Quick Reference

Ruby on Rails

초고속 웹 개발의 시작

O'REILLY®

한빛미디어 Hanbit Media, Inc.

Appendix B
빠른 참조

이 책의 목적은 독자들이 루비 온 레일즈를 빠르게 습득하여 실무에 적용할 수 있는 기반을 마련하는 것이다. 이 책에서는 레일즈의 핵심 부분들이 어떻게 동작하여, 간단한 웹 애플리케이션을 어떻게 만들어가는지 알아보았다. 레일즈는 이 책과 같은 입문서에서 다룰 수 있는 내용보다 훨씬 많은 기능을 지원한다. 이 부록은 레일즈가 지원하는 기능의 간단한 소개와 더 자세한 정보를 얻을 수 있는 링크를 제공한다.

이 참고 자료는 커트 힙스(Curt Hibbs)의 "What Is Ruby on Rails"★와 InVisible GmbHd의 "InVisible Ruby On Rails Reference 1.1.2"★★ 그리고 루비 온 레일즈 공식 API 문서(http://api.rubyonrails.com)에서 가져와서 정리한 것이다. 이 부록은 크리에이티브 커먼즈 라이센스(http://creativecommons.org/licenses/by-sa/2.5/)에 의해 배포할 수 있으며, http://www.hanbitbook.co.kr/exam/1453에서 내려받을 수 있다.

★ 각주 | "What Is Ruby on Rails"은 2005년 10월 ONLamp.com에 게재되었다. (http://www.onlamp.com/pub/a/onlamp/2005/10/13/what_is_rails.html)

★★ 각주 | "InVisible Ruby On Rails Reference 1.1.2"는 Creative Commons 라이센스에 의해 공개되었다. 원본은 http://blog.invisible.ch/files/rails-reference-1.1.html에서 볼 수 있다.

01_ 일반

문서

사용자 환경에 설치할 수 있는 API 문서

```
gem_server
```

http:// localhost:8088/

레일즈 공식 API

http://api.rubyonrails.com

검색 가능한 레일즈 API

http://rails.outertrack.com

http://railshelp.com

루비 문서

http://ruby-doc.org

여러 가지 API 문서

루비, 루비 온 레일즈, HTML, CSS, 자바스크립트, DOM 등 여러 가지를 망라하고 있다.

http://www.gotapi.com

지원하는 웹 서버

WEBrick

Mongrel

Lighttpd

Apache

MS IIS

자세한 내용은 http://wiki.rubyonrails.org/rails/pages/FAQ#webservers에서 찾아 볼 수 있다.

지원하는 데이터베이스

DB2

Firebird

MySQL

Oracle

PostgreSQL

SQLite

SQL Server

자세한 내용은 http://wiki.rubyonrails.org/rails/pages/DatabaseDrivers에서 찾아볼 수 있다.

통합 개발 환경

오픈 소스

Eclipse/RDT

http://rubyeclipse.sourceforge.net

FreeRIDE

http://freeride.rubyforge.org

RadRails(Eclipse/RDT 기반)

http://www.radrails.org

RDE(Ruby Development Environment)

http://homepage2.nifty.com/sakazuki/rde_e.html

상용

ArachnoRuby

http://www.ruby-ide.com/ruby/ruby_ide_and_ruby_editor.php

Komodo

http://www.activestate.com/Products/Komodo

편집기

여러 가지 에디터

http://wiki.rubyonrails.org/rails/pages/Editors

디버깅

로그 파일

development.log, test.log, production.log 등의 파일들을 찾아본다.

대화형 레일즈 콘솔(Interactive Rails Console)

http://wiki.rubyonrails.com/rails/pages/Console

http://www.clarkware.com/cgi/blosxom/2006/04/04

브레이크포인트

http://wiki.rubyonrails.com/rails/pages/HowtoDebugWithBreakpoint

디버거

통합 개발 환경(IDE) 참조

레일즈 디버그 팝업

http://www.bigbold.com/snippets/posts/show/697

새 레일즈 애플리케이션 만들기

```
rails app_name
```

옵션

-d=xxx or **--database=xxx**

사용할 데이터베이스를 지정한다(mysql, oracle, postgresql, sqlite3 등). 기본값은 mysql이다.

-r=xxx or **--ruby-path=xxx**

루비 경로를 지정한다. 지정하지 않으면 env를 이용하여 루비를 찾는다.

-f or **-freeze**

vendor/rails에 프레임워크를 복사한다.

02_ 테스트

```
rake test                       # 모든 유닛 및 기능 테스트를 한다.
rake test:functionals           # 기능 테스트를 한다.
rake test:integration           # 통합 테스트를 한다.
rake test:units                 # 단위 테스트를 한다.
```

유닛 테스트

```
rake test:units
```

사용할 수 있는 검증(assertion)

```
assert_kind_of Class, @var  # 같은 종류의 클래스
assert @var                 # nil이 아님
assert_equal 1, @p.id       # 동일함
@product.destroy
assert_raise(ActiveRecord::RecordNotFound) {  Product.find( @product.id ) }
```

기능 테스트

```
rake test:functionals
```

요청

```
get :action # 해당 액션에 대한 GET 요청
get :action, :id => 1,
        {  session_hash }, # 세션 변수
        {  flash_hash }     # 플래시에 저장할 텍스트 메시지

post :action, :foo => {  :value1 => 'abc', :value2 => '123' },
              {  :user_id => 17 },
              {  :message => 'success' }

get, post, put, delete, head
```

```
assert_response :success
# 사용 가능한 매개변수
#   :success
#   :redirect
#   :missing
#   :error
```

리다이렉트

```
assert_redirected_to :action => :other_action
assert_redirected_to :controller => 'foo', :action => 'bar'
assert_redirected_to http://www.invisible.ch
```

템플릿 렌더링

```
assert_template "post/index"
```

변수 할당

```
assert_nil assigns(:some_variable)
assert_not_nil assigns(:some_variable)
assert_equal 17, assigns(:posts).size
```

특정 태그 렌더링

```
assert_tag :tag => 'body'
assert_tag :content => 'Rails Seminar'
assert_tag :tag => 'div', :attributes => { :class => 'index_list' }
assert_tag :tag => 'head', :parent => { :tag => 'body' }
assert_tag :tag => 'html', :child => { :tag => 'head' }
assert_tag :tag => 'body', :descendant => { :tag => 'div' }
assert_tag :tag => 'ul',
           :children => { :count => 1..3,
                          :only => { :tag => 'li' } }
```

통합 테스트

```
rake test:integration
```

가상 통합 테스트

```
require "#{ File.dirname(__FILE__)} /../test_helper"
```

```
class UserManagementTest < ActionController::IntegrationTest
   fixtures :users, :preferences

   def test_register_new_user
       get "/login"
       assert_response :success
       assert_template "login/index"

       get "/register"
       assert_response :success
       assert_template "register/index"

       post "/register",
            :user_name => "happyjoe",
       :password => "neversad"
       assert_response :redirect
       follow_redirect!
       assert_response :success
       assert_template "welcome"
end
```

자세한 내용은 http://jamis.jamisbuck.org/articles/2006/03/09/integration-testing-in-rails-1-1에서 찾아볼 수 있다.

