

Desarrollo web con Ruby on Rails 3

ActiveRecord: "callbacks", observadores y transacciones

Áncor González Sosa Imobach González Sosa

Banot.net
<http://www.banot.net/>

Octubre de 2010



Contenidos

1

Callbacks

- ¿Qué son?
- Definición
- Ejecución
- `around_callbacks`

2

Observadores

3

Transacciones



Contenidos

1

Callbacks

- ¿Qué son?
- Definición
- Ejecución
- `around_callbacks`

2

Observadores

3

Transacciones



¿Qué son los «callbacks»?

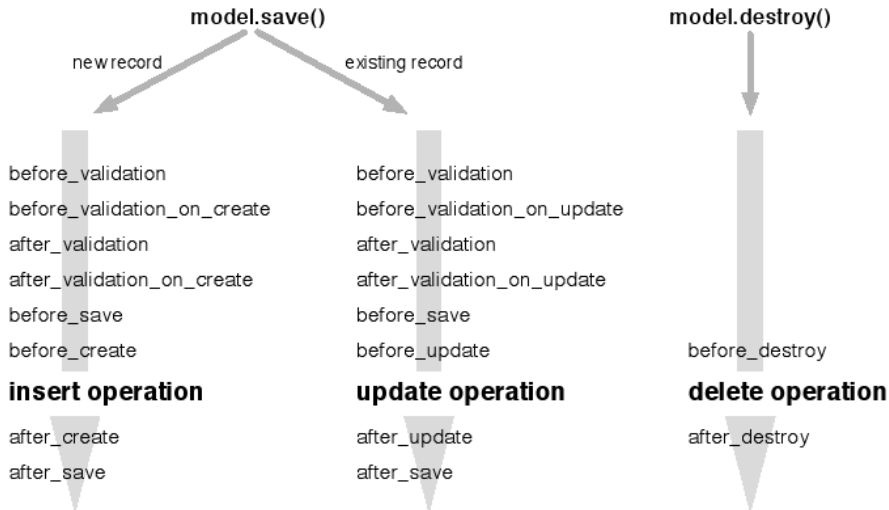
¿Qué son?

Mecanismo que permite ejecutar código en ciertos puntos del ciclo de vida de un objeto (creación, validación, destrucción...)

- Básicamente, cuando se crea, se guarda, se actualiza, se borra, se valida o se carga de la base de datos.
- ActiveRecord define 19 *callbacks*
- Las clases derivadas los heredan de sus padres



Secuencia



Los 19 callbacks de AR

- `after_initialize`
- `after_find`
- `after_touch`
- `before_validation` y `after_validation`
- `before_save`, `around_save` y `after_save`
- `before_create`, `around_create` y `after_create`
- `before_update`, `around_update` y `after_update`
- `before_destroy`, `around_destroy` y `after_destroy`
- `after_commit`
- `after_rollback`



Contenidos

1 Callbacks

- ¿Qué son?
- **Definición**
- Ejecución
- `around_callbacks`

2 Observadores

3 Transacciones



Definición de callbacks

- Redefiniendo métodos (`before_create`, `after_save`, etc.)
 - ▶ Esto tiene consecuencias en la herencia
- Estilo “macro”: definiendo métodos y registrándolos
 - ▶ Esta opción admite numerosas variantes



Redefinir los métodos

```
class User < ActiveRecord::Base

  protected
  def after_create
    logger.info 'New user created'
  end
end
```



Estilo “macro”

```
class User < ActiveRecord::Base
  before_save :encrypt_password

  private
  def encrypt_password
    # Cifrar la clave
  end
end
```



Estilo “macro” con bloques

- Si el método es muy pequeño, se puede redefinir pasando un bloque

```
class WikiPage < ActiveRecord::Base
  # Este efecto se consigue usando un campo updated_at
  before_save { |p| p.last_updated = Time.now }
end
```



Estilo “macros”: cadenas

- Ojo al uso de las comillas simples

```
class Topic < ActiveRecord::Base
  # Este efecto se consigue usando la opción :dependant
  before_destroy 'self.class.delete_all "parent_id = #{id}"'
end
```



Estilo “marcos”: objetos

```
class BankAccount < ActiveRecord::Base
  before_save      EncryptionWrapper.new("credit_card")
end

class EncryptionWrapper
  def initialize(attribute)
    @attribute = attribute
  end

  def before_save(record)
    # Cifra el atributo correspondiente
  end
end
```



Contenidos

1 Callbacks

- ¿Qué son?
- Definición
- **Ejecución**
- `around_callbacks`

2 Observadores

3 Transacciones



Orden de ejecución

- Si se define más de un «callback» para el mismo punto, se ejecutan en el mismo orden en que se definieron

```
class User < ActiveRecord::Base
  before_save :set_group           # 2
  before_validation :set_password_if_null # 1

  protected

  def before_save                  # 3
    . . .
  end
```



Callbacks condicionales

- Al registrar un “callback” pueden especificarse condiciones
- Como las validaciones, pueden usarse `:if` y `:unless` (incluso a la vez)

```
class < ActiveRecord::Base
  before_validation :set_default_password,
    :if => :is_password_blank?
end
```



Cancelación

- La ejecución se hace dentro de una transacción
- Si un “before callback” devuelve falso (no `nil`) o lanza una excepción, se hace un “rollback”
- Si un “after callback” lanza una excepción, también se hace un “rollback”



Contenidos

1 Callbacks

- ¿Qué son?
- Definición
- Ejecución
- **around_callbacks**

2 Observadores

3 Transacciones



around_callbacks

- Son un tipo particular de «callback»
- Usan el método `yield` para ejecutar la operación en curso

```
class User < ActiveRecord::Base

  protected

  def around_created
    logger.info "User will be created!"
    yield
    logger.info "User have been created!"
  end
```



Observadores

¿Qué son?

Son clases que «observan» todo el ciclo de vida de otros objetos e implementan «callbacks»

- Permite extraer y reutilizar código común
- También sirve para mantener más limpios los modelos en ciertos casos



Observadores

```
class CommentObserver < ActiveRecord::Observer
  def after_create(comment)
    Notifications.deliver_new_comment("bofh@domain",
                                     comment)
  end
end
```

- El nombre del modelo al que «observa» se infiere del nombre del observador
- Si hace falta, se puede usar el método «observe»



Observadores

```
class MyObserver < ActiveRecord::Observer
  observe Comment

  def after_create(comment)
    Notifications.deliver_new_comment("bofh@domain",
                                      comment)
  end
end
```

- El nombre del modelo al que «observa» se infiere del nombre del observador
- Si hace falta, se puede usar el método «observe»



Observando más de un modelo

```
class MyObserver < ActiveRecord::Observer
  observe Comment, Post

  def after_create(object)
    Notifications.deliver_new_comment("bofh@domain",
                                      object)
  end
end
```

- El nombre del modelo al que «observa» se infiere del nombre del observador
- Si hace falta, se puede usar el método «observe»



Transacciones

Transacciones a nivel de BBDD y a nivel de objetos

```
Account.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

```
Account.transaction(david, mary) do
  david.withdrawal(100)
  mary.deposit(100)
end
```



Ejercicio

- 1 Escribir un callback para asegurarse de que el estado de un partido recién creado sea 'u' (upcoming)
- 2 ¿Por qué el callback que has usado es el más apropiado?

