

SERVEUR/CLIENT SUR IOS

Pier-Olivier Thibault

2 ans d'expérience iOS

@pothibo

<http://www.pothibo.com>

<https://github.com/pothibo>

Un aperçu de l'environnement

Un aperçu de l'environnement

- Objective-C, un superset de C

Un aperçu de l'environnement

- Objective-C, un superset de C
- Pas de garbage collection

Un aperçu de l'environnement

- Objective-C, un superset de C
- Pas de garbage collection
- Environnement fermé (Closed source)

Architecture

Architecture

- MVC Pattern
 - ✓ couche d'abstraction réseau
(ASIHTTPRequest, HTTPRiot, etc.)
 - ✓ *Caching*

Structure de notre app

- MVC Pattern
 - ➔ Model: myPhoto
 - ➔ Controller: ImageListController
 - ➔ View: ImageListCell

Structure de notre app

- ASIHTTPRequest
 - Populaire, stable
 - Supporte Blocks (>iOS 4.0)
 - Permet le *queueing* des requêtes
 - Injectable dans la structure MVC

MVC + Internet

- Controller
 - Requête pour la liste d'object (JSON)
 - JSON = rapide & léger
 - Affichage rapide des résultats
 - Les objets plus lourd peuvent être affiché de manière *asynchron*

MVC + Internet

- View

- Placeholder pour les images dans le cloud
- Requêtes pour les images (lourd)
- Mise-à-jour du contenu en temps réel

* Key-Value Observing

Key-Value Observing

- Le coeur et l'âme de iOS
- Basé sur les principes du Key-Value coding
- Semblable à NotificationCenter

- Coupler à la couche d'abstraction réseau
- Modèle se met à jour automatiquement
- Réduit les “points of failure”
- Code réutilisable
- Etc...

Code

myPhoto.h

- ➡ Model
- Controller
- View

Code

myPhoto.h

➔ Model

- Controller
- View

```
#import <Foundation/Foundation.h>

@class ASIHTTPRequest;

@interface myPhoto : NSObject {
    NSString *title;
    NSString *details;

    UIImage *image;

    @private
    ASIHTTPRequest *request;
}

@property (nonatomic, retain) NSString *title, *details;
@property (nonatomic, retain) UIImage *image;
@end
```

Code

ImageListController.h

- Model
- ➔ Controller
- View

Maintenant,
ImageListController.m...

Code

ImageListController.h

- Model
- ➔ Controller
- View

```
#import <Foundation/Foundation.h>

@class ASIHTTPRequest;

@interface myPhoto : NSObject {
    NSString *title;
    NSString *details;

    UIImage *image;

private
    ASIHTTPRequest *request;
}

@property (nonatomic, retain) NSString *title, *details;
@property (nonatomic, retain) UIImage *image;
@end
```

Maintenant,
ImageListController.m...

Ce que nous avons

Ce que nous avons

- Recherche les objets dans le cloud

Ce que nous avons

- Cherche les objets dans le cloud
- On les garde dans un NSArray

Ce que nous avons

- Cherche les objets dans le cloud
- On les garde dans un NSArray
- On affiche une cellule par objet

Ce que nous avons

- Cherche les objets dans le cloud
- On les garde dans un NSArray
- On affiche une cellule par objet
- Rien de bien compliqué...

Ce qui manque

- Nos objets sont vides.
- Aucune image n'est retournée
- Nos cellules affichent rien

Compléter le model

- - (id) initWithDictionary:(NSDictionary *)
 - Idéal pour initialiser
 - Variables (Et valider!)
 - Requête pour nos images au serveur

Ce que nous avons

- myPhoto est maintenant configuré
- Les images sont téléchargées
- Un *placeholder* est affiché
- Les cellules affichent la bonne information

Ce qui manque

- Les *placeholders* ne sont pas updatés
- Cellule toujours pas optimisé

Observers

- On se sert du Key-Value Observers pour “observer” les instances de notre modèle
- Quand les images sont rafraîchies, on devrait rafraîchir nos cellules
 - `(void)setNeedsDisplay;`

Comment observer?

myPhoto

```
@property (nonatomic, retain) UIImage *image;
```

Cellule

```
– (void)addObserver:(NSObject *)observer  
forKeyPath:(NSString *)keyPath options:  
(NSKeyValueObservingOptions)options context:(void  
*)context;
```

```
observer: cellule  
keyPath: @"image"  
options: NSKeyValueObservingOptionNew
```

UITableViewCell

- Une subclass permet plusieurs avantages
 - ✓ Permet de passer par référence l'objet au complet
 - ✓ Permet d'optimiser la cellule
 - ✓ Permet d'encapsuler KVO pour chaque cellule

ImageListCell

- Contient ImageListCellView
- Se fit sur TableViewSuite
- ImageListCellView devient le coeur du contenu de la cellule
- ImageListCell n'est qu'un *container* et un proxy.

ImageListCell

- Contient ImageListCellView
- Se fit sur TableViewSuite
- ImageListCellView devient le coeur du contenu de la cellule
- ImageListCell n'est qu'un *container* et un proxy.

```
#import <UIKit/UIKit.h>

@class myPhoto, ImageListCellView;
@interface ImageListCell : UITableViewCell {
    ImageListCellView *imageListView;
}
//Proxy
- (void)setPhoto:(myPhoto *)photo;
- (myPhoto *)photo;
@end
```

```
#import <UIKit/UIKit.h>

@class myPhoto;

@interface ImageListCellView : UIView {
    BOOL highlighted;
    myPhoto *photo;
}

@property (nonatomic, assign) myPhoto *photo;
@property (nonatomic, assign) BOOL highlighted;

@end
```

Maintenant

- On reçoit des objets en JSON du serveur
- De manière *asynchron*, on reçoit les images du serveur.
- Les cellules sont mises à jour à la réception des images
- Nos cellules sont encore rapide!

Résumé

- Bâti une application iOS
- Chercher de l'information dans le cloud avec *ASIHTTPRequest*
- *Parser* l'information et convertir en objet natif (*NSObject* + *JSON*)
- Afficher l'information en utilisant des principes du *Key-Value Observer*
- Optimisation de *UITableViewCell* pour garder le *scrolling* fluide

Libraries

- JSON-Framework
 - <http://stig.github.com/json-framework/>
- ASIHTTPRequest
 - <http://allseeing-i.com/ASIHTTPRequest/>

MERCI!

<https://github.com/pothibo/WaQ-Demo>