테스트에 대한 자세한 내용

자세한 내용은 http://manuals.rubyonrails.com/read/book/5에서 찾아볼 수 있다.

rake

rake는 루비 버전의 make 유틸리티이다. 레일즈는 rake로 다음과 같은 작업을 지원한다.

```
rake db:fixtures:load        # 현재 환경의 데이터베이스에 픽스쳐를 불러들인다.
                             # 특정 픽스쳐는 FIXTURES=x,y와 같이 불러들인다.
rake db:migrate              # db/migrate에 있는 스크립트로 데이터베이스를
                             # 마이그레이션한다. 특정 버전으로의 마이그레이션은
                             # VERSION=x로 설정한다.
rake db:schema:dump          # db/schema.rb 파일을 생성한다. 액티브 레코드가
                             # 지원하는 DB에 사용할 수 있다.
rake db:schema:load          # schema.rb 파일을 데이터베이스에 불러들인다.
```

```
rake db:sessions:clear       # 세션 테이블을 초기화한다.
rake db:sessions:create      # CGI::Session::ActiveRecordStore에 사용할
                             # 세션 테이블을 생성한다.
rake db:structure:dump       # 데이터베이스 구조를 SQL 파일로 출력한다.
rake db:test:clone           # 현재 환경의 데이터베이스 스키마로
                             # 테스트 데이터베이스를 재구성한다.
rake db:test:clone_structure # 테스트 데이터베이스를 개발 환경 구조와
                             # 동일하게 재구성한다.
rake db:test:prepare         # 테스트 데이터베이스를 준비하고, 스키마를 불러온다.
rake db:test:purge           # 테스트 데이터베이스를 초기화한다.

rake doc:app                 # 애플리케이션 HTML 파일들을 만든다.
rake doc:clobber_app         # rdoc 결과물을 모두 삭제한다.
rake doc:clobber_plugins     # 플러그인 문서를 삭제한다.
rake doc:clobber_rails       # rdoc 결과물을 모두 삭제한다.
rake doc:plugins             # 설치된 모든 플러그인에 관한 문서를 생성한다.
rake doc:rails               # 레일즈 HTML 파일을 생성한다.
rake doc:reapp               # RDOC 파일를 강제로 재생성한다.
rake doc:rerails             # RDOC 파일를 강제로 재생성한다.

rake log:clear               # log/ 디렉토리에 있는 모든 *.log 파일들을
                             # 0바이트로 초기화한다.

rake rails:freeze:edge       # 최신 Edge Rails에 현재 애플리케이션을 저장한다.
                             # 특정 버전은 REVISION=x로 지정한다.
rake rails:freeze:gems       # 현재 gems에 현재 애플리케이션을 저장한다.
                             # (vendor/rails에 저장된다)
rake rails:unfreeze          # gems나 edge에서 현재 애플리케이션을 가져와서
                             # 애플리케이션을 실행한다.

rake rails:update            # 레일즈에서 스크립트와 public/javascripts를 갱신한다.
rake rails:update:javascripts # 현재 설치된 레일즈를 이용하여 javascripts를 갱신한다.
rake rails:update:scripts    # 새로운 스크립트를 애플리케이션의 script 디렉토리에 추가한다.
rake stats                   # 애플리케이션의 코드 통계(KLOC 등)를 낸다.

rake test                    # 모든 단위 및 기능 테스트를 실행한다.
rake test:functionals        # functionalsdb:test:prepare 테스트를 실행한다.
rake test:integration        # integrationdb:test:prepare 테스트를 실행한다.
rake test:plugins            # pluginsenvironment 테스트를 실행한다.
rake test:recent             # recentdb:test:prepare 테스트를 실행한다.
rake test:uncommitted        # uncommitteddb:test:prepare 테스트를 실행한다.
rake test:units              # unitsdb:test:prepare 테스트를 실행한다.
```

```
rake tmp:cache:clear      # tmp/cache에 있는 모든 파일과 디렉토리를 삭제한다.
rake tmp:clear            # tmp/에 있는 모든 세션, 캐시 및 소켓 파일들을 삭제한다.
rake tmp:create           # 세션, 캐시, 소켓을 저장할 tmp 디렉토리를 만든다.
rake tmp:sessions:clear   # tmp/sessions에 있는 모든 파일을 삭제한다.
rake tmp:sockets:clear    # tmp/sessions에 있는 모든 ruby_sess.* 파일을 삭제한다.
```

스크립트

```
script/about                        # 환경에 대한 정보
script/breakpointer                 # 브레이크포인트 서버를 실행한다.
script/console                      # 레일즈 콘솔
script/destroy                      # 제너레이터로 생성한 파일들을 삭제한다.
script/generate                     # -> 제너레이터
script/plugin                       # -> 플러그인
script/runner                       # 레일즈 컨텍스트에서 작업을 실행한다.
script/server                       # 개발 서버를 실행한다.
                                    # http://localhost:3000

script/performance/profiler         # 연산량이 많은 메소드를 프로파일링한다.
script/performance/benchmarker      # 여러 메소드를 벤치마킹한다.

script/process/reaper
script/process/spawner
```

제너레이터

```
ruby script/generate model ModellName
ruby script/generate controller ListController show edit
ruby script/generate scaffold ModelName ControllerName
ruby script/generate migration AddNewTable
ruby script/generate plugin PluginName
ruby script/generate mailer Notification lost_password signup
ruby script/generate web_service ServiceName api_one api_two
ruby script/generate integration_test TestName
ruby script/generate session_migration
```

옵션

`-p or --pretend`

실제 생성/수정은 하지 않고, 실행만 한다.

`-f or --force`

기존 파일을 덮어 쓴다.

`-s or --skip`

존재하는 파일들은 건너뛴다.

`-q or --quiet`

표준 출력을 막는다.

`-t or --backtrace`

디버깅: 오류에 대한 추적 정보를 보여준다.

`-h or --help`

도움말을 보여준다.

`-c or --svn`

subversion과 연동하여 파일을 수정한다. (참고: svn은 경로에 있어야 한다.)

