionMailer

UTILIZZARE I LAYOUT NELLE EMAIL

Con la nuova versione di Rails, abbiamo una funzionalità che delizierà gli spammer!

Scherzi a parte, come abbiamo i layout per i controller, adesso li abbiamo anche per i mailer. E' sufficiente aggiungere `_mailer` ai nomi dei vostri file, è il layout verrà applicato automaticamente.

Vediamo qualche esempio. Anzitutto le view:

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

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

Quindi `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
```

In tal modo, ogni nuovo mailer utilizzerà questo layout.

Per impedire ad un mailer di utilizzare il layout di default, è sufficiente includere `:layout => false` nel corpo del messaggio della mail.

```
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
```

Potete anche indicare un layout differente cambiando il valore dell'opzione `:layout`. In tal caso, il layout utilizzato non deve necessariamente avere il suffisso `_mailer`.

```
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
```

Infine, potete anche indicare ad un `ActionMailer` quale layout utilizzare:

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

Capitolo 6

Railties

I PLUGIN APPARTENGONO AL PASSATO?

In Rails 2.1 era possibile utilizzare le gem come plugin per i nostri progetti. Per far ciò era sufficiente creare la directory **rails** all'interno della directory contenente le gem del progetto (`vendor/gems`, NdT), e includere un file **init.rb**.

Ciò giustificava l'esistenza di `config.gem` e `rake gems`. Ma dal momento che è possibile caricare le gem direttamente dentro la nostra applicazione Rails, prima o poi i plugin spariranno?

Sembra che ciò stia proprio accadendo. Con la nuova versione di Rails, ad esempio, è stata introdotta una modifica che ci permette di inizializzare i plugin indifferentemente attraverso **init.rb** nella directory root dei plugin, oppure con **rails/init.rb** (analogamente a quello che avviene per le gem). Quest'ultima strada è quella consigliata.

Così potete creare una gem (che si comporta come un plugin) ed istallarla in due possibili modi:

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

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

Questo ci evita anche di dover mantenere due file **init.rb** (uno nella root dei plugin e l'altro nella directory rails).

MIGLIORATO IL SUPPORTO PER THIN

Ora `script/server` verifica che **Thin** sia presente e, in tal caso, cerca di usarlo. Ciò è molto utile se state utilizzando **Thin** nel vostro ambiente di produzione (e magari anche in sviluppo). Per fare in modo che ciò funzioni è necessario che aggiungiate la seguente linea di codice ad **environment.rb**:

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

EFFETTUARE I TEST, CON GIT, DEI SOLI FILE FUORI COMMIT

C'è un task **rake** veramente utile che in pochi conoscono:

```
rake test:uncommitted
```

Come indica il nome, esegue i test per i soli file che non sono ancora stati inseriti nella commit al repository Subversion anziché eseguire tutti i test.

Abbiamo utilizzato molto questa funzionalità, ma passando a Git ci siamo accorti che non era ancora supportata. Tuttavia una recente patch garantisce di avere il supporto anche per Git da oggi in poi.

RAILS.INITIALIZED?

Un nuovo metodo chiamato `Rails.initialized?` è stato aggiunto a Rails. Vi permette di sapere se tutte le operazioni relative all'inizializzazione del framework sono terminate.

ORA CACHE_CLASSES È ABILITATO PER DEFAULT

Nei file di configurazione del vostro progetto c'è sicuramente la seguente linea di codice:

```
config.cache_classes = false
```

In pratica dice a Rails di non mettere in cache il codice del vostro progetto; ovvero, per ogni richiesta dei client deve ricaricare nuovamente tutto il codice. Benché ciò rallenti l'esecuzione, è ideale in fase di sviluppo perché vi evita di dover far ripartire il server ad ogni modifica che fate sul codice.

In produzione, al contrario, è cruciale abilitare tale opzione.

In Rails 2.1 se la precedente linea di codice non era presente nel vostro file di configurazione, Rails assumeva di non dover metter in cache le classi — questo per default.

In Rails 2.2 tale comportamento è stato invertito; se non viene trovata questa opzione di configurazione allora tutte le classi vengono messe in cache. Ciò aiuta gli utenti Rails meno esperti a non installare i propri progetti con la modalità di sviluppo.

EVITARE CHE CONFIG.GEM CARICHI DETERMINATE GEM

Una delle novità introdotte con Rails 2.1 era `config.gem`, che vi permetteva di indicare a quali gem il vostro progetto dipendeva.

Con questa nuova funzionalità sono stati aggiunti alcuni task (rake, NdT) per semplificarci la vita, come `rake gem:install` che istalla automaticamente tutte le dipendenze.

Ma occorre fare attenzione nel configurare le dipendenze, dal momento che in alcuni casi il nome della gem non coincide con il nome del file da passare a `require`. Ad esempio, la gem **aws-s3** dovrebbe essere caricata con il nome **aws/s3**, sostituendo il trattino con uno slash. Per prevenire questa situazione è stata aggiunta l'opzione `:lib` a `config.gem`. Ecco come si comporterebbe per l'esempio appena fatto:

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

Dopo un po' è emerso un altro problema: "Il mio progetto dipende da tale gem, ma non voglio caricarla adesso. Come posso fare?".

Nella nuova versione di Rails è possibile impostare a `false` l'opzione `:lib`, che impedisce di caricare la gem.

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

Pur senza caricare la gem, Rails la istallerà comunque quando verrà lanciato il task `rake gems:install`, e sarà inclusa in ogni altro task relativo alle gem.

THREADSAFE!

Ora Rails ha un nuovo metodo di configurazione per abilitare la modalità multithread:

