

Sviluppo web

dall'antichità all'avanguardia e ritorno

Michele Beltrame

Perl.It

1993

Era una notte buia e tempestosa

CGI

(NCSA, CERN, ...)

C

C++

Hmmm

- Allocazione memoria
- Tipi statici
- Limitate librerie text-processing

Web

\Rightarrow

text processing

string.h

H m m m m

Perl

```
# Regular expression
```

```
if ( $cc =~ /^\d{4}-\d{2}-\d{2}$/ )  
    # data (migliorabile)  
}
```

```
# split, join, ...
```

Codice:

mix tra HTML e Perl/C/altro

```
print '<body>';  
print '<p>' . $contenuto . '</p>';
```

Quasi accettabile...

...finché i siti li facevano i programmatori

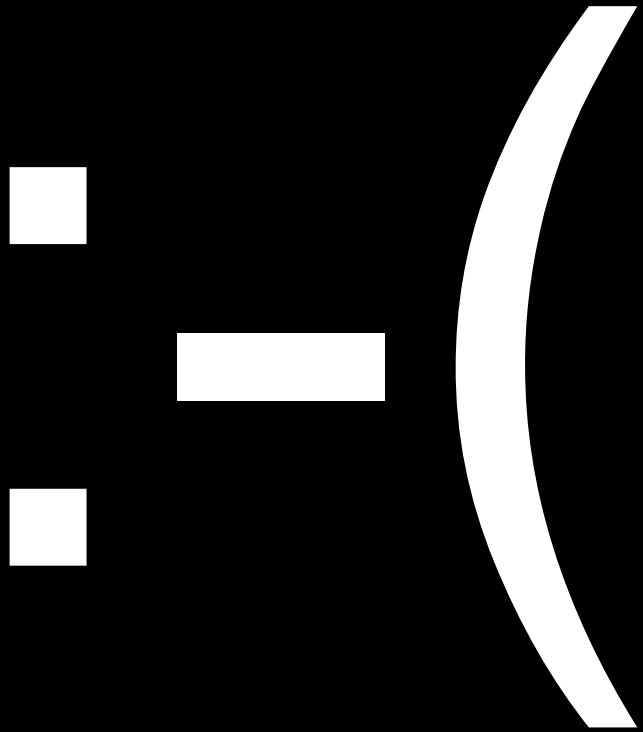
Web designer

Siti più belli...

...a volte

Difficoltà di integrazione

```
print '<ul>';
for my $article (@$articles) {
    print '<li>';
    if ( $article->{id} == $selected_article ) {
        # Niente CSS fino al 1996
        print '<font color="#0000FF">' . $article->{title} . '</font>';
    } else {
        print $article->{title};
    }
    print '</li>';
}
print '</ul>';
```



Poco manutenibile in ogni caso

Separare codice HTML da linguaggio scripting

Sistemi di templating

```
[% peanut %]  
[% FOREACH product IN lproducts %]  
  
[% product.name %]  
  
[% END %]
```

HTML dentro linguaggio scripting

MALE

linguaggio scripting dentro HTML

MALE

MALE

PHP

originariamente scritto in Perl

da un tizio per gestire la propria home page

Non si può mai sapere
come la gente userà
ciò che lasci in giro
sul web

PHP, (some of) the bad

- Namespace caotico
- Librerie (mysql, mysqli, gd) incluse nel core
- Unicode-agnostic
- Poco espressivo (per un linguaggio dinamico)
- API bizzarra: *mysql_escape_string?* *mysql_real_escape_string?!?!?*
- **Ha cresciuto una serie di sviluppatori**
...che non pensano che quelli sopra siano problemi
...poco attenti alla sicurezza

Ma

- Estrema semplicità di deployment
 - Il codice funziona ovunque
 - Funziona ovunque nello stesso modo!
 - Tutti i provider supportano PHP

PHP

ha reso il sito web dinamico la norma

non è poco

e ora anche PHP usa dei
sistemi di templating

:-)

Java

Il **solo** linguaggio per applicazioni web mission-critical



"vietiamo l'ereditarietà multipla"

MI è utile nello 0,0042% dei casi
raccomandare una best practice
ma non vietare di ignorarla

il codice non sarà manutenibile!

brutte cose succederanno al vostro software!

moriranno dei cuccioli di gatto!!!