Plug-ins

```
script/plugin discover           # 플러그인 저장소를 찾는다.
script/plugin list               # 사용 가능한 모든 플러그인을 보여준다.
script/plugin install where      # "where" 플러그인을 설치한다.
script/plugin install -x where   # where 플러그인을 SVN external로 설치한다.
script/plugin install http://invisible.ch/projects/plugins/where
script/plugin update             # 설치된 플러그인을 갱신한다.
script/plugin source             # 소스 저장소를 추가한다.
script/plugin unsource           # 소스 저장소를 삭제한다.
script/plugin sources            # 소스 저장소 목록을 보여준다.
```

자세한 내용은 http://wiki.rubyonrails.com/rails/pages/Plugins에서 찾아볼 수 있다.

플러그인 검색: http://www.agilewebdevelopment.com/plugins.

03_ RJS(루비 자바스크립트)

예를 들어보자.

```
update_page do |page|
     page.insert_html :bottom, 'list', "<li>#{ @item.name} </li>"
```

```
        page.visual_effect :highlight, 'list'
      page.hide 'status-indicator', 'cancel-link'
    end
```

다음과 같은 자바스크립트를 생성해낸다.

```
new Insertion.Bottom("list", "<li>Some item</li>");
new Effect.Highlight("list");
["status-indicator", "cancel-link"].each(Element.hide);
```

자세한 내용은 다음 문서들을 참고한다.

- http://api.rubyonrails.com/classes/ActionView/Helpers/PrototypeHelper/JavaScriptGenerator/GeneratorMethods.html
- http://www.codyfauser.com/articles/2005/11/20/rails-rjs-templates
- http://scottraymond.net/articles/2005/12/01/real-world-rails-rjs-templates
- http://www.rubynoob.com/articles/2006/05/13/simple-rails-rjs-tutorial

04_ 액티브 레코드

자동으로 대응

- Tables → classes
- Rows → objects(instances of model classes)
- Columns → object attributes

테이블의 클래스 대응은 영어의 복수형을 따른다.

- Invoice 모델 클래스는 invoices 테이블에 대응된다.
- Person 모델 클래스는 people 테이블에 대응된다.
- Country 모델 클래스는 countries 테이블에 대응된다.
- SecurityLevel 모델 클래스는 security_levels 테이블에 대응된다.

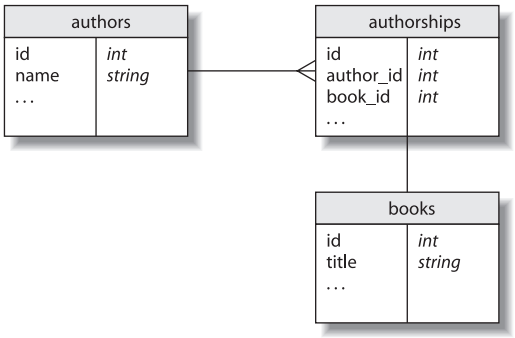
자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Base.html을 참고한다.

연관

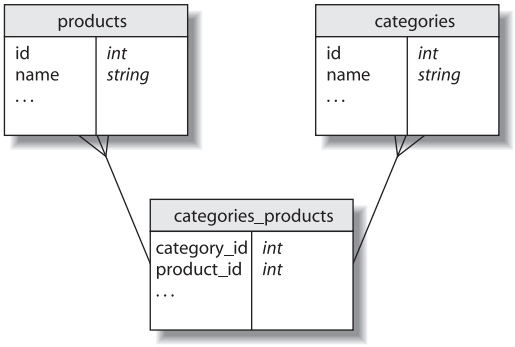
4가지 연관 모델([그림 B-1]과 [그림 B-2])

```
has_one
has_many
belongs_to
has_and_belongs_to_many
def Order < ActiveRecord::Base
  has_many :line_items
  belongs_to :customer      # 데이터베이스 테이블에 "customer_id" 컬럼이 있다.
end

def LineItem < ActiveRecord::Base
  belongs_to :order         # 데이터베이스 테이블에 "order_id" 컬럼이 있다.
end

def Customer < ActiveRecord::Base
  has_many :orders
  has_one :address
end

def Address < ActiveRecord::Base
  belongs_to :customer
end

belongs_to  :some_model,
        :class_name  => 'MyClass',      # 다른 클래스 이름을 지정한다.
        :foreign_key => 'my_real_id',   # 다른 기본 키를 설정한다.
        :conditions  => 'column = 0'    # 이 조건을 만족할 경우에 위에서 설정한
                                        # 두 가지 설정을 사용한다.
has_one :some_model,
                                        # belongs_to에 대응하여:
        :dependent   => :destroy        # 연관된 객체들을 삭제한다.
        :order       => 'name ASC'      # 정렬을 위한 SQL 명령

has_many :some_model
                                        # has_one에 대응하여:
```

```
        :dependent => :destroy          # 각 객체의 destroy를 호출하여
                                        # 모든 연관된 데이터를 삭제한다.
        :dependent => :delete_all       # destroy 메소드를 호출하지 않고
                                        # 모든 연관된 데이터를 삭제한다.
        :dependent => :nullify          # 연관 객체들을 삭제하지 않고,
                                        # 연관을 무효화시킨다.
        :group => 'name'                # GROUP BY를 추가한다.
        :finder_sql => 'select ....'    # 레일즈 파인더 대신 사용
        :counter_sql => 'select ...'    # 레일즈 카운터 대신 사용
def Category < ActiveRecord::Base
  has_and_belongs_to_many :products
end
def Product < ActiveRecord::Base
  has_and_belongs_to_many :categories
end
```



[그림 B-1] 1:1 및 1:N 관계

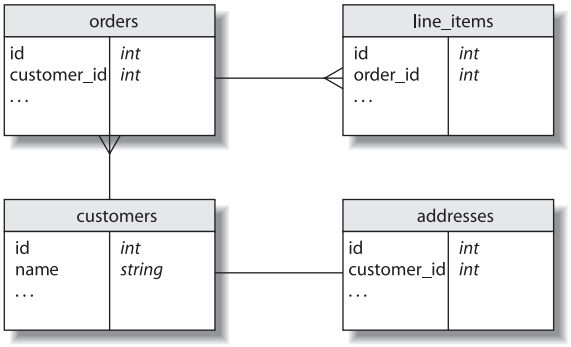


[그림 B-2] M:N 관계

categories_products 테이블

- category_id 컬럼이 있다.
- product_id 컬럼이 있다.
- id 컬럼이 없다.

연관 결합 모델([그림 B-3])



[그림 B-3] 쓰루 모델(Through model)

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

class Authorship < ActiveRecord::Base
  belongs_to :author
  belongs_to :book
end

class Book < ActiveRecord::Base
  has_one :authorship
end

@author = Author.find :first
@author.authorships.collect { |a| a.book }        # 저자가 쓴 모든 책을 가져온다.
```

```
@author.books                                          # Authorship 결합 모델을
                                                       # 사용하여 모든 책을 가져온다.
```

has_many 관계로도 가능하다.

