

Lezione 9: Sessioni e autenticazione

Matteo Vaccari

<http://matteo.vaccari.name/>
matteo.vaccari@uninsubria.it



(cc) Matteo Vaccari. Published in Italy.
Attribution – Non commercial – Share alike 2.5

HTTP: stateless and anonymous

Each time your application receives a request from the browser is like the first time they've talked to each other.

title

Sessione: una sequenza di transazioni HTTP

- ▶ compiute da uno stesso utente
- ▶ in un intervallo di tempo limitato (ore)

Come implementare una *sessione* sopra un protocollo *stateless*?

Cookie: un frammento di informazione “opaco”

cookie A handle, transaction ID, or other token of agreement between cooperating programs.

The claim check you get from a dry-cleaning shop is a perfect mundane example of a cookie; the only thing it's useful for is to relate a later transaction to this one (so you get the same clothes back)

*Now mainstream in the specific sense of **web-browser cookies**.*

— the Jargon File

cookie

*a packet of data sent by an Internet server to a browser, which is returned by the browser each time it subsequently accesses the same server, used to **identify** the user or **track** their access to the server.*

— *Oxford American Dictionary*

I cookie viaggiano con gli header http

```
HTTP/1.1 200 OK
Date: Thu, 19 Dec 2002 08:05:53 GMT
Server: Apache/1.3.26 (Unix) PHP/4.2.1
Set-Cookie: foobar=2303242061a73ca9f99988920629bc3c; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html
```

Protocollo

0. Il browser chiede una pagina al server `www.foo.com`
1. Il server manda lo header `Set-Cookie`:
2. Il browser conserva il cookie
3. D'ora innanzi per ogni pagina di `www.foo.com`, il browser manda lo header `Cookie`:

Cookies, cont.

Permettono a un sito web di conservare dati sul browser dell'utente

Applicazioni:

- ▶ conservare le preferenze
- ▶ mantenere una sessione
- ▶ identificare il browser (e l'utente)

Ci sono *rilevanti* problemi di privacy

Anatomia di un cookie

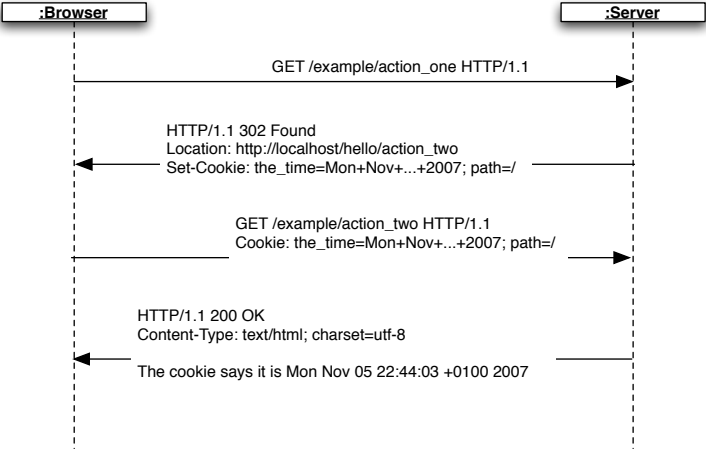
Un cookie è fatto di:

1. un nome: es. RUBY_SESSID
2. un dominio: solo quel dominio può leggere il cookie
3. un valore: una sequenza di lettere e numeri
4. una data di scadenza

Cookies in Rails I

```
class ExampleController < ApplicationController
  def action_one
    cookies[:the_time] = Time.now.to_s
    redirect_to :action => 'action_two'
  end
  def action_two
    cookie_value = cookies[:the_time]
    render(:text => "The cookie says it is #{cookie_value}")
  end
end
```

Cookies in Rails II



Uso appropriato dei cookies

a) preferenze dell'utente

b) **chiave** per i dati **server-side**

⇒ session data

Cookie poisoning

Un negozio online ha riservato uno sconto del 10% sul prossimo acquisto a un cliente:

```
cookies[:discount] = 10
```

L'utente smaliziato modifica il cookie in

```
c:\Documents and settings\pippo\Temporary Internet Files
```

e lo fa diventare 90 ...

⇒ MAI fidarsi del contenuto di un cookie!

Algoritmo per conservare la sessione

1. arriva un nuovo utente
2. genero un numero casuale molto grande: il **session id**
es. 11e144e3b510f0c5263e78eff0eb6e4a
3. associo il session id all'utente con un cookie
4. conservo i dati della sessione in un **file**
tmp/sessions/sess_11e144e3b510f0c5263e78eff0eb6e4a
5. quando l'utente torna, il browser mi dà il session id con un cookie

Sessione != database

I dati della sessione sono *temporanei*...