è giusto vi sia un sentiero consigliato

ma nulla dovrebbe
volontariamente
essere reso impossibile

Python

Attenti agli spazi!

Ma è un gran linguaggio

1995

Netscape LiveScript
JavaScript

Grande idea

Pessimo nome

ECMAScript?

```
# Linguaggio ancora poco potente  
# ma un cambiamento interessante
```

```
visibility: hidden;  
display: none;
```

tic tac

... trascorre il Medioevo del web programming ...

tic tac

20004

Web 2.0

Qualunque cosa voglia dire

AJAX

XMLHttpRequest

Possiamo comunicare col server
senza ricaricare la pagina

Yay!

Talmente semplice e stupido
che nessuno ci aveva mai pensato

- autocompletamento
- form dinamiche
- validazione on-the-fly

la scalabilità diventa un argomento
anche per applicazioni "normali"

Ruby

Ruby

Ruby on Rails

Prime release di Rails

- Lento
- Scarso supporto Unicode
- Leak
- Bug vari
- Ideale per applicazioni medio-grandi

Ma

MVC per tutti

Esisteva già ma...

Rails l'ha fatto conoscere

Modern Web Programming

Il framework che vanta più
tentativi di imitazione

Molti dei quali riusciti
meglio dell'originale

- Django (Python)
- Catalyst (Perl)
- un'infinità in vari linguaggi

Dispatcher avanzati

<http://mysite.ext/it/products/view/231>

<http://mysite.ext/en/products/buy/342>

it/products/buy/342

Lingua

```
sub lang : Chained('/') PathPart('') CaptureArgs(1) {  
  my ( $self, $c, $lang ) = @_  
  # ...  
}
```

Prodotti

```
sub products : Chained('lang') PathPart('products') CaptureArgs(0) {  
  my ( $self, $c ) = @_  
  # ...  
}
```

Endpoint 1

```
sub view : Chained('products') PathPart('view') Args(1) {  
  my ( $self, $c, $id ) = @_  
  # ...  
}
```

Endpoint 2

```
sub buy : Chained('products') PathPart('buy') Args(1) {  
  my ( $self, $c, $id ) = @_  
  # ...  
}
```

mod_rewrite

=>

Web-server agnostic

Proliferazione di web server piccoli ed efficienti

- lighttpd
- nginx
- ...

ORM

(Object-Relational Mapper)

una storia di amore e di odio

```
my $rs = $c->model('Dbs::Animali')->search({
    'me.anni'      => { '<=' => 4 },
    'razza.nome'   => 'beagle',
}, {
    'join'         => 'razza',
    'order_by'     => 'me.nome',
    'rows'         => 42,
});
```

```
select me.*
from animali me
join razze razza on razza.id = me.idrazza
where me.anni <= 4 and razza.nome = 'beagle'
order by me.nome
limit 42
```

~2007

REST

HTTP

~~GET /entries/edit?id=42~~

GET /entries/edit/42

~~GET /entries/edit?id=42&task=update~~

POST /entries/edit/42

GET /entries/edit/new

GET /entries/delete/42

GET /entries/42

POST /entries/42

PUT /entries

DELETE /entries/42

```
<form method="put">
```

supportato

solo in XHTML 2.0

XHTML 2.0???

si usa AJAX

AJAJ

in realtà

JSON

la più grande invenzione del web moderno
(2005)

serializzazione XML

- prolissa
- poco leggibile

standard per serializzare i dati
e trasmetterli tra client e server

facilmente leggibile dagli umani!

HTML 5

Contenitore per vari "standard"
(molti dei quali client-side)

ma c'è qualcosa di bello

XHTML

 ==>


```
class="miaclasses" => class=miaclasses
```

JavaScript

```
//<![CDATA[  
//]]>
```

```
<! --
```

```
-->
```

WebSocket

Connessione full-duplex persistente
client \Leftrightarrow server

Niente più hack:

- AJAX polling
- Flash (sik)

Riduzione overhead per invio di dati

=>

Incremento di velocità di
vari ordini di grandezza

NodeJS

async

=>

prestazioni molto alte
rispetto a thread/fork

arriva anche negli altri linguaggi

Perl

Mojolicious

framework leggero

Sebastian Riedel

- un tizio un po' permaloso
- un geniale designer di API

```
use Mojolicious::Lite;
```

```
get '/' => { text => 'Tu sia dannato, Barone Rosso!' };
```

```
app->start;
```