```
class Firm < ActiveRecord::Base
  has_many   :clients
  has_many   :invoices, :through => :clients
  has_many   :paid_invoices, :through => :clients, :source => :invoice
end

class Client < ActiveRecord::Base
  belongs_to :firm
  has_many   :invoices
end

class Invoice < ActiveRecord::Base
  belongs_to :client
end

@firm = Firm.find :first
@firm.clients.collect { |c| c.invoices }.flatten     # 회사의 모든 고객에 대한
                                                     # 송장을 가져온다.
@firm.invoices       # Client 결합 모델을 이용하여 모든 송장을 가져온다.
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Associations/ClassMethods.html에서 찾아본다.

검증

```
validates_presence_of :firstname, :lastname  # 값이 꼭 존재해야 한다.
validates_length_of :password,
                    :minimum => 8            # 8자 이상
                    :maximum => 16           # 16자 이하
                    :in => 8..16             # 8자에서 16자 사이
                    :too_short => 'way too short'
                    :too_long => 'way to long'

validates_acceptance_of :eula                # 약관에 동의해야 한다.
                        :accept => 'Y'       # 기본값: 1 (체크 박스에 적합)
```

```
validates_confirmation_of :password
# 비밀번호 필드의 값과 password_confirmation의 값은 서로 일치해야 한다.

validates_uniqueness_of :user_name           # user_name은 유일해야 한다.
                        :scope => 'account_id'       # 조건:
                                             # account_id = user.account _id

validates_format_of :email                   # 필드는 정규식을 만족해야 한다.
                    :with => /^(+)@((?:[ -a-z0-9] +/.)+[ a-z]{ 2,} )$/i

validates_numericality_of   :value           # 값은 숫자이다.
                            :only_integer => true
                            :allow_nil => true

validates_inclusion_in  :gender,             # 값은 열거형이다.
                        :in => %w( m, f )

validates_exclusion_of  :age                 # 값은 열거형이 아니다.
                        :in => 13..19        # 10대 청소년은 대상이 아니다.

validates_associated :relation
# 연관된 객체가 유효한지 검증한다.
```

검증 옵션

```
:message => 'my own errormessage'
:on      => :create                  # 또는 :update (검증만 한다.)
:if      => ...                      # oder Proc 메소드를 호출한다.
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Validations.html을 참고하기 바란다.

계산

```
Person.average :age
Person.minimum :age
Person.maximum :age
Person.count
Person.count(:conditions => "age > 26")
Person.sum :salary, :group => :last_name
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Calculations/ClassMethods.html을 참고하기 바란다.

파인더(Finders)

```
find(42)                                          # id가 42인 객체
find([ 37, 42] )                                  # id가 37, 42인 객체들
find :all
find :first,
     :conditions => [  "name = ?", "Hans" ]       # 조건에 부합하는 첫번째 레코드를
                                                  # 검색한다.
```

기타 옵션

```
:order => 'name DESC'               # 정렬을 위한 SQL 명령
:offset => 20                       # 20번째 항목부터 시작한다.
:limit => 10                        # 10개의 객체만 반환한다.
:group => 'name'                    # 'GROUP BY' SQL 명령
:joins => 'LEFT JOIN ...'           # LEFT JOIN 한다. (자주 쓰이지 않는다)
:include => [ :account, :friends]   # LEFT OUTER JOIN 한다.
:include => {  :groups => {  :members=> {  :favorites } } }
:select => [ :name, :adress]        # SELECT * FROM 대신 사용한다.
:readonly => true                   # 읽기 전용 객체
```

동적 속성 파인더

```
Person.find_by_user_name(user_name)
Person.find_all_by_last_name(last_name)
Person.find_by_user_name_and_password(user_name, password)
Order.find_by_name("Joe Blow")
Order.find_by_email("jb@gmail.com")
Slideshow.find_or_create_by_name("Winter")
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Base.html을 찾아보기 바란다.

범위(Scope)

```
Employee.with_scope(
    :find => {  :conditions => "salary > 10000",
                :limit => 10 } ) do
```

```
  Employee.find(:all)           # => SELECT * FROM employees
                                #             WHERE (salary > 10000)
                                #              LIMIT 10

  # 범위는 누적된다.
  Employee.with_scope(
    :find => { :conditions => "name = 'Jamis'" }) do
    Employee.find(:all)         # => SELECT * FROM employees
                                #            WHERE ( salary > 10000 )
                                #             AND ( name = 'Jamis' ))
                                #              LIMIT 10
  end

  # 이전 범위는 포함하지 않는다.
  Employee.with_exclusive_scope(
    :find => { :conditions => "name = 'Jamis'" }) do
    Employee.find(:all)   # => SELECT * FROM employees
                          #             WHERE (name = 'Jamis')
  end
end
```

자세한 내용은 다음 사이트를 참고하기 바란다.

- http://www.codyfauser.com/articles/2006/02/01/using-with_scope-to-refactor-messy-finders
- http://blog.caboo.se/articles/2006/02/22/nested-with_scope

Acts

acts_as_list

```
class TodoList < ActiveRecord::Base
    has_many :todo_items, :order => "position"
  end

  class TodoItem < ActiveRecord::Base
    belongs_to :todo_list
    acts_as_list :scope => :todo_list
  end
```

```
todo_list.first.move_to_bottom
todo_list.last.move_higher
```

자세한 내용은 다음 사이트를 참고하기 바란다.

- http://api.rubyonrails.com/classes/ActiveRecord/Acts/List/ClassMethods.html
- http://api.rubyonrails.com/classes/ActiveRecord/Acts/List/InstanceMethods. html

acts_as_tree

```
class Category < ActiveRecord::Base
    acts_as_tree :order => "name"
  end

  Example :
  root
   /_ child1
        /_ subchild1
        /_ subchild2

  root      = Category.create("name" => "root")
  child1    = root.children.create("name" => "child1")
  subchild1 = child1.children.create("name" => "subchild1")

  root.parent   # => nil
  child1.parent # => root
  root.children # => [child1]
  root.children.first.children.first # => subchild1
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Acts/Tree/ClassMethods.html을 참고하기 바란다.

