Le langage PHP

Introduction à la programmation Web (1/3)

Benjamin Canou

6 avril 2018

Cours INE11 @ ENSTA

Plan de ce cours

1. Quelques rappels d'HTML pour s'échauffer
2. Comment est né et fonctionne PHP
3. Comment générer une page HTML en PHP
4. Comment passer d'une page à l'autre avec PHP

La structure du Web : HTML

Commençons par un exemple :

```
 1 : <html>
 2 :   <head>
 3 :     <title>Page sans grand intérêt</title>
 4 :     <meta charset="utf-8" />
 5 :   </head>
 6 :   <body>
 7 :     <h1>Bonjour</h1>
 8 :     <p id="banner">
 9 :       Cliquez ici :
10 :       <a href="http://www.ragondin.com/ra/gon/din.html">
11 :         <img src="images/radongin.png">
12 :       </a>
13 :     </p>
14 :   </body>
15 : </html>
```

Langage à deux niveaux de grammaire :

1. Langage de documents structurés (XML/SGML) :
   - Texte brut pour le contenu
   - Balises pour la structure (simples comme img ou composites comme div)
   - Attributs pour préciser le sens des balises (ex. src)
2. Sous-ensemble accepté par le navigateur (HTML/XHTML) :
   - Noms possibles des balises
   - Schémas d'imbrication des balises
   - Attributs acceptés

Navigation **hypertexte** entre les documents et sites

- Au sein d'un site : **path Unix (URL relative)**
  `images/radongin.png`
- Pour joindre un autre serveur : **URL absolue**
  `http://www.ragondin.com/ra/gon/din.html`
  - Protocole (ex. http, ftp, file)
  - Domaine (ex. ragondin.com)
  - URL relative au sein du site cible (ex. ra/gon/din.html)
  - Paramètres (cf. la suite du cours)
  - Fragment (liens au sein des pages)

Sous le capot : HTTP

Quand je clique sur `http://domaine/dir/page.html`, je veux dire :

1. **Donne moi le fichier page.html,**
2. **situé dans le dossier dir,**
3. **sur la machine joignable sous le nom domaine,**
4. **en utilisant le protocole HTTP.**

Ce qu'il se passe chez moi :

1. Mon navigateur identifie les composantes de l'URL,
2. il demande au serveur DNS l'adresse IP correspondant à domaine,
3. il ouvre une connexion TCP sur le port 80 à cette adresse,
4. il envoie une requête HTTP pour demander le fichier dir/page.html,
5. il reçoit une réponse HTTP contenant le fichier demandé,
6. il ferme la connexion ou attend la fermeture du serveur,
7. il l'affiche et lance l'interaction.

Ce qu'il se passe sur le serveur :

1. Le serveur attend des connexions TCP sur son port 80,
2. il reçoit une requête HTTP pour demander le fichier dir/page.html,
3. il vérifie l'existence du fichier,
4. il envoie une réponse HTTP contenant le fichier demandé,
5. il ferme la connexion.

Contenu des échanges

- **La requête**

```
1 : GET /dirs/page.html HTTP/1.1
2 :  -- ligne vide --
```

- **La réponse**

```
1 : HTTP/1.1 200 OK
2 :  -- ligne vide --
3 : <html>
4 :   ...
5 : </html>
```

En pratique, les messages sont plus complexes, pour observer :

- **Dans la console :** `telnet ragondin.com 80`
- **Utiliser l'onglet réseau des developper tools dans Fx ou Chrome**

Requête :

```
1 : MÉTHODE /url/relative HTTP/1.1
2 : En-tête: valeur
3 : En-tête: valeur
4 : ...
5 :   -- ligne vide --
6 : (corps de la requête)
```

Les méthodes :

- **GET : accès à une page**
- **POST : accès avec envoi d'informations (dans le corps)**
- **Beaucoup d'autres existent, moins utilisés**

Les en-têtes :

- **Sur le format du corps : Content-Length, Content-Type, etc.**
- **Pour les proxys / le cache : Expires, Date, etc.**