Da shell...

```
$ morbo hello.pl
```

```
Server available at http://127.0.0.1:3000.
```

Deployment:

Preforking server

FastCGI

CGI

Plack/PSGI

Dispatcher + IO

Routes

```
# Regex (overkill)
qr|/animali/list/(\d+)| -> $self->render(text => $1);

# Route
/animali/list/:id -> $self->render(text => $id);

# Routes & MVC
$r->route('/contatti')
    ->to(controller => 'pagine', action => 'cont');
```

```
# Nested routes (riutilizzo codice)
# /animali      -> undef
# /animali/list -> {controller => 'records', action => 'list'}
# /animali/view -> {controller => 'records', action => 'view'}

my $foo = $r->route('/animali')->to(controller => 'records');
$foo->route('/list')->to(action => 'list');
$foo->route('/view')->to(action => 'view');
```

Ai perlisti piace funzionale...

```
# Closure (callback)
```

```
$r->route('/animali/cane/abbaia')->to(cb => sub {  
    my $self = shift;  
    $self->render(text => 'Bau');  
});
```

```
# Placeholder restrittivi
```

```
$r->route('/animali/:specie')  
  ->to(controller => 'animali', action => 'info');
```

```
$r->route('/animali/:specie', specie => ['cane', 'gatto'])  
  ->to(controller => 'animali', action => 'info');
```

```
$r->route('/animali/:specie', specie => qr/^leo/)  
  ->to(controller => 'animali', action => 'info');
```

```
# HTTP method (REST)
```

```
$r->route('/animali')->via('GET', 'POST')  
    ->to(controller => 'animali', action => 'info');
```

```
$r->route('/animali')->via('DELETE')  
    ->to(controller => 'animali', action => 'delete');
```

Routes sintetiche
(Mojolicious::Lite)

Do-What-I-Mean

ma un po' meno assoggettabili al proprio volere

```
get '/animali' => sub { ... }
```

```
post '/animali' => sub { ... }
```

```
delete '/animali' => sub { ... }
```

```
# Formati
```

```
# /animali      -> {controller => 'animali'}
```

```
# /animali.html -> {controller => 'animali', action => 'info', format => 'html'}
```

```
# /animali.txt  -> {controller => 'animali', action => 'info', format => 'text'}
```

```
$r->route('/animali')->to(controller => 'animali', action => 'info');
```

```
# $format = ...
```

Utili per supportare URL più significative
e per inviare risposte diverse in base alla richiesta

Risposta in base all'header HTTP

Accept: application/json

Accept: text/plain

```
get '/list/:offset' => sub {
  my $self      = shift;
  my $numbers = [42 .. $self->param('offset')];
  $self->respond_to(
    json => {json => $numbers},
    txt  => {text => join(', ', @$numbers)}
  );
};
```

JSON

```
post '/update' => sub {
  my $self = shift;
  my $chartdata = $self->req->json;

  # Codice che faccia qualcosa

  return $self->render_json({
    status      => 'ok',
    updated     => \@updated,
    not_updated => \@not_updated,
  });
}
```

WebSocket

```
# Un semplice pong
websocket '/ping' => sub {
  my $self = shift;
  # ... async ...
  $self->on(message => sub {
    my ($self, $message) = @_ ;
    $self->send_message("hai detto: $message");
  });
};
```

Tutta la potenza di CPAN

- ORM: DBIx::Class, Rose::DB::Object, Fey::ORM, ...
- Templating: TT, Text::Xslate, HTML::Zoom, ...
- Interfacciamento a database
- FTP, POP3, SSH, SMTP, IRC, ...
- Import/export Excel, CSV, PDF, ...
- ritocco immagini, generazione barcode, ...
- [molto molto, altro]

Molto *altro*

Interessante?

Divertente?

Mojolicious

<http://mojolicious.org>

Cerca anche...

Dancer

Catalyst

The Perl Foundation

Google Code-In

- contest per studenti universitari
- sviluppo software open source
- punti, magliette, soldi, viaggi

non in Italia

- problemi burocratici :-/
- per ora

Grazie!

www.perl.it