콜백

콜백은 액티브 레코드 객체가 생성되어 소멸되기까지의 생명 주기 동안, 객체의 상태 변경 전후로 취할 작업을 설정할 수 있는 기능을 제공한다. ([표 B-1])

[표 B-1] 액티브 레코드 객체의 생명 주기

객체 상태	콜백
save	
valid?	
	before_validation
	before_validation_on_create
validate	
validate_on_create	
	after_validation
	after_validation_on_create
	before_save
	before_create
create	
	after_create
	after_save

예

```
class Subscription < ActiveRecord::Base
  before_create :record_signup
private
  def record_signup
    self.signed_up_on = Date.today
  end
end

class Firm < ActiveRecord::Base
  # 회사가 없어지면 회사와 관련된 고객과 직원들을 모두 삭제한다.
  before_destroy { |record| Person.destroy_all "firm_id = #{record.id}" }
  before_destroy { |record| Client.destroy_all "client_of = #{record.id}" }
end
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Callbacks.html을 참고하기 바란다.

관찰자(Observers)

Observer 클래스는 원본 클래스에 콜백을 추가하지 않고, 콜백의 기능을 대신할 수 있는 환경을 제공한다.

```
class CommentObserver < ActiveRecord::Observer
  def after_save(comment)
    Notifications.deliver_comment("admin@do.com", "New comment was posted", comment)
  end
end
```

- 관찰자는 app/model/model_observer.rb에 저장한다.
- 관찰자를 활성화시키려면 config/environment.rb 파일에 다음과 같이 한다.

```
config.active_record.observers = :comment_observer, :signup_observer
```

자세한 내용은 http://api.rubyonrails.com/classes/ActiveRecord/Observer.html을 참고하기 바란다.

마이그레이션

```
> ruby script/generate migration MyAddTables
```

db/migrations/001_my_add_tables.rb 파일이 생성된다. up()과 down() 메소드가 데이터베이스 스키마를 변경한다.

```
def self.up     # 데이터베이스 스키마를 다음 버전으로 올린다.
  create_table :table, :force => true do |t|
    t.column :name, :string
    t.column :age, :integer, {  :default => 42 }
    t.column :description, :text
     # :string, :text, :integer, :float, :datetime, :timestamp, :time,
     # :date, :binary, :boolean
  end
  add_column :table, :column, :type
  rename_column :table, :old_name, :new_name
  change_column :table, :column, :new_type
  execute "SQL Statement"
  add_index :table, :column, :unique => true, :name => 'some_name'
  add_index :table, [  :column1, :column2 ]
```

```
end

def self.down   # 이전 버전으로 되돌린다.
  rename_column :table, :new_name, :old_name
  remove_column :table, :column
  drop_table :table
  remove_index :table, :column
end
```

마이레이션 실행

```
> rake db:migrate
> rake db:migrate VERSION=14
> rake db:migrate RAILS_ENV=production
```

자세한 내용은 다음 사이트를 참고하기 바란다.

- http://api.rubyonrails.org/classes/ActiveRecord/Migration.html
- http://glu.ttono.us/articles/2005/10/27/the-joy-of-migrations
- http://jamis.jamisbuck.org/articles/2005/09/27/getting-started-with-activerecord-migrations

05_ 컨트롤러

컨트롤러 메소드

컨트롤러의 public 메소드들은 /controller/action과 같은 형태의 URL 형식(예, /world/hello)으로 호출할 수 있다.

```
class WorldController < ApplicationController
def hello
  render :text => 'Hello world'
end
```

GET이나 POST에 상관없이, 모든 요청 매개변수는 params 해시에 담겨진다.

```
/world/hello/1?foo=bar
id = params[ :id]      # 1
foo = params[ :foo]    # bar
```

컨트롤러 메소드에서 정의한 인스턴스 변수들은 해당하는 뷰 템플릿에서 사용할 수 있다.

```
def show
  @person = Person.find( params[ :id] )
end
```

응답으로 보낼 수 있는 형식을 결정한다.

```
def index
  @posts = Post.find :all

  respond_to do |type|
    type.html # 기본값 weblog/index.rhtml을 렌더링한다.
    type.xml  { render :action => "index.rxml" }
    type.js   { render :action => "index.rjs" }
  end
end
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionController/Base.html을 참고하기 바란다.

렌더

일반적으로 컨트롤러 메소드와 같은 이름의 뷰 템플릿으로 결과를 렌더링한다.

액션

```
render :action => 'some_action'     # 기본값. 컨트롤러 메소드 "some_ action"에서
                                    # 따로 지정할 필요는 없다.
render :action => 'another_action', :layout => false
render :action => 'some_action', :layout => 'another_layout'
```

부분 뷰

부분 뷰는 _로 시작하는 이름의 파일에 저장된다. (_error, _subform, _listitem)

```
render :partial => 'subform'
render :partial => 'error', :status => 500
render :partial => 'subform', :locals => { :variable => @other_variable }
```

```
render :partial => 'listitem', :collection => @list
render :partial => 'listitem', :collection => @list, :spacer_template => 'list_divider'
```

템플릿

액션을 렌더링하는 것과 유사하며, 템플릿 루트 디렉토리(app/views)를 기준으로 템플릿을 찾는다.

```
render :template => 'weblog/show'  # app/views/weblog/show를 렌더링한다.
```

파일(Files)

```
render :file => '/path/to/some/file.rhtml'
render :file => '/path/to/some/filenotfound.rhtml', status => 404, :layout => true
```

텍스트

```
render :text => "Hello World"
render :text => "This is an error", :status => 500
render :text => "Let's use a layout", :layout => true
render :text => 'Specific layout', :layout => 'special'
```

내장 템플릿

ERb로 소형 템플릿을 렌더링한다.

```
render :inline => "<%= 'hello , ' * 3 + 'again' %>"
render :inline => "<%= 'hello ' + name %>", :locals => { :name => "david" }
```

RJS

```
def refresh
  render :update do |page|
    page.replace_html  'user_list', :partial => 'user', :collection => @users
    page.visual_effect :highlight, 'user_list'
  end
end
```

content_type 바꾸기

```
render :action => "atom.rxml", :content_type => "application/atom+xml"
```

리다이렉트

```
redirect_to(:action => "edit")
redirect_to(:controller => "accounts", :action => "signup")
```

무시

```
render :nothing
render :nothing, :status => 403    # 거부
```

자세한 내용은 다음 사이트를 참고하기 바란다.

- http://api.rubyonrails.com/classes/ActionView/Base.html
- http://api.rubyonrails.com/classes/ActionController/Base.html

URL 라우팅

config/routes.rb 파일

```
map.connect '', :controller => 'posts', :action => 'list' # 기본값
map.connect ':action/:controller/:id'
map.connect 'tasks/:year/:month', :controller => 'tasks',
                                  :action => 'by_date',
                                  :month => nil, :year => nil,
                                  :requirements => { :year => /\d{4}/,
                                                     :month => /\d{1,2}/ }
```

자세한 내용은 http://manuals.rubyonrails.com/read/chapter/65를 참고하기 바란다.

