POLYTECH LILLE

Université Lille1 Sciences et Technologies

Rapport de création

Modifier et comprendre DataTable en profondeur

```php
/**
 * Transform a string with this format id:value,id:value... in an array
 * Return null if a column of id can't find
 * @param string $index index with this format: id:value,id:value
 * @return array|null
 */
private function parseIndex($index) {
    $error = false;
    $indexCSV = array();
    $allIndex = explode(',', $index);

    foreach ($allIndex as $aCSV) {
        $aIndex = explode(':', $aCSV);
        $name = $aIndex[0];
        $value = $aIndex[1];
        $indexCSV[$name] = $value;

        // Vérification que la colonne existe
        $col = $this->getCol($name);
        if ($col == null)
            $error = true;
    }

    if ($error)
        return null;
    else
        return $indexCSV;
}
```


Edouard COMTET

Génie Informatique et Statistiques - Promo 2014

Février 2014 sous la direction de Olivier CARON

Rapport de création

Modifier et comprendre DataTable en profondeur

1. Introduction

Lors de la création d'une partie d'administration, la plupart des développeurs doivent régulièrement mettre en place des tableaux pour la gestion (tableau des utilisateurs, des news, …).

Le but de ce framework est de permettre, en quelques lignes de code, de générer un tableau HTML/CSS/Javascript relié à une base de données. La modification du tableau sera, de manière automatique, répercutée dans la base.

Ce document est destiné aux développeurs qui devront modifier ou comprendre cet outil. En expliquant le fonctionnement interne de celui-ci et les passerelles entre les différentes parties, ce document permet d'en appréhender la complexité.

Dans la suite de ce document, on considèrera que le framework est installé dans un dossier du nom *DataTable* et que le fichier utilisant ce dernier est à la racine et se nomme *index.php*.

Ce doucement est rattaché à son homologue qui permet d'identifier les pré-requis nécessaire à l'utilisation de cet outil.

Ce framework se base sur DataTable, disponible à l'adresse http://datatables.net/. Le code source de cette amélioration est disponible depuis https://github.com/doudou34/DataTable.

1. Généralité

A. Exécution des requêtes SQL

Pour gérer l'exécution des requêtes avec PHP, le framework utilise PDO, une extension de PHP. Pour des raisons de sécurité, l'intégralité des requêtes sont d'abords préparé puis exécuté. Le connexion avec la base de données se fait grâce à la fonction *connect* de la classe *Table*, l'exécution des requêtes avec la fonction *prepareExecute* de la même classes qui prendra en paramètre la requête SQL et un tableau avec les paramètres de cette dernière. Cette dernière fonction s'occupera de vérifier la validité de la connexion à la base de données.

Ainsi, la plupart des fonctions qui créées une requête SQL, retourne un tableau associatif comprenant une partie *requete* et une seconde *parameters*.

B. Gestion de la clef primaire

Pour pouvoir identifier le tuple[1] de la base de données à modifier, il est nécessaire que le framework gère proprement la clef primaire d'une table. Pour cela, lors de l'affichage d'une ligne dans le tableau HTML, un attribut index est généré et se présente sous cette forme: index1:value1,index2:value2. La fonction *parseIndex* prends en paramètre une chaine de caractère et retourne un tableau comportant les index sous la forme:

Clef du tableau	Valeur
index1	value1
index2	value2

C. Modifier le style CSS

Ce framework se base sur le style CSS Bootstrap. Pour modifier le style par défaut, il faut écraser celui de *table.css*. Pour modifier le style CSS Bootstrap, voir sa documentation.

[1] Ligne dans la base de données, parfois confondu avec le terme ligne

2. Comment s'affiche une table

L'affichage d'une instance *Table* passe par l'appel à la fonction *show* qui prends pour seul paramètre: le chemin relatif où est stocké le framework par rapport au fichier d'utilisation. Ceci est utile pour pouvoir afficher des images ou faire appel au fichier *ajax.php* décrit plus bas.