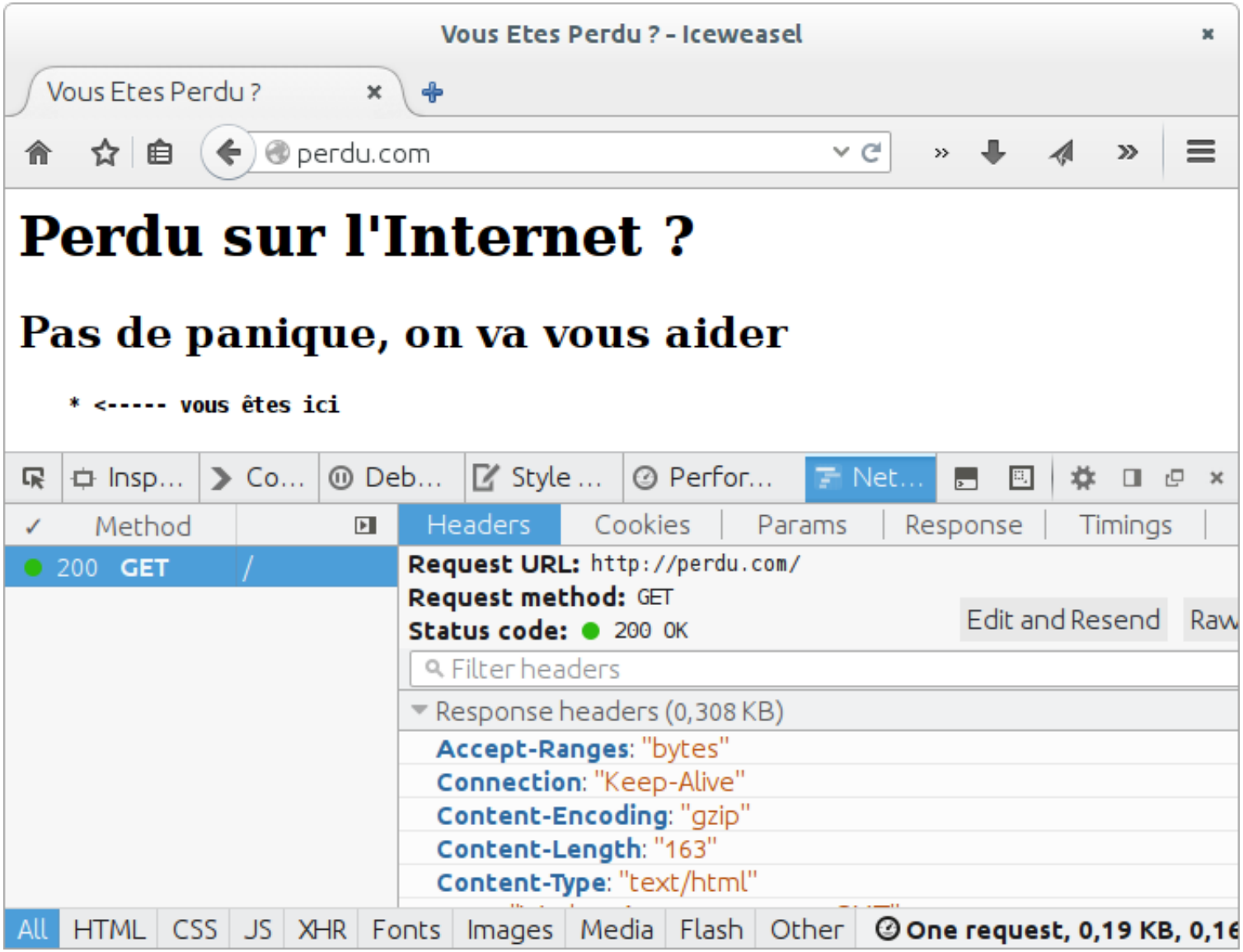
Réponse :

```
1 : HTTP/1.1 code message
2 : En-tête: valeur
3 : En-tête: valeur
4 : ...
5 :   --- ligne vide ---
6 : (corps de la réponse)
```

Les codes :

