

Cocoaで 「設定よりも規約」

概要

- 「設定よりも規約」パターンをCocoaに適応
- メソッド名に規則性を持たせる事で記述量を減らす
- SEL型とNSString型の相互変換積極的に使う

目的

- 宣言的な記述による可読性の向上
- ドメイン特化言語的記述を行いたい場合

動機

RoRの「設定よりも規約」では、DBのテーブル名はクラスに、カラム名はフィールド名に自動でマッピングされている。これはXMLファイルで個々に設定するよりも、柔軟性は低いが十分有用な方法だ。

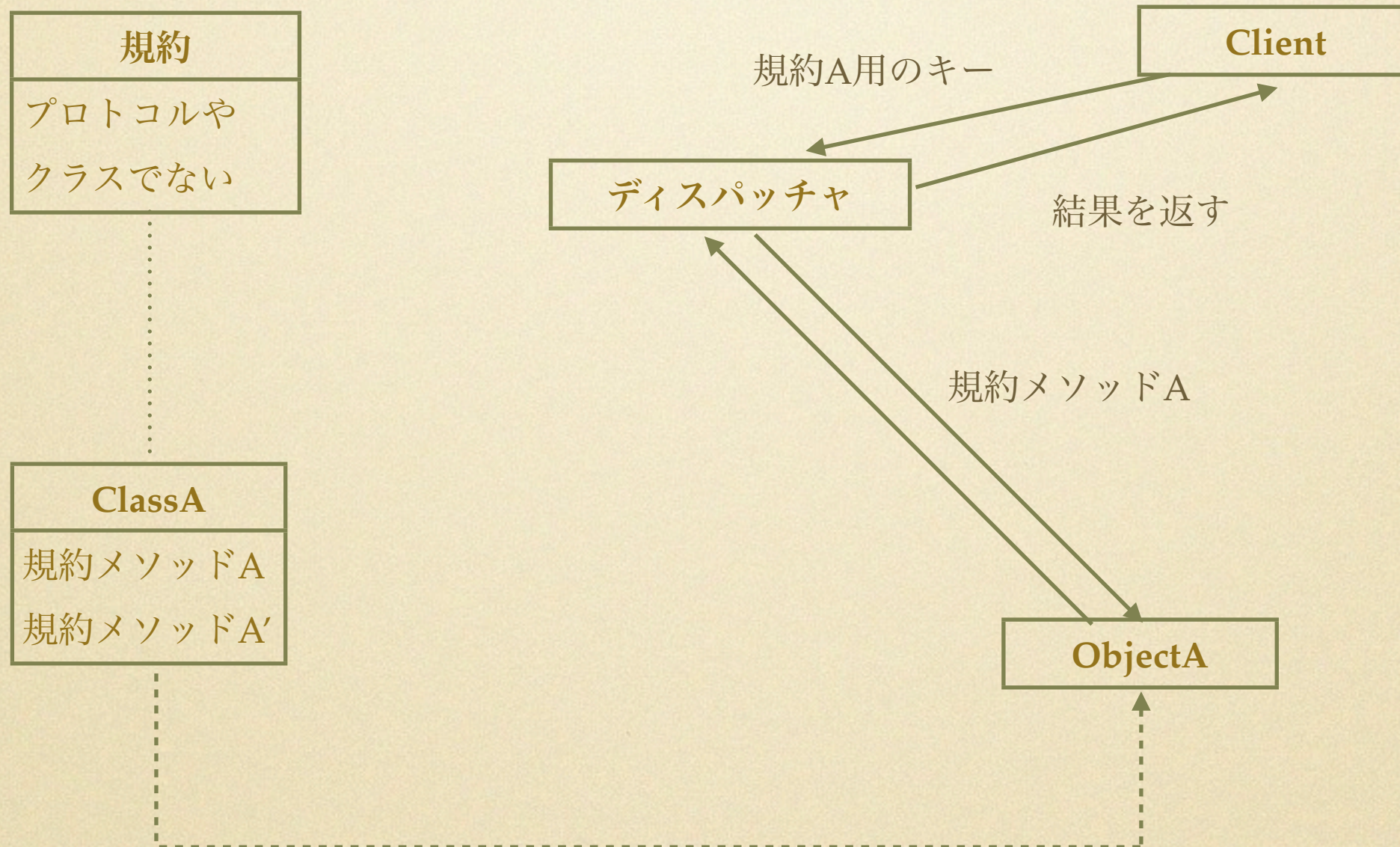
CocoaではKVCのアクセッサペアの命名規約が「設定よりも規約」にあたる。この規約によって、アクセスキーが実際にどの変数にアクセスするかの設定を書かなくても良くなっている。

何度も同じ事を書かない事でコードディングを省力化したいし、そのためにObjectに設定を読み込ませるような事もしたくない。

適用可能性

- コーディング量を減らしたい時
- メソッドの相互依存を減らしたい時
- 仕様変更による再コーディングを減らしたい時

構造



構成要素

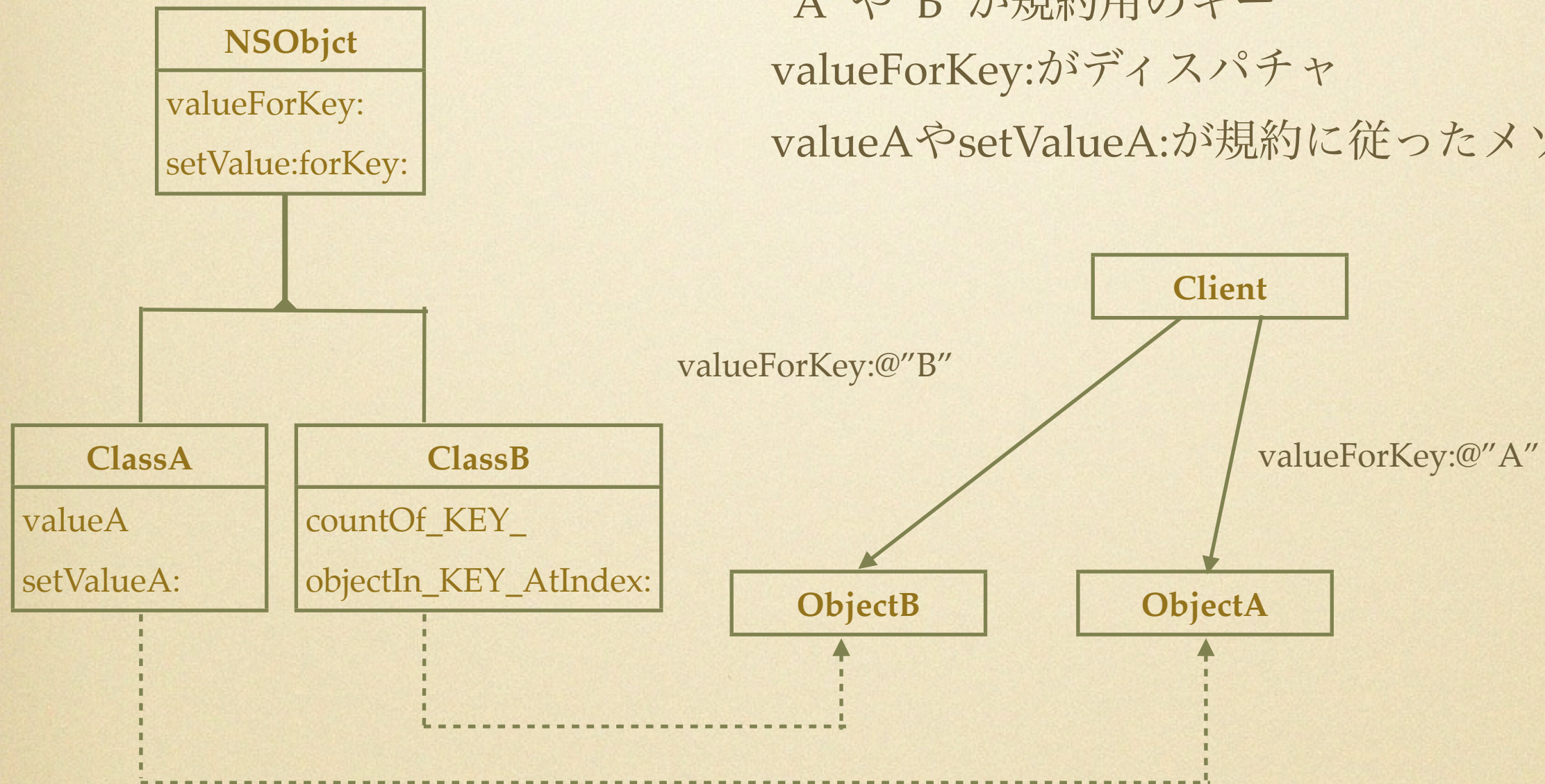
- 規約に従った名前が付けられたメソッド群
- デスパッチャに渡すキー(文字列やセレクタ)
 - クライアントオブジェクトが持つ
- メソッドを呼出す"ディスパッチャ"
- 単純な関数から、メソッド、クラス時もある

実際の例