```
config.threadsafe!
```

Quando questo metodo è presente nei vostri file di configurazione (generalmente in **environments/production.rb**) abilitate le azioni dei vostri controller ad accettare più richieste contemporaneamente, utilizzando più connessioni al database. Dipendentemente dalla vostra applicazione, e dall'ambiente di esercizio, ciò permette di gestire più richieste poiché meno copie di Rails vengono caricate in memoria (lasciandola, quindi, libera — NdT).

Questo, inoltre, disabilita il caricamento post-istallazione delle dipendenze. Per maggiori informazioni fate riferimento al capitolo "Abilitare e disabilitare il caricamento delle dipendenze" nel capitolo **ActiveSupport**.

FIXTURES_PATH

I task `rake db:fixtures:load` e `rake db:fixtures:identify` prendono un nuovo parametro opzionale: `FIXTURES_PATH`.

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

In questo modo potete indicare un path alternativo per le vostre fixtures (es: spec/fixtures).

ASSOCIAZIONI BELONGS_TO AUTOMATICHE NELLA GENERAZIONE DEGLI SCAFFOLD

Se state già utilizzando Rails 2.2 provate ad eseguire il seguente comando per creare un nuovo modello:

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

Notate come abbiamo indicato che comments ha un riferimento alla tabella `posts`, oppure — detto il altri termini — che un commento appartiene (`belongs_to`) ad un post. Ora osservate il file generato per il modello (**app/models/comment.rb**):

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

La relazione tra le due tabelle è stata automaticamente aggiunta al modello. Questa è una nuova funzionalità della versione 2.2.

MESSAGGI DI AIUTO PER I PRINCIPANTI DI RAILS

Un nuovo messaggio è stato aggiunto al file 500.html per aiutare i programmatori che sono appena agli inizi con Rails:

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

Rails tiene a cuore i nuovi arrivati.

DEBUGGING NELLA CONSOLE RAILS

Come potete far partire Rails con il debugger, attraverso `script/server --debugger`, lo stesso potete fare con la console: `script/console --debugger`. Questa opzione carica la libreria **ruby-debug** quando la console viene inizializzata.

E' più semplice utilizzare tale opzione anziché dovere digitare `require 'ruby-debug'` nella console quando vi occorre.

ORA DB:MIGRATE:REDO ACCETTA LA VERSIONE DELLE MIGRATION

Il task `rake db:migrate:redo` è sempre stato molto utile per testare e rieseguire le più recenti migration. Ora questo task è ancora più utile, vi permette infatti di indicare attraverso l'opzione `VERSION` quale migrazione volete eseguire di nuovo:

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

SUPPORTO PER IL DATABASE IBM DB2 NEL GENERATORE DI RAILS

E' stata aggiunta un'opzione che vi permette di generare progetti Rails con il supporto per il database IBM DB2. Per creare un progetto Rails con tale supporto è sufficiente eseguire il seguente comando nel vostro terminale:

```
`rails app_name -d ibm_db`
```

Rails creerà il vostro progetto con il seguente contenuto per il file **database.yml**:

