KVO , Delegate

Jaemin

KVO, Key Value Observing

- key-value observing이란 프레임 워크 수준에서 값에 대한 변경 사항을 추적해 주도록 하는 역할을 하는 매커니즘을 말한다. 어떠한 객체에 대한 키를 등록하면 이 키 값이 변경될 때 자동으로 옵저버에게 통보가 된다.

  출처: http://yujuwon.tistory.com/21 [Ju Factory]

- 해당 key의 value가 변할 때 통보받을 수 있는 방법

KVO, Key Value Observing

KVO(Key-value Observing)

정의

모델 객체의 어떤 값이 변경되었을 경우 이를 UI에 반영하기 위해서 컨트롤러는 모델 객체에 Observing을 도입하여 델리게이트에 특정 메시지를 보내 처리할 수 있도록 하는 것입니다.

특징

- 일대일, 일대다 관계에 대해서도 Observing을 적용할 수 있습니다.
- 모델 데이터에 반영되는 구조를 가진 앱은 코코아 바인딩을 사용하면 코드 작성을 최소화 할 수 있습니다.

예제

Observer로 등록하기

```
- (void)registerAsObserver {
    /*
     Register 'inspector' to receive change notifications for the "openingBalance" property
 of
     the 'account' object and specify that both the old and new values of "openingBalance"
     should be provided in the observe… method.
     */
    [account addObserver:inspector
              forKeyPath:@"openingBalance"
                 options:(NSKeyValueObservingOptionNew |
                          NSKeyValueObservingOptionOld)
                 context:NULL];
}
```

NSKeyValueObservingOptioinNew는 NSKeyValueChangeNewKey 키에 대한 새 값을 저장합니다.

NSKeyValueObservingOptionOld는 NSKeyValueChangeOldKey 키에 대한 이전 값을 저장합니다.

Observer로부터 통보 받기

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

    if ([keyPath isEqual:@"openingBalance"]) {
        [openingBalanceInspectorField setObjectValue:
            [change objectForKey:NSKeyValueChangeNewKey]];
    }
    /*
     Be sure to call the superclass's implementation *if it implements it*.
     NSObject does not implement the method.
     */
    [super observeValueForKeyPath:keyPath
                         ofObject:object
                           change:change
                          context:context];
}
```

Observer에서 제거하기

```
- (void)unregisterForChangeNotification {
    [observedObject removeObserver:inspector forKeyPath:@"openingBalance"];
}
```

```objc
-(void)viewWillAppear:(BOOL)animated{
    [super viewWillAppear:animated];

    [self.child1 addObserver:self forKeyPath:@"name" options:NSKeyValueObservingOptionNew |
        NSKeyValueObservingOptionOld context:nil];
    [self.child1 addObserver:self forKeyPath:@"age" options:NSKeyValueObservingOptionNew |
        NSKeyValueObservingOptionOld context:nil];
}
```

```objc
-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change
    context:(void *)context{

    if ([keyPath isEqualToString:@"name"]) {
        NSLog(@"The name of the child was changed.");
        NSLog(@"%@", change);
    }

    if ([keyPath isEqualToString:@"age"]) {
        NSLog(@"The age of the child was changed.");
        NSLog(@"%@", change);
    }

}
```

```
-(void)viewWillAppear:(BOOL)animated{
    ...

    [self.child1 setValue:@"Michael" forKey:@"name"];
}
```

```
The name of the child was changed.
{
    kind = 1;
    new = Michael;
    old = George;
}
```

Delegate

- Delegate 사용하는 목적
  - 특징적인 역할을 하는 함수들을 특정한 조건이나 의도를 가지고 묶음으로 만들어내 좀 더 간편하게 사용하도록 하기 위함
  - UIApplication과 같은 복잡한 객체를 상속하는 것을 피하고, 메소드를 재정의 하지 않음으로써 객체 지향적인 프로그래밍을 하기 위함

차이점

- KVO가 갖는 가장 큰 장점 : 값의 변경이 있을때마다 값 변경을 알리는 작업을 일일이 하지 않아도 된다

- Delegate는 이벤트가 발생 할 때마다 작업을 일일이 해줘야 한다