필터

컨트롤러가 요청을 처리하기 전 또는 후에 필터를 이용하여 요청을 변경할 수 있다. 예를 들어, 필터를 이용하여 인증, 암호화, 압축 등에 사용할 수 있다.

```
before_filter :login_required, :except => [  :login ]
before_filter :autenticate, :only => [  :edit, :delete ]
after_filter :compress
```

간단한 필터는 proc을 사용할 수도 있다.

```
before_filter { |controller| false if controller.params["stop_action"] }
```

필터들의 순서는 prepend_before_filter와 prepend_after_filter로 정할 수 있다.
(prepend_before_filter :some_filter는 some_filter를 통과할 필터들의 맨 앞에 삽입한다)
부모 클래스에서 필터를 정의하고, 자식 클래스에서 이를 무시할 수 있다.

```
skip_before_filter :some_filter
skip_after_filter :some_filter
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionController/Filters/ClassMethods.html을 참고하기 바란다.

세션(session)/플래시(flash)

여러 요청 간에 데이터를 공유할 때는 세션이나 플래시 해시를 이용한다. 플래시는 다음 요청 때까지만 값(주로 문자열)을 저장하고 있고, 세션은 전체 세션 동안 값을 유지한다.

```
session[ :user] = @user
flash[ :message] = "Data was saved successfully"

<%= link_to "login", :action => 'login' unless session[ :user] %>
<% if flash[ :message] %>
<div><%= h flash[ :message] %></div>
<% end %>
```

세션 관리

세션 관리를 끌 수 있다.

```
session :off                          # 세션 관리를 끈다.
session :off, :only => :action        # 이 :action에 대해서만 끈다.
session :off, :except => :action      # 이 :action만 제외하고 끈다.
session :only => :foo,                # HTTPS를 사용할때 :foo에 대해서만 켠다.
        :session_secure => true
```

```
session :off, :only => :foo,          # 웹 서비스를 사용할 때 foo에 대해서만 끈다.
        :if => Proc.new { |req| req.parameters[ :ws] }
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionController/SessionManagement/ClassMethods.html을 참고하기 바란다.

쿠키

설정

```
cookies[ :user_name] = "david"          # => 간단한 세션 쿠키를 설정한다.
cookies[ :login] = { :value => "XJ-122", :expires => Time.now + 3600}
                                        # => 1시간 후 만료되는 쿠키를 설정한다.
```

읽기

```
cookies[ :user_name] # => "david"
cookies.size               # => 2
```

삭제

```
cookies.delete :user_name
```

쿠키 설정에 사용하는 옵션

value

쿠키의 값이나 배열

path

쿠키를 참조하는 경로(기본적으로 애플리케이션의 루트이다)

domain

쿠키를 참조하는 도메인

expires

쿠키가 만기되는 시간(Time 객체)

secure

보안 쿠키 여부 결정. 기본적으로 비보안 쿠키이며, 보안 쿠키는 HTTPS 서버로만 전송된다.

자세한 내용은 http://api.rubyonrails.com/classes/ActionController/Cookies.html 을 참고하기 바란다.

06_ 뷰

뷰 템플릿

모든 뷰 템플릿은 app/views 아래 컨트롤러 이름과 같은 디렉토리에 저장된다. 확장자는 템플릿의 종류를 결정짓는다.

*.rhtml
: 루비 HTML(ERB)

*.rxml
: 루비 XML (Builder)

*.rjs
: 루비 자바스크립트

컨트롤러의 모든 인스턴스 변수는 뷰에서 사용 가능하다. 또한 다음과 같은 특수 객체도 사용할 수 있다.

headers
: 응답으로 나가는 헤더

request
: 들어온 요청에 대한 객체

response
: 응답으로 나가는 객체

params
: 요청 매개변수 해시

session
: 세션 해시

controller

현재 컨트롤러

RHTML

RHTML은 HTML과 루비를 태그로 섞어 놓은 것이다. 모든 루비 프로그래밍에서 이용할 수 있다.

```
<% %>    # 루비 코드를 실행한다.
<%= %>   # 루비 코드를 실행하고 결과를 표시한다.

<ul>
<% @products.each do |p| %>
  <li><%= h @p.name %></li>
<% end %>
</ul>
```

〈%= %〉 태그 사이의 루비 코드 실행 결과는 그대로 HTML 페이지로 출력된다. 이때 의도하지 않은 HTML 코드가 출력되는 것을 방지하려면 h( ) 함수를 이용하여 HTML 이스케이프한다. 예를 들면 다음과 같다.

```
<%=h @user_entered_notes %>
```

RXML

XML 파일을 만든다.

```
xml.instruct!                # <?xml version="1.0" encoding= "UTF-8"?>
xml.comment! "a comment"     # <!-- a comment -->
xml.feed "xmlns" => "http://www.w3.org/2005/Atom" do
  xml.title "My Atom Feed"
  xml.subtitle h(@feed.subtitle), "type" => 'html'
  xml.link url_for( :only_path => false,
                    :controller => 'feed',
                    :action => 'atom' )
  xml.updated @updated.iso8601
  xml.author do
    xml.name "Jens-Christian Fischer"
    xml.email "jcfischer@gmail.com"
  end
  @entries.each do |entry|
```

```
    xml.entry do
      xml.title entry.title
      xml.link "href" => url_for ( :only_path => false,
                                   :controller => 'entries',
                                   :action => 'show',
                                   :id => entry )
      xml.id entry.urn
      xml.updated entry.updated.iso8601
      xml.summary h(entry.summary)
    end
  end
end
```

자세한 내용은 http://rubyforge.org/projects/builder/를 참고하기 바란다.