```
# 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
```

Capitolo 7

Internazionalizzazione (i18n)

Tutto cominciò nel settembre del 2007 quando un gruppo di sviluppatori iniziò a scrivere un plugin chiamato **rails-i18n**, il cui scopo era quello di eliminare la necessità di patch manuali su Rails per ottenere un'applicazione internazionalizzata.

CHE COSA SIGNIFICA I18N?

Anzitutto, per comprendere l'acronimo, è necessario avere profonde conoscenze matematiche. Contiamo insieme, quante lettere ci sono tra la I e la N della parola "internationalization"? 18. Molto bene: i18n.

Discorso analogo per "localization" e l10n.

Avete già visto alcuni siti con le bandierine dei paesi in alto, che vi permettono di scegliere la lingua? Quando cliccate su una di esse, l'intero contenuto del sito cambia con la lingua selezionata. Questo viene indicato con "internatinalization" o, come abbiamo appena visto, i18n.

Ovviamente siamo stati un po' superficiali. Il più delle volte non è solo il testo a cambiare da un paese ad un altro, non possiamo non considerare il formato delle date, i fusi orari, le unità di misura e, forse l'elemento principale, la valuta.

I18N E L10N, CHE DIFFERENZA C'È?

L'internazionalizzazione è quando predisponete il vostro software a poter essere utilizzato per differenti paesi.

La localizzazione è quando adattate un vostro software per un particolare paese, ad esempio prendendo un sito statunitense che accetta pagamenti solo tramite PayPal e modificandolo affinché accetti anche pagamenti con altre banche.

CHE NOVITÀ CI SONO IN RAILS?

In Rails 2.2 il plugin per l'internazionalizzazione è stato aggiunto al core.

Questo non significa che Rails sia stato sottoposto a cambiamenti radicali. Allo stato attuale, quasi nulla è stato modificato. Continua ad essere il framework di sempre, con le validazioni in inglese, etc.

La differenza e che se avevate necessità dei messaggi in portoghese, o in qualsiasi altra lingua, dovevate creare una patch a mano. Non possiamo non menzionare il noto Brazilian Rails, che faceva esattamente questo per tradurre i messaggi di ActiveRecord.

Il principale miglioramento consiste nel fatto che in Rails 2.2 adesso disponiamo di un modo più semplice e standardizzato per farlo, utilizzando un'interfaccia comune.

COME FUNZIONA?

Praticamente questa gem è divisa in due parti:

- La prima parte aggiunge un manciata di nuovi metodi alle API di Rails. Tali metodi vi permettono di accedere all'altra parte della gem.
- La seconda è un semplice backend che implementa tutta la logica richiesta per fare in modo che Rails funzioni come prima, utilizzando per default la configurazione en-US.

La differenza principale consiste nel fatto che questo modulo può essere sostituito da un altro che supporta l'internazionalizzazione da voi richiesta. Ancora meglio: sta nascendo una serie di plugin che fanno proprio questo.

Le fondamenta di tale implementazione risiedono in un modulo chiamato `i18n`, il quale proviene da una gem incorporata in `ActiveSupport`. Questo modulo aggiunge le seguenti funzionalità a Rails:

- Il metodo `translate`, utilizzato per recuperare le traduzioni.
- Il metodo `localize`, utilizzato per "tradurre" gli oggetti `Date`, `DateTime`, e `Time`.

Oltre a tali metodi questo modulo contiene il codice necessario per caricare e conservare i backend di localizzazione. Inoltre, è provvisto di un gestore di eccezioni standard in grado di catturare le eccezioni sollevate nei backend.

Sia il backend che il gestore di eccezioni possono (e dovrebbero) essere estesi. Esistono anche alias per `#translate` e `#localize` che vi permettono di risparmiare un po' di tempo: `#t` ed `#l`, rispettivamente. Questi metodi sono disponibili sia nelle viste che nei controller.

ESEMPI PRATICI

Il miglior modo per imparare ad utilizzare il supporto all'internazionalizzazione presente in Rails è vedere come funziona in pratica. Vi suggeriamo di dare un'occhiata al progetto creato da Sven Fuchs e compagnia su: **http://i18n-demo.phusion.nl/**.

SUPPORTO ALLA PLURALIZZAZIONE NEI METODI DI INTERNAZIONALIZZAZIONE

Talvolta alcune traduzioni dipendono da un numero o una quantità. E' uno scenario tanto comune che il package per l'internazionalizzazione ha dovuto necessariamente tenerne conto.

Ad esempio, nel metodo `distance_in_words` (riferita al tempo, NdT), la frase "1 secondo" funziona bene quando il tempo è minore o uguale ad uno, ma deve essere pluralizzata quando il tempo è più lungo.

Nei file di localizzazione potete internazionalizzare le frasi che nella vostra applicazione dipendono da una quantità, come in questo caso:

```
datetime:
  distance_in_words:
    x_seconds:
```

```
        one: "1 secondo"
        other: "{{count}} secondi"
```

E' un'utile funzionalità utilizzata in diversi metodi nativi di Rails, e che potete utilizzare nei vostri metodi.

Capitolo 8

Prestazioni

MIGLIORARE LE PRESTAZIONI DI RAILS

Jeremy Kemper ha lavorato su diversi miglioramenti delle prestazioni di Rails. Ad esempio ha migliorato l'efficienza di **ERb**, come pure alcuni metodi come `concat` e `capture`, che vengono utilizzati da molti **helper** in Rails.

Jeremy ha anche rivisto il processo di inizializzazione dei **partial** ed ha ottimizzato diversi helper che generano codice **javascript**.

Anche la classe `RecordIdentifier` è stata migliorata attraverso l'uso del caching. `RecordIdentifier` ingloba una serie di convenzioni per maneggiare oggetti `ActiveRecord` e `ActiveResource`, o ogni tipo di modello che ha un `id`.

E' interessante osservare questo tipo di azioni. Rails sta diventando più grande e complesso, pertanto i processi di ottimizzazione dovranno diventare parte integrante del futuro sviluppo del framework.

CREARE TEST PER LE PRESTAZIONI

Rails 2.2 ha un nuovo **generatore** che vi permette di creare dei test di prestazioni (performance test).

Eseguendo il seguente comando in un terminale:

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

viene generato il file **test/performance/login_test.rb**. Vediamo il codice prodotto:

```
require 'performance/test_helper'

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

In questo file potete inserire tutti i test legati alle prestazioni che volete misurare, quando questi verranno eseguiti verrà prodotto un report con tutte le informazioni ad essi relativi, con i tempi di esecuzione, l'uso della memoria, etc. Per fare ciò eseguite il seguente comando nel vostro terminale:

```
[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.
```

Capitolo 9

Bug e correzioni

ACTIVERECORD

Correzioni al conflitto dei namespace tra named_scope e :joins.

Quando utilizzavate `with_scope` insieme a `:joins`, tutti gli attributi delle tabelle secondarie venivano aggiunti al modello della tabella principale.

Il metodo find_all non funzionava in named_scope

Quando eseguivate il metodo `find_all` in un `named_scope`, questo non veniva reindirizzato al metodo `proxy_found` come avrebbe dovuto. Ciò sollevava l'eccezione `NoMethodError`.

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

Questo problema poteva essere aggirato utilizzando il metodo `to_a`:

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

Nella nuova versione di Rails tale problema è stato risolto.

I partial update non aggiornavano lock_version se nulla veniva modificato

Quando utilizzavate il locking ottimistico con i partial update venivano eseguite delle query non necessarie. Questo problema è stato risolto.

CORREZIONI IN TIME#END_OF_QUARTER E DATE#END_OF_QUARTER

Non appena fu rilasciato Rails 2.1 venne incontrato un errore serio. Se avete dei progetti creati sotto questa versione, aprite ad una sessione **irb** e provate:

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

ERROR!

Come mai? L'implementazione di `end_of_quarter` aveva un bug. In pratica arrivava al termine dell'ultimo mese del trimestre, e quindi recuperava l'ultimo giorno. Il problema è che avanzava solo sul mese, e poiché nell'esempio si parte dal 31 marzo, il metodo cerca di creare un'istanza di `Date` con il 31 giugno, che non esiste. Nel caso di un oggetto `Time` non viene sollevata alcuna eccezione, ma viene restituita la data sbagliata: 31 luglio.

Questo errore è stato corretto nell'attuale versione, ma se state ancora utilizzando Rails 2.1 per qualche progetto fate attenzione, dal momento che questo errore si manifesta quando utilizzate il metodo `end_of_quarter` con l'ultimo giorno di maggio, luglio o agosto.

CORREZIONE AI TASK DB:MIGRATE:DOWN E DB:MIGRATE:UP

Nelle precedenti versioni di Rails, quando lanciavate `rake db:migrate:down VERSION=some_version_number` i record nella tabella **schema_migrations** non venivano aggiornati.

Questo comportava che dopo aver eseguito `rake db:migrate:down` o `up`, se aveste provato ad eseguire `rake db:migrate` alcune migrazioni non sarebbero state eseguite. Lasciateci mostrare il problema:

```
$ ./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)

$
```

Questo problema è stato corretto facendo in modo che la tabella **schema_migrations** venga effettivamente aggiornata dopo avere eseguito questi task.

