

Criando validates personalizados, exemplo com cpf e cnpj

Validates são muito fáceis de usar ! E gerar validates personalizados podem lhe poupar ainda mais tempo ! Neste artigo veremos como criar um validate para cpf e cnpj, e com isso, criar qualquer validate que desejar.

Criando um validate de brincadeira

Imagine que gostaria de validar se um determinado atributo é uma String ! Poderíamos criar um validate como no exemplo abaixo:

```
class Usuario < ActiveRecord::Base
  validates_string_of :nome, :message => 'mensagem customizada'
end
```

Nosso model

Ok, esse validate não existe ! Então vamos implementá-lo

```
module ActiveRecord::Validations::ClassMethods
  # Exemplo de um validate customizado
  def validates_string_of(*attr_names)
    message = "inválido"
    configuration = { :message => message, :on => :save }
    configuration.update(attr_names.extract_options!)
    #percorrendo nosso objeto para encontrar errors
    validates_each(attr_names, configuration) do |record, attr_name, value|
      #validando o atributo e adicionando em errors
      unless value.is_a?(String)
        record.errors.add(attr_name, configuration[:message])
      end
    end
  end
end
```

Implementando nosso validate de brincadeira

Neste momento, a linha que merece destaque é `value.is_a? String`, é neste momento que estamos efetuando a validação propriamente dita e adicionando a mensagem de erro.

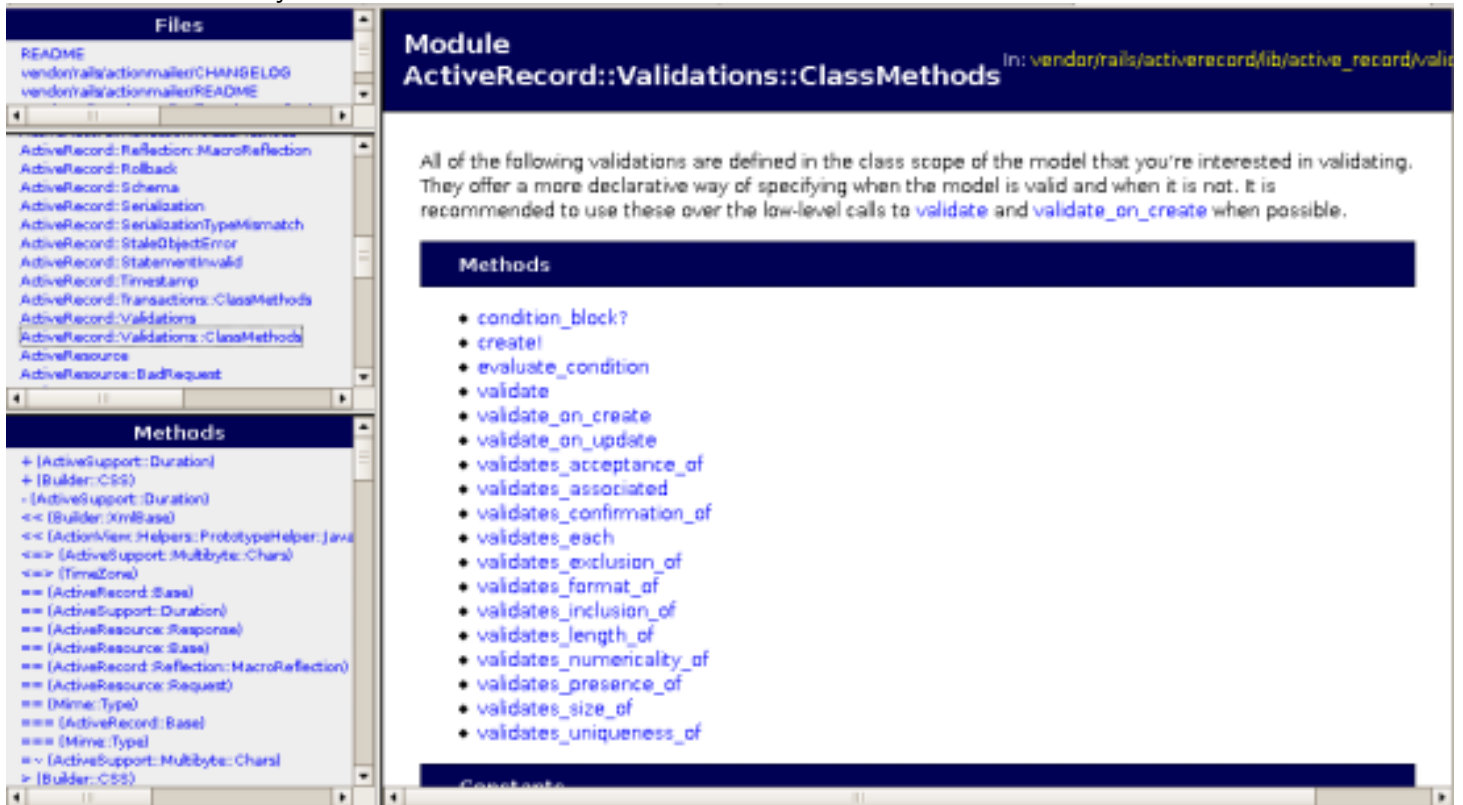
```
[ariveira@viper custom_validates]$ script/console
Loading development environment (Rails 2.0.2)
>> u = Usuario.new
=> #<Usuario id: nil, nome_completo: nil, nome_conhecido: nil, cpf: nil, rg: nil
, data_nascimento: nil, sexo: nil, naturalidade: nil, nacionalidade: nil, filiac
ao: nil, estado_civil_id: nil, profissao_id: nil, cargo_id: nil, arquivado: nil,
 usuario_cadastro_id: nil, usuario_alteracao_id: nil, created_at: nil, updated_a
t: nil, vendedor: nil, file_name: nil, content_type: nil, ext: nil, token: nil,
im: nil>
>> u.valid?
=> false
>> u.errors.full_messages.to_s
=> "Nome conhecido inválido"
>> █
```