Non sostituiscono un database

Tipicamente: in sessione conservo lo `user_id`

Dove salviamo la sessione?

a) (Obsoleto) **serializzata** su file temporanei

```
$ ls -l tmp/sessions/ruby_sess.*  
-rw---- 1 tl608515 users 79 Dec 21 17:48 ruby_sess.056a2770c0f91362  
-rw---- 1 tl608515 users 98 Dec 23 01:18 ruby_sess.1fee3459a1a7fa81  
...
```

b) Default Rails: **crittata** nel cookie stesso

c) Nel database

```
# (create the session table with 'rake db:sessions:create')
```

```
config.action_controller.session_store = :active_record_store
```

Perché il session id deve essere casuale?

C'era un web server che forniva session id sequenziali: 1, 2, 3,...

Gli utenti potevano **indovinare** il numero di sessione

Manipolando i cookies potevano **rubare** la sessione di un'altro utente

Altri meccanismi per propagare la sessione

(Alcuni utenti disabilitano i cookies...)

a) codificare l'id nel PATH della url

`http://www.amazon.com/.../home.html/103-9557789-1670200`

⇒ devo configurare il web server in modo speciale

b) codificare l'id nella query

`http://www.foo.com/bar?sessid=11e144e3b510f0c5263e78eff0eb6e4a`

⇒ devo *riscrivere* tutti i link interni al mio sito

Autenticazione

Autenticazione: provare l'identità dell'utente

Metodo tradizionale: login e password

Problemi:

- ▶ le connessioni HTTP viaggiano **in chiaro**
⇒ *considerate la possibilità di usare HTTPS per le password*
- ▶ molti utenti scelgono **password deboli**
 - ▶ dictionary attack
 - ▶ brute force attack
- ▶ ⇒ **contare** i tentativi di login falliti
- ▶ disabilitare l'account e/o l'IP (temporaneamente?)
- ▶ notificare un amministratore

Due meccanismi di autenticazione

- ▶ Autenticazione attraverso HTTP
- ▶ Autenticazione attraverso una form e variabile di sessione

Autenticazione http

app/some_controller.rb

```
class SomeController < ApplicationController
  before_filter :authenticate # filtro!
  ...
  protected

  def authenticate
    authenticate_or_request_with_http_basic do |username, password|
      username == 'foo' && password == 'bar'
    end
  end
end
```

(Se l'autenticazione fallisce l'utente ottiene un 403: Not Authorized)

<http://railscasts.com/episodes/82>

La tabella utenti

1. `script/generate model User`
2. `script/generate controller Login`

La tabella *users*

```
def self.up
  create_table :users do |t|
    t.string :name

    # salviamo la password crittata con SHA1
    t.string :hashed_password, :length => 40

    # ruoli: user o admin
    t.string :role, :default => 'user'
  end
end
```

Il modello User

Definiamo il comportamento desiderato con un test

```
require 'test_helper'
```

```
class UserTest < ActiveSupport::TestCase
  test "password is hashed when we create the user" do
    user = User.new(:name => 'pippo', :password => 'secret')
    user.save!

    # ottenuta con
    # echo -n 'secret' | shasum
    SHA1_OF_SECRET = 'e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4'

    assert_equal SHA1_OF_SECRET, user.hash_password
  end
end
```

Il modello User

```
require 'digest/sha1'
```

```
class User < ActiveRecord::Base
  attr_accessor :password # questo attributo esiste solo in RAM

  # filtro eseguito prima della 'insert'
  def before_create
    self.hashed_password = User.hash_password(self.password)
  end

  private

  def User.hash_password(password)
    Digest::SHA1.hexdigest(password)
  end
end
```

Estendiamo il modello User

```
class UserTest < ActiveSupport::TestCase
  # setup viene eseguito prima di tutti i test
  def setup
    User.delete_all
    @pluto = User.create! :name => 'pluto', :password => 'foo'
  end

  test "failed login" do
    assert_nil User.authenticate("pluto", "bad password")
    assert_nil User.authenticate("bad name", "foo")
  end

  test "login successful" do
    found = User.authenticate("pluto", "foo")
    assert_equal found, @pluto
  end
end
```

Ancora sul modello

```
class User < ActiveRecord::Base
  ...
  private

  def User.hash_password(password)
    Digest::SHA1.hexdigest(password)
  end

  def User.authenticate(name, password)
    hashed_password = hash_password(password)
    User.find_by_name_and_hashed_password(name, hashed_password)
  end
end
```

L'azione di *login*

```
class LoginController < ApplicationController
  def login
    logged_in_user = User.authenticate(params[:name], params[:password])
    if logged_in_user
      session[:user_id] = logged_in_user.id
      redirect_to(:action => 'index')
    else
      flash[:notice] = 'Invalid user/password combination'
    end
  end
end
```

Come limitare l'accesso?

app/controllers/application.rb

```
class ApplicationController
  before_filter :authorize

  protected

  def authorize
    unless session[:user_id]
      flash[:notice] = "Please log in"
      redirect_to :controller => "login", :action => "login"
    end
  end
end
```

app/controllers/login_controller.rb

```
class LoginController < ApplicationController
  before_filter :authorize, :except => :login
  # ...
end
```