- **200+ : Ok**
- **300+ : Redirection**
- **400+ : Erreur du client (le fameux 404 Not Found)**
- **500+ : Erreur du serveur**

Examen avec Firefox



Des pages dynamiques : CGI

But : rendre dynamique le contenu des pages

- En fonction de données rentrées
- En reconnaissant les utilisateurs

Solution : délégation de la réponse à des programmes externes

Il faut passer des paramètres aux programmes :

- Paramètres GET à la fin de l'url :
  `/page.php?arg1=val1&arg2=val2&...`
- Paramètres POST dans le corps de la requête, via formulaire

```
1 : <form action="norm.php" method="POST">
2 :   x = <input type="number" name="x"><br/>
3 :   y = <input type="number" name="y"><br/>
4 :   <input type="submit" name="calculer">
5 : </form>
```

Standard : CGI (Common Gateway Interface)

- Le programme reçoit les paramètres dans ses variables d'environnement,
- il lit le corps de la requête sur son entrée standard,
- il écrit la réponse directement sur sa sortie standard.

On peut utiliser :

- Des langages applicatifs généralistes (Java, C++, OCaml),
- des langages de script généralistes (Perl, Python),
- des langages dédiés : PHP.

Le Web *pour les nuls* : PHP

Qu’est-ce que PHP ?

Techniquement :

- Un langage de script (langage non typé fait pour manipuler du texte),
- que l’on va utiliser pour générer des pages en réponse à des requêtes,
- que l’on pourrait utiliser de façon indépendante.

Historiquement :

- Conçu dans un *garage* comme alternative plus simple à Perl,
- de plus en plus de traits ajoutés (objets, passage par référence, etc.),
- un très grand succès au sein du fameux LAMP,
- de nombreuses bibliothèques et frameworks.

Nous allons juste en voir un aperçu.

Premiers pas

Hello world en PHP:

```
1 : <?php
2 :   echo "Hello␣world" ;
3 : ?>
```

On peut tisser du PHP et du HTML :

```
1 : Hello world in <?php
2 :   echo (date ("F")) ;
3 : ?> !
```

Qui équivaut à :

```
1 : <?php
2 :   echo "Hello␣world␣in␣" ;
3 :   echo (date ("F")) ;
4 :   echo "␣!" ;
5 : ?>
```

Syntaxe de base :

- **Affichage :** `echo "..." ;`
- **Appel de fonction :** `f (x, y)`
- **Opérations arithmétiques classiques**
- **Égalité un peu spéciale :** `==`, `===`
- **Conversion de type :** `(int) "1234"` **(et implicites)**

Types (dynamiques) :

- **entiers, flottants, booléens, chaînes**
- **tableaux associatifs**
- **objets, fonctions (pas au programme)**

Variables :

- **Variable :** `$var`
- **Affectation :** `$var = expr`
- **Affectation par référence :** `$var =& expr`
- **Test :** `isset ($var)`

Tableaux associatifs :

- **Création :** `$tab = array ()`
- **Accès :** `$tab[0]` **(indice : nombre ou chaîne)**
- **Affectation :** `$tab[0] = expr`
- **Test :** `isset ($tab[0])`
- **Suppression :** `unset ($tab[0])`
- **Création et initialisation :** `$tab = array ("a","b",23)`
- **Initialisation par champs :** `$tab = array ("x" => 2, "y" => 4)`

Structures de contrôle à la C :

- `if (condition) { /* si vrai */ }`
- `if (condition) { /* si vrai */ } else { /* sinon */ }`
- `while (condition) { /* corps */ }` **(attention aux boucles infinies !)**
- `for (init ; condition ; incr) { /* corps */ }`
- `foreach (tab as $val) { /* corps */ }`
- `foreach (tab as $index => $val) { /* corps */ }`
- **Dans les boucles :** `break`, `continue`

Définition de fonction :

```
1 : function roots ($a, $b = 0 /* valeur par defaut */) {
2 :   $c = $a + $b ; /* nouvelle variables locale $c */
3 :   return $a + $b ;
4 : }
```

Variables globales :

```
1 : function f($y = 4) {
2 :   $var = expr ;   /* $var est locale */
3 :   global $var2 ;  /* utilisation de la globale $var2 */
4 :   $var2 = expr ;  /* $var2 est globale */
5 : }
6 : /* à la sortie, $var est oubliée, $var2 est modifiée */
```

Il faut utiliser `global $v`dans chaque fonction utilisant `$v`.