Testando

No console do rails (vá até o diretório do projeto e digite `ruby script/console`) podemos simular uma validação para checarmos se tudo está correndo como previsto.

Primeiro instanciamos um objeto Usuario. Após, chamados o método valid? que processa a validação do objeto. Se você chamar os métodos save, por exemplo o método valid? é automaticamente chamado através dos callbacks previstos. Chamando valid? diretamente podemos verificar se existem erros no objeto.

Observe que o valor retornado é false, indicando que houve problemas na validação. Com o método errors.full_messages.to_s, estamos imprimindo uma mensagem de texto completa de todos os atributos com problemas. Onde, errors é um array contendo os erros, full_messages é o mesmo resultado de erros porém de forma completa, e por último to_s para converter um array para uma string a fim de facilitar a visualização.



Entendo como um validate funciona:

1) Os validates se encontram no módulo ActiveRecord::Validations::ClassMethods, você pode acessar a API do ruby on rails <http://api.rubyonrails.org> e verificar nessa classe os validates constantes lá. Como em Ruby as classes são abertas, podemos redefinir a classe ampliando suas funcionalidades, assim abrimos o próprio módulo citado acima e adicionamos nossos novos validates que nada mais são que métodos constantes no módulo.

2) A variável update = { :message => 'inválido'}, define os valores "default's" de nosso validate e configuration.update(attr_names.extract_options!) atualiza os valores que passamos customizados no model como por exemplo:

```
validates_string_of :nome_conhecido, :message => 'mensagem customizada'
```

3) Por último varremos uma iteração com validates_each onde:
record é o próprio objeto model
attr_name é o nome do atributo
value é o valor do atributo

Criando os modules CnpjHelper e CpfHelper

Agora iremos criar 2 módulos CnpjHelper e CpfHelper, onde cada um terá um método validate que retorna true/false se o valor passo seja válido ou não.