Cette fonction va récupérer le type des champs, afficher le tableau, la fenêtre d'ajout puis sérialiser l'instance de *Table* dans un fichier. Nous nous intéresserons ,dans cette partie, qu'à la partie affichage du tableau et de la fenêtre.

A. Partie tableau HTML

Pour afficher le tableau HTML, le framework utilise la fonction *showTable*. C'est elle qui s'occupera de fournir la structure du tableau et délèguera l'affichage des tuples de la base de données à *showBody*.
Son premier rôle va être de construire un élément <table /> avec une succession d'attributs comme la langue à utiliser, le chemin relatif, son identifiant unique et les différentes options choisies par l'utilisateur (fonction CRUD, tri, pagination,…).

La fonction *showBody* va récupérer les données et, pour chaque tuple, construire son champ index, sauvegarder sa valeur dans un attribut et générer l'élément <tr />.

B. Récupération des données

La fonction showBody va utiliser *getData* et *getDataSQL* pour récupérer les données. Cette dernière fonction comporte 3 variables majeures. Une première qui enregistre la clause SELECT, une seconde pour la clause FROM et la dernière pour la clause WHERE qui prendra en compte les filtres et les relations clefs primaires / clefs étrangères. La construction de la requête est des plus classiques.

C. Partie fenêtre d'ajout

Les fonctions *showModal* et *showInput* permettent de générer une fenêtre qui s'affichera lors du clique sur le bouton d'ajout. Pour cela, le framework utilise

Bootstrap. La fonction *showInput* permet de créer un champ spécifique suivant le type de la colonne.

3. Comment fonctionne la partie javascript

Pour fonctionner, cet outil utilise la librairie jQuery. A la création d'un objet avec *$('table[dataTable]').dataTablePlus()*, la fonction *init* sera exécutée et une suite d'action sera déclenché (mise en place du CSS bootstrap, liaison des évènements avec ajax, …).

Parmi celle là, on peut noter l'utilisation de la fonction *send*, qui va envoyer grâce à AJAX une requête au fichier *ajax.php*. Les fonctions *setValue* et *getValue* permettent de gérer les types (suivant le type de la colonne dans la base de données, la manière de lui mettre une valeur ou de la récupérer sera différente). La fonction *transformCase* va mettre dans la cellule la bonne valeur (prise en compte notamment des booléens qui s'affichent avec des icônes).

4. Comment établir la liaison entre le PHP et le javascript

La liaison entre les deux langages, et qui donne son côté web 2.0 au framework, se fait grâce à ajax. Un fichier ajax.php récupère l'ensemble des requêtes et les exécute.

Pour garder le côté sécurisé d'une application, l'objet Table est sérialité et sauvegardé dans le répertoire temp grâce à la fonction serializeTable. La fonction checkTempFile permet de supprimer les fichiers plus vieux d'une heure (pour ne pas garder une historique trop important). Lorsque que la page est quitté par l'utilisateur, un dernièr évènement ajax est déclenché pour supprimer le fichier grâce à la fonction removeSerializedTable.

Chaque objet Table est identifié par un identifiant composé de sa place en mémoire et de la date de sa création. Pour chaque requête ajax, la partie javascript transmet l'identifiant, la fonction a exécuté et les paramètres.

Les opérations de mise à jour, de suppression et d'ajout se font par l'intémerdiaire des 3 fonctions publics du framework: updateData, deleteDate et createData. Les requêtes SQL étant un peu plus complexe à généré, deux fonctions, deleteSQL et createSQL permettent de générer les requêtes pour plus de clarté.

5. Comment ajouter un driver

Pour récupérer les différents types des champs, le framework s'adapte suivant selon le driver de la base (MySQL, PostGreSQL, …).

2 exemples de drivers sont à votre disposition:
- PGSQL qui corresponds à une base de données sous PostGreSQL.
- MySQL et Default qui correspondent à une base de données sous MYSQL.