Paramètres du script

Variables super-globales `$_GET` **et** `$_POST`**:**

- `$_GET["var"]`**: paramètre d'URL**
  Passées dans l'adresse après un ?
  Ex. `/test.php?a=3&b=bob` **=>** `$_GET["a"]="3"` **et** `$_GET["b"]="bob"`
- `$_POST["var"]`**: champ de formulaire**
  Passées dans le corps de la requête.

Exemple de page utilisant les paramètres :

```
 1 : function roots ($a, $b, $c) {
 2 :   $delta = $b * $b - 4 * $a * $c ;
 3 :   if ($delta === 0) {
 4 :     return (array (- $b - sqrt($delta) / (2 * $a))) ;
 5 :   } else if ($delta > 0) {
 6 :     return (array (- $b - sqrt($delta) / (2 * $a),
 7 :                    - $b + sqrt($delta) / (2 * $a))) ;
 8 :   } else { return array () ; }
 9 : }
10 :
11 : $a = (int) $_GET["a"] ;
12 : $b = (int) $_GET["b"] ;
13 : $c = (int) $_GET["c"] ;
14 :
15 : echo (var_dump (roots ($a, $b, $c)))
```

Appel avec : `roots.php?a=12&b=7&c=23`

Pattern: un paramètre pour indiquer l'action :

- On cherche le paramètre `action`.
- S'il n'existe pas, on choisit une valeur par defaut (ex. accueil).
- Si l'action n'est par reconnue (bug ou attaque), on produit une erreur.

Pour chaque action:

- On analyse les autres arguments.
- On appelle une fonction dédiée.

Exemple: un paramètre pour indiquer choisir la commande :

```
1 : if ($_GET["command"] == "roots") {
2 :     $a = (int) $_GET["a"] ;
3 :     $b = (int) $_GET["b"] ;
4 :     $c = (int) $_GET["c"] ;
5 :
6 :     echo (var_dump (roots ($a, $b, $c))) ;
7 : } else { /* default */
8 :     echo "<form␣method='GET'>" ;
9 :     echo "␣␣a␣=␣<input␣type='text'␣name='a'>" ;
10 :    echo "␣␣b␣=␣<input␣type='text'␣name='b'>" ;
11 :    echo "␣␣c␣=␣<input␣type='text'␣name='c'>" ;
12 :    echo "␣␣<input␣type='hidden'␣name='command'␣value='roots'>" ;
13 :    echo "␣␣<input␣type='submit'␣value='solve'>" ;
14 :    echo "</form>" ;
15 : }
```

Connexion et état

Solution possible : encoder l'état dans la requête

- **Ex (stupide):** `/app.php?shampoing=23&saucisses=6`
- **À limiter aux cas simples et sûrs**
- **Permet l'échange et l'enregistrement d'URLs**

Sessions :

- **Mécanisme pour simuler un protocole connecté au dessus d'HTTP.**
- **Permet de stocker des données sur le serveur le temps d'une connexion.**

Utilisation très simple :

```
1 : session_start () ;
2 : if (!isset ($_SESSION["count"])) {
3 :   echo "C'est␣votre␣premère␣fois<br>"
4 :   $_SESSION["count"] = 0;
5 : } else {
6 :   echo "Vous␣avez␣déjà␣chargé␣" . $_SESSION["count"] . "␣pages<br>" ;
7 :   $_SESSION["count"]++
8 : }
```

Pour fermer la connexion : `session_destroy()`

Pour aller plus loin : à coupler avec une base de données

La semaine prochaine : JavaScript

http://benjamin.canou.fr/Cours/ENSTA/INE11

benjamin@ocamlpro.com