Estamos colocando eles em um módulo para que você possa aproveitar esses métodos em outros lugares do seu projeto, e ainda, outros métodos poderiam ser agregados como format que receberia um valor e adicionaria os "pontos" e "traços" etc.

```

module CpfHelper

  def self.validate(value)
    value = value.to_s
    value.gsub!(/[^\d-9]/, '')
    controle = ""
    digito = value.slice(-2, 2)
    if value.size == 11
      fator = 0
      2.times do |i|
        soma = 0
        9.times do |j|
          soma += value.slice(j, 1).to_i * (10 + i - j)
        end
        soma += (fator * 2) if i == 1
        fator = (soma * 10) % 11
        fator = 0 if (fator == 10)
        controle << fator.to_s
      end
    end
    ( controle == digito )
  end

end

```

CnpjHelper

```

module CnpjHelper

  def self.validate(value)
    value = value.to_s
    controle = ""
    digito = value.slice(-2, 2)
    value.gsub!(/[^\d-9]/, '')
    value = value.slice(1, value.size) if value.size == 15
    if ( value.size == 14 ) then
      fator = 0
      2.times do |i|
        soma = 0;
        12.times do |j|
          soma += value.slice(j, 1).to_i * ((11 + i - j) % 8 + 2)
        end
        soma += fator * 2 if i == 1
        fator = 11 - soma % 11;
        fator = 0 if fator > 9
        controle << fator.to_s
      end
    end
    ( controle == digito )
  end

end

```

CnpjHelper

```
module ActiveRecord::Validations::ClassMethods

  def validates_cnpj_of(*attr_names)
    configuration = { :message => "inválido", :on => :save }
    configuration.update(attr_names.extract_options!)
    #percorrendo nosso objeto para encontrar errors
    validates_each(attr_names, configuration) do |record, attr_name, value|
      #validando o atributo e adicionando em errors
      unless CnpjHelper.validate(value)
        record.errors.add(attr_name, configuration[:message])
      end
    end
  end

  def validates_cpf_of(*attr_names)
    configuration = { :message => "inválido", :on => :save }
    configuration.update(attr_names.extract_options!)
    #percorrendo nosso objeto para encontrar errors
    validates_each(attr_names, configuration) do |record, attr_name, value|
      #validando o atributo e adicionando em errors
      unless CpfHelper.validate(value)
        record.errors.add(attr_name, configuration[:message])
      end
    end
  end
end
```

Alterando nosso CustomValidates

```
class Usuario < ActiveRecord::Base
  validates_cpf_of :cpf
end
```

Alterando nosso model

```
[ariveira@viper custom_validates]$ script/console
```

```
Loading development environment (Rails 2.0.2)
```

```
>> u = Usuario.new
```

```
=> #<Usuario id: nil, nome_completo: nil, nome_conhecido: nil, cpf: nil,
, data_nascimento: nil, sexo: nil, naturalidade: nil, nacionalidade: nil
ao: nil, estado_civil_id: nil, profissao_id: nil, cargo_id: nil, arquivo
usuario_cadastro_id: nil, usuario_alteracao_id: nil, created_at: nil, u
t: nil, vendedor: nil, file_name: nil, content_type: nil, ext: nil, token
im: nil>
```

```
>> u.valid?
```

```
=> false
```

```
>> u.errors.full_messages.to_s
```

```
=> "Cpf inválido."
```

```
>> u.cpf = '111.111.111-11'
```

```
=> "111.111.111-11"
```

```
>> u.valid?
```

```
=> true
```

```
>> █
```

Testando no console

Verifique passamos como cpf '111.111.111-11'. Esse cpf é valido porque seu digito irá bater. Porém, se você quiser tornar inválido esse cpf basta colocar um código em seu CpfHpler.validate que contenha uma array de cpf's inválidos caso o valor passado como parâmetro conste no array automaticamente seria retornado false

Concluindo

Toda essa estrutura pode ser montada em um plugin, que você pode baixar http://www.objecttraining.com.br/custom_validates.zip Onde CnpjHelper e CpfHelper podem estar no diretorio lib. Já o módulo ActiveRecord::Validations::ClassMethods, devem estar no arquivo init.rb, pelo fato de redefinir uma classe, ou em outro arquivo, mas nesse caso o arquivo init.rb deve fazer um require desse arquivo.

Bom código,

Alexandre Riveira