RJS

HTML과 XML 템플릿 이 외에도, 레일즈는 자바스크립트 템플릿도 사용할 수 있다. 이를 이용하여 페이지의 복잡한 기능을 손쉽게 수정할 수 있다. 다음과 같은 메소드를 이용하여 페이지 항목들을 조정할 수 있다.

select

DOM 항목을 선택한다.

```
page.select('pattern'    # CSS pattern을 통해 페이지에서 필요한 요소를 선택한다.
                         # select('p'), select('p.welcome b')
page.select('div.header em').first.hide
page.select('#items li').eacj do |value|
  value.hide
end
```

insert_html

특정 위치의 DOM에 내용을 삽입한다.

```
page.insert_html :position, id, content
```

위치는 다음 중 하나이다.

```
:top
:bottom
:before
:after
```

replace_html

특정 DOM 항목의 inner HTML을 바꾼다.

```
page.replace_html 'title', "This is the new title"
page.replace_html 'person-45', :partial => 'person', :object => @person
```

replace

특정 DOM 항목의 전체 HTML(항목 자체)을 바꾼다.

```
page.replace 'task', :partial => 'task', :object => @task
```

remove

특정 DOM 항목을 삭제한다.

```
page.remove 'edit-button'
```

hide

특정 DOM 항목을 숨긴다.

```
page.hide 'some-element'
```

show

특정 DOM 항목을 보이게 한다.

```
page.show 'some-element'
```

toggle

DOM 항목을 숨기거나 보이게 한다.

```
page.toggle 'some-element'
```

alert

경고창을 띄운다.

```
page.alert 'Hello world'
```

redirect_to

브라우저를 지정한 위치로 리다이렉트한다.

```
page.redirect_to :controller => 'blog', :action => 'show', :id => @post
```

call

다른 자바스크립트 함수를 호출한다.

```
page.call foo, 1, 2
```

assign

자바스크립트 변수에 값을 할당한다.

```
page.assign "foo", 42
```

<<

페이지에 직접 자바스크립트를 삽입한다.

```
page << "alert('hello world);"
```

delay

블록에 있는 코드에서 지정한 시간(초 단위)동안 실행을 멈춘다.

```
page.delay(10) do
   page.visual_effect :fade, 'notice'
end
```

visual_effect

Scriptaculous 효과를 호출한다.

```
page.visual_effect :highlight, 'notice', :duration => 2
```

sortable

정렬 가능한 항목을 생성한다.

```
page.sortable 'my_list', :url => { :action => 'order' }
```

dragable

드래그 가능한 항목을 생성한다.

```
page.dragable 'my_image', :revert => true
```

drop_receiving

드롭할 수 있는 항목을 생성한다.

```
page.drop_recieving 'my_cart', :url => { :controller => 'cart',
                                         :action => 'add' }
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionView/Base.html을 참고하기 바란다.

보조 함수

주로 데이터를 표시할 때 사용하는 작은 함수들은 보조 함수로 추출할 수 있다. 그리고 각 뷰는 app/helpers 디렉토리에 해당하는 보조 클래스를 갖고 있다. 공통 기능을 하는 보조 함수는 app/helpers/application_helper.rb에 저장한다.

링크

```
link_to "Name", :controller => 'post', :action => 'show', :id => @post.id
link_to "Delete", {  :controller => "admin",
  :action => "delete",
  :id => @post },
{  :class => 'css-class',
  :id => 'css-id',
  :confirm => "Are you sure?" }

image_tag "spinner.png", :class => "image", :alt => "Spinner"

mail_to "info@invisible.ch", "send mail",
      :subject => "Support request by #{@user.name}",
        :cc => @user.email,
        :body => '....',
        :encoding => "javascript"

stylesheet_link_tag "scaffold", "admin", :media => "all"
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionView/Helpers/UrlHelper.html을 참고하기 바란다.

HTML 폼

폼

```
<%= form_tag {  :action => :save }, {  :method => :post }  %>
```

이는 지정한 액션에 POST 요청을 하는 폼 태그를 만든다.

파일 업로드를 위한 MIME 멀티파트 폼은 :multipart =〉 true 옵션을 설정한다.

텍스트 필드

```
<%= text_field :modelname, :attribute_name, options  %>
```

다음은 폼의 텍스트 입력 필드를 만든다.

```
<input type="text" name="modelname[ attribute_name] " id= "attributename" />
```

예를 들면 다음과 같다.

```
text_field "post", "title", "size" => 20
        <input  type="text" id="post_title" name="post[ title] "
                    size="20" value="#{ @post.title} " />
```

숨겨진 필드를 만든다.

```
<%= hidden_field ... %>
```

비밀번호 필드를 만든다(* 문자로 채워진다).

```
<%= password_field ... %>
```

파일 필드를 만든다.

```
<%= file_field ... %>
```

텍스트 영역

```
<%= text_area ... %>
```

예를 들어보자.

```
text_area "post", "body", "cols" => 20, "rows" => 40
```

다음과 같은 결과를 만들어 낸다.

```
<textarea cols="20" rows="40" id="post_body" name="post[ body] ">
```

```
        #{ @post.body}
</textarea>
```

라디오 버튼

```
<%= radio_button :modelname, :attribute, :tag_value, options %>
```

예를 들어보자.

```
radio_button "post", "category", "rails"
radio_button "post", "category", "java"
         <input type="radio" id="post_category" name=" post[ category] "
value="rails" checked="checked" />
       <input type="radio" id="post_category" name=" post[ category] " value="java" />
```

체크 박스

```
<%= check_box :modelname, :attribute, options, on_value, off_value %>
```

예를 들어보자.

```
check_box "post", "validated"   # post.validated? 는 1 또는 0을 반환한다.
       <input type="checkbox" id="post_validate" name=" post[ validated] "
             value="1" checked="checked" />
       <input name="post[ validated] " type="hidden" value="0" />

check_box "puppy", "gooddog", {} , "yes", "no"
       <input type="checkbox" id="puppy_gooddog" name="puppy [ gooddog] " value="yes" />
      <input name="puppy[ gooddog] " type="hidden" value="no" />
```

옵션

select 태그를 만든다. 선택 목록을 배열로 넘긴다.

```
<%= select :variable, :attribute, choices, options, html_options %>
```

예를 들어보자.

```
select  "post",
        "person_id",
        Person.find_all.collect { |p| [  p.name, p.id ] } ,
        {  :include_blank => true }

 <select name="post[ person_id] ">
```

```
    <option></option>
    <option value="1" selected="selected">David</option>
    <option value="2">Sam</option>
    <option value="3">Tobias</option>
  </select>

<%= collection_select :variable, :attribute, choices, :id, :value %>
```

날짜와 시간

```
<%= date_select :variable, :attribute, options %>
<%= datetime_select :variable, :attribute, options %>
```

예를 들어보자.

```
date_select "post", "written_on"
date_select "user", "birthday", :start_year => 1910
date_select "user", "cc_date", :start_year => 2005,
                               :use_month_numbers => true,
                               :discard_day => true,
                               :order => [ :year, :month]

datetime_select "post", "written_on"
```

end_form 태그

```
<%= end_form_tag %>
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionView/Helpers/FormHelper.html을 참고하기 바란다.

레이아웃

레이아웃은 HTML 페이지의 주변을 정의한다. 이를 이용하여 일관성 있는 짜임새를 정의할 수 있다. 레이아웃은 app/views/layouts 디렉토리에 저장한다.