POSTGRESQL

In PostgreSQL la sintassi **typecast** è del tipo: `#bodylt;column>::#bodylt;type>`

`ActiveRecord` interpretava questa sintassi come un nome "bind", e si lamentava del fatto che il valore non venisse passata tramite un `Hash`. Adesso questo problema è stato corretto, permettendovi di fare cose del tipo:

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

BUG CORRETTO IN RENAME_COLUMN

Questa modifica consiste in una correzione del metodo `rename_column`. Per comprendere quale sia il problema, prendiamo uno scenario di esempio. Anzitutto, creiamo una **migration**:

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

Quindi creiamo una seconda **migration** che rinomini la colonna **name**:

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

Se provate questo esempio, noterete che una volta eseguito il metodo `rename_column`, la "nuova" colonna `first_name` non ha più il valore di default assegnato nella prima **migration**, come invece dovrebbe.

Questo bug è stato corretto nella corrente versione di Rails.

IL METODO ACTIVERECORD#COUNT NON INCLUDE UN ALIAS PER LE ASSOCIAZIONI

Supponiamo che abbiate la seguente associazione `has_many :through`:

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

Quando cercate un libro, potete includere le informazioni dell'autore:

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

Questo non presenta alcun problema, ma provando ha fare la stessa cosa con il metodo `count`:

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

Otteniamo un errore. Questo accade perché la tabella **authorships** viene inclusa due volte nella stessa query.

Il metodo `find` è più intelligente, dal momento che crea un alias per la tabella, cosa che il metodo `count` non fa. Sappiamo che probabilmente l'esempio fatto non è dei migliori, ma dimostra comunque il problema in oggetto.

Questo bug è stato corretto. Adesso il metodo `count` si comporta come `find` in tutti i casi in cui si presenta un `:include`.

BUG CORRETTO IN ACTIVERECORD#COUNT

C'era un bug in `ActiveRecord#count` che si manifestava con `has_many` unitamente alle opzioni `:limit` o `:offset`. Vediamo un esempio:

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

Nel codice sopra, quando cercate di recuperare i commenti di un post solamente due di questi vengono restituiti.

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

# Notate l'SQL generato:
# SELECT * AS count_all FROM "comments" WHERE
# ("comments".post_id = 1) LIMIT 2
```

Ma quando utilizziamo il metodo `count`:

```
post.comments.count # => 3

# Notate l'SQL generato:
# SELECT count(*) AS count_all FROM "comments" WHERE
# ("comments".post_id = 1)
```

Come potete vedere, l'errore è dovuto al fatto che la clausola `LIMIT 2` non veniva inclusa nella query SQL.

Ovviamente questo problema è stato risolto con la versione 2.2 di Rails.

BUG CORRETTO IN SUBMIT_TAG

Quando utilizzavate il metodo `submit_tag` con l'opzione `:disable_with` abilitata, all'invio del form verso il server veniva soppresso il parametro `:commit`. Questo perché dopo aver sottomesso il form, l'evento javascript **onclick** prima disabilitava il pulsante e quindi inviava il form al server, ed in un form HTML gli elementi disabilitati non vengono sottomessi.

Ciò costituisce un problema in tutti i casi in cui il form ha più di un `submit_tag` e la sua logica di creazione/aggiornamento dipende dai valori del parametro `:commit`.

Questo problema è stato risolto inserendo del codice, in testa al javascript, che copia i valori di tale parametro in un campo **hidden** del form e quindi lo invia al server, anche se il pulsante è disabilitato.

TEST DEI NAMED ROUTE CON BUG

Esiste un determinato bug nelle versioni precedenti la 2.2 che compare quando un functional test verifica un named route con parametri prima di eseguire la richiesta. Per capire di cosa stiamo parlando, osservate il seguente esempio:

```
def test_something
  post_url(:id => 1) # Prima della richiesta questo genera un errore
  get :edit, :id => 1
  post_url(:id => 1) # Qui funziona
end
```

Questo problema è stato corretto in Rails 2.2.

ADESSO TIME#ADVANCE RICONOSCE GIORNI E SETTIMANE PARZIALI

Dopo il rilascio di Rails 2.1 il metodo `Time#advance` smise di funzionare correttamente con i periodi di tempo parziali. Esempio:

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

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

Questo bug è stato corretto in Rails 2.2.

ERRORE NEL CREARE DUE CONTROLLER CON LO STESSO NOME

In alcuni casi con Rails 2.1 se cercavate di creare due controller con lo stesso nome, ma in namespace differenti, veniva comunque generato un errore:

```
$ 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.
  ...
```

Ancora un altro bug corretto con questa versione.

Capitolo 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]