Formulaire HTML : le minimum à savoir

schéma général pour les formulaires simples

```
<form action="/resource/to/go" method="post|get" >
   <!-- inputs elements -->
   <¯- intput element of submit type -->
</form>

```

Validation GET : paramètres `name=val` renvoyés dans la chaîne de requête (*query string*).

```
get /resource/to/go?name=val&name2=val2
```

Validation POST : les paramètres sont renvoyés dans le corps de la requête.

```
post /resource/to/go
Content-Length: 19
Content-Type: application/x-www-form-urlencoded

name=val&name2=val2
```

tag input et fonctionnement global

Chaque tag input inclus dans un formulaire a :

1. un attribut `name`
2. un attribut `type`
3. un attribut `value` : indique la valeur par défaut ou la valeur renvoyée (en fonction du `type`)

Quand le formulaire est validé, un paramètre est renvoyé par tag input présent dans le formulaire.

- Le nom du paramètre est la valeur donnée à l'attribut `name`.
- La valeur est soit celle saisie par l'utilisateur pour les types permettant une saisie, soit la valeur de l'attribut `value`.

L'attribut `type` code la présentation html faite par le navigateur pour que l'utilisateur renseigne la valeur.

- text : un champ de texte.
- password : idem, mais on ne voit que * à l'insertion
- radio : l'utilisateur ne peut sélectionner qu'un bouton radio sur tous ceux

ayant le même attribut `name`

- checkbox : idem avec sélection multiple.
- submit : génère un boutton permettant la validation du formulaire.

tag label

```
<form>
  <label for="login">Identifiant</label>
  <input type="text" name="login"/>
</form>

```

Le tag `label` définit un label à afficher pour un tag `input`. L'attribut `for` du label doit correspondre à l'attribut `name` de l'input visé.

Rails et la gestion des URL

Pour tester dans la console les méthodes provenant de la couche de routage, préfixer par app

```
rails console
>> app.people_path
=> "/people"
```

routes nommées

Disponibles au niveau des contrôleurs et des vues :

```
routage : people GET    /people(.:format)      {:action=>"index", :controller=>"people"}
people_path => /people

routage : person GET    /people/:id(.:format)  {:action=>"show", :controller=>"people"}
person_path(42) => /people/42
```

Le suffixe `_url` rajoute en plus le nom du serveur :

```
rails console
>> app.people_url
=> "http://www.example.com/people"
```

url_for

`url_for` permet de constituer une url :

- en spécifiant le contrôleur, l'action et les paramètres par appel de la couche de routage :

  url_for(:controller => :people, :action => :show, :id => 42) => "http://www.example.com/people/42"

- en donnant directement l'instance d'une classe descendant d'`ActiveRecord::Base` :

```
1 p = Person.new
2 url_for p
3 => "http://www.example.com/people"
4 p = Person.first
5 url_for p # appelle p#to_param pour avoir l'id
6 => "http://www.example.com/people/1"
7
```

Rails et helpers

Rails offre des méthodes disponibles dans les vues permettant de facilement générer des tags html et leur attributs.

Pour tester un helper dans la console rails, il suffit de prefixer par helper.

Par exemple :

```
rails console
>> helper.link_to "toto", "http://toto.org"
=> "<a href=\"http://toto.org\">toto</a>"
```

link_to

Le but de ce helper est de générer le tag a avec l'attribut `href`.

```
link_to(texte_lien, url/url_for={}, html_options = {})
```

Le 2e argument spécifie l'attribut `href`. 2 formes pour cet argument : soit du texte

```
link_to "toto", "/ailleurs"
=> "<a href=\"/ailleurs\">toto</a>"
link_to "toto", people_path
=> "<a href=\"/people\">toto</a>"
```

soit en utilisant `url_for` :

```
p = Person.new
link_to "create a person", p
=> "<a href=\"http://www.example.com/people\">create a person</a>"
```

Le 3e argument permet de spécifier des options HTML, comme une class (`:class => "toto"`) ou l'identifiant (`:id => "titi`)

```
link_to "toto", "http://toto.org", :id => 42, :class => :toto
```

```
=> "<a href=\"http://toto.org\" class=\"toto\" id=\"42\">toto</a>"
```

Helpers pour formulaire de base

```
form_tag(url_for_options = {}, options = {}, &block)
```

crée la première ligne et la dernière du formulaire. Le contenu du bloc donne le contenu du formulaire, avec les helpers :

```
submit_tag(value = "Save changes", options = {})
text_field_tag(name, value = nil, options = {})
password_field_tag(name = "password", value = nil, options = {})
radio_button_tag(name, value, checked = false, options = {})
check_box_tag(name, value = "1", checked = false, options = {})
label_tag(name = nil, content_or_options = nil, options = nil, &block)
```