```
<html>
  <head>
    <title>Form: <%= controller.action_name %></title>
    <%= stylesheet_link_tag 'scaffold' %>
  </head>
```

```
  <body>
    <%= yield %>    # 이곳에 내용이 표시된다.
  </body>
</html>
----
class MyController < ApplicationController
  layout "standard", :except => [  :rss, :atom ]
...
end
----
class MyOtherController < ApplicationController
  layout :compute_layout

  # 이 메소드는 상황에 따라 적절한 레이아웃을 선택한다.
  def compute_layout
    return "admin" if session[ :role] == "admin"
    "standard"
  end
  ...
end
```

레이아웃은 컨트롤러의 인스턴스 변수들을 사용할 수 있다.
자세한 내용은 http://api.rubyonrails.com/classes/ActionController/Layout/ClassMethods.html을 참고하기 바란다.

부분 뷰

부분 뷰는 뷰를 생성하기 위한 구성 단위이다. 이는 같은 부분을 자주 표시할 때 유용하다. 이들은 파일에 저장된다.

```
render :partial => 'product'
```

이 명령은 _product.rhtml 파일에 있는 부분 뷰를 불러와서 인스턴스 변수 @product에 저장한다. 부분 뷰는 @product를 참고하면 된다.

```
render :partial => 'product', :locals => {  :product => @bought }
```

이 명령은 같은 부분 뷰를 불러오지만, 다른 인스턴스 변수에 할당한다.

```
render :partial => 'product', :collection => @product_list
```

이 명령은 @product_list의 각 항목에 대한 부분 뷰를 각 항목의 @product에 할당한다. 반복 카운터는 자동으로 partial_name_counter와 같은 형식(이전 예에서는 product_ counter)으로 템플릿에서 사용할 수 있다.

자세한 내용은 http://api.rubyonrails.com/classes/ActionView/Partials.html을 참고하기 바란다.

07_ Ajax

레이아웃에 자바스크립트 라이브러리를 포함시킨다.

```
<%= javascript_include_tag :defaults %>
```

원격 액션으로 링크하기

```
<%= link_to_remote "link", :update => 'some_div',
                      :url => { :action => 'show', :id => post.id } %>

<%= link_to_remote "link", :url => { :action => 'create',
                           :update => { :success => 'good_div',
                                        :failure => 'error_div' },
                           :loading => 'Element.show('spinner'),
                           :complete => 'Element.hide('spinner') } %>
```

콜백

:loading

브라우저가 원격 문서를 불러오고 있을 때 호출된다.

:loaded

브라우저가 원격 문서를 모두 불러왔을 때 호출된다.

:interactive

원격 문서를 모두 불러오지 않았더라도, 사용자가 원격 문서와 상호 작용을 할 수 있을 때 호출된다.

:success

XMLHttpRequest가 모든 작업을 마쳤을 때 호출되며, HTTP 상태 코드는 2XX 범위에 있다.

:failure

XMLHttpRequest가 모든 작업을 마쳤을 때 호출되며, HTTP 상태 코드는 2XX 범위 밖에 있다.

:complete

XMLHttpRequest가 작업을 완료했을 때 호출된다. (success/failure가 정의되어 있으면 이들이 먼저 호출되고 complete가 호출된다.)

상태 코드를 그대로 응답하는 방법도 있다.

```
link_to_remote word,
    :url => { :action => "action" },
    404 => "alert('Not found...? Wrong URL...?')",
    :failure => "alert('HTTP Error ' + request.status + '!')"
```

Ajax 폼

POST 요청 대신 XMLHttpRequest로 요청을 보내는 폼을 만들 수 있다. 매개변수는 정확히 같은 방법으로 보낸다. (컨트롤러는 params를 이용하여 매개변수를 사용한다) 자바스크립트를 지원하지 않는 브라우저에 대해서는 :html 옵션에 :action 메소드를 설정한다.

```
form_remote_tag :html => { :action => url_for(:controller => 'controller',
                                              :action => 'action'),
                           :method => :post }
```

텍스트 필드 자동 완성하기

뷰 템플릿

```
<%= text_field_with_auto_complete :model, :attribute %>
```

컨트롤러

```
auto_complete_for :model, :attribute
```

관찰 필드(Observe Field)

```
<label for="search">Search term:</label>
<%= text_field_tag :search %>
<%= observe_field(:search,
                  :frequency => 0.5,
                  :update => :results,
                  :url => { :action => :search }) %>
<div id="results"></div>
```

옵션

```
:on => :blur    # 이벤트 트리거(기본값은 :changed 또는 :clicked)
:with => ...    # 자바스크립트 표현식으로서 어떤 값을 보냈는지 저장
                # 기본값: "value"
:with => 'bla'  # "'bla' = value"
:with => 'a=b'  # "a=b"
```

관찰 폼(Observe Form)

observe_field와 같다.

periodically_call_remote

```
<%= periodically_call_remote(:update => 'process-list',
                             :url => { :action => :ps },
                             :frequency => 2 ) %>
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionView/Helpers/JavaScriptHelper.html을 참고하기 바란다.

08_ 애플리케이션 구성하기

많은 부분들은 config/environment.rb 파일에서 설정할 수 있다. 다음 목록은 완벽하지 않다.

세션 설정

```
config.action_controller.session_store = :active_record_store
# active_record_store 또는 :drb_store 또는
# :mem_cache_store 또는 :memory_store 또는 사용자가 직접 정의한 클래스

ActionController::Base.session_options[ :session_key] = 'my_app'
    # 애플리케이션에 자체적인 session_key를 사용한다.
ActionController::Base.session_options[ :session_id] = '12345'
    # 이 session_id를 사용한다. 지정하지 않으면 새로 생성된다.
ActionController::Base.session_options[ :session_expires] = 3.minute.from_now
    # 세션의 만료 시간을 설정
ActionController::Base.session_options[ :new_session] = true
    # 새로운 세션을 강제로 생성한다.
ActionController::Base.session_options[ :session_secure] = true
    # HTTPS일 때만 세션을 사용한다.
ActionController::Base.session_options[ :session_domain] = 'invisible.ch'
# 이 세션이 유효한 도메인을 설정한다. (기본값은 서버의 호스트 이름이다)
ActionController::Base.session_options[ :session_path] = '/my_app'
# 이 세션이 해당되는 경로.
# 기본값은 CGI 스크립트의 디렉토리이다.
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionController/SessionManagement/ClassMethods.html을 참고하기 바란다.

캐시 설정

```
ActionController::Base.fragment_cache_store = :file_store, "/path/to/cache/directory"
```

자세한 내용은 http://api.rubyonrails.com/classes/ActionController/Caching.html을 참고하기 바란다.

찾아보기

가~바

사~차

카~하

기타

A~E

F~I

J~P

R~Z