Pour ajouter votre propre driver, la nouvelle classe doit étendre la classe *TableDriver* et vous devez modifier la fonction *setConcreteDriver*. Les différentes fonctions abstraites de la classe parente permettent de:
- *setTable*: permet de sauvegarder la $table pour l'exécution des requêtes
- *getTable*: retourne la table actuellement utilisé
- *getSQL*: retourne la fonction SQL a exécuté sous la forme d'un tableau associatif requête / paramètres.
- *getColumnName*: en lui passant un tableau, retourne le nom de la colonne pour cette ligne du tableau (en théorie le nom de la colonne correspondant dans la requête)
- *getColumnType*: même fonction que précédemment mais retourne le type de la colonne (en théorie le nom de la colonne correspondant dans la requête)
- *getColumnLength*: même fonction que précédemment mais retourne la longueur maximale de la colonne(en théorie le nom de la colonne correspondant dans la requête)

6. Comment ajouter une colonne

La gestion des colonnes est primordiales dans ce framework. Voici un guide pour ajouter une colonne.

Tout d'abords il faut créer votre nouvelle classe qui devra étendre une colonne existante ou alors la colonne de base: *Col.php*. Il vous faudra aussi rajouter votre fonction addCol dans la classe Table et les différents paramètres nécessaires.

La classe Col définit les fonctions élémentaires nécessaires au bon fonctionnement du framework. Rien ne vous empêche de rajouter des fonctions dans votre classe pour plus de clarté comme dans la classe ColLinked qui prends en compte la gestion des filtres. Il sera alors possible d'accéder à votre objet avec la fonction getCol de l'objet Table.

Voici une description des fonctions utilisé dans la classe Col:

- *setType*: enregistre le type de la colonne (voir la partie Gestion des types). Si le type est linked, la fonction getSelectOption sera utilisé.
- *getHeadLabel*: retourne le libellé de cette colonne
- *getHeadName*: retourne le nom de la colonne dans la base de données
- *getBody*: retourne la valeur pour cette colonne pour les données fournit (généralement retourne le nom de la colonne dans les données fournit).
- *getIndex*: Retourne une chaine de caractère contenant la valeur de l'index pour ce champ (vide si ce champ n'est pas une clef primaire).
- *getTables*: Retourne un tableau de chaine de caractère contenant les tables utilisés pour ce champ.
- *getField*: Retourne le champ que cette colonne doit afficher (dans la base de données).
- *getLinks*: Retourne une chaine de caractère contenant les différentes condition à mettre dans la clause WHERE lors de la sélection des données.
- *getSelectOption*: Retourne la requête SQL pour récupérer les différents possibilités lors de la modification ou de l'ajout de ce champ.
- *isVisible*: Retourne vrai si la colonne est visible par l'utilisateur
- *isCreatable*: Retourne vrai si lors de l'ajout d'une ligne, l'utilisateur doit renseigner ce champ
- *getValue*: Retourne la valeur de ce champ pour les données fournit

- *update*: Retourne la requête SQL pour mettre à jour ce champ (index et les différentes clefs primaires à utiliser pour identifier la ligne et new est la nouvelle valeur).
- *create*: Retourne la valeur de ce champ lors de la création (value est la valeur saisie par l'utilisateur).

7. Gestion des types

Les différents types des champs (text, varchar, …) sont gérés à plusieurs endroits du tableau. Si vous voulez rajouter la gestion d'un type au framework, voici la liste des endroits à prendre en compte:

- Dans le fichier *Table.php*, dans la fonction *showInput*, vous devez rajouter l'affichage pour votre type.
- Dans le fichier *table.js*, dans les fonctions setValue et getValue, vous devez rajouter votre type pour que le framework puisse récupérer sa valeur et la mettre à jour. Les fonctions prennent en paramètre l'élément jQuery où il faut récupérer ou mettre à jour sa valeur, le type du champ et pour la fonction setValue, la valeur à y mettre.

8. Bibliographie & Annexes

PDO
Bootstrap
DataTable
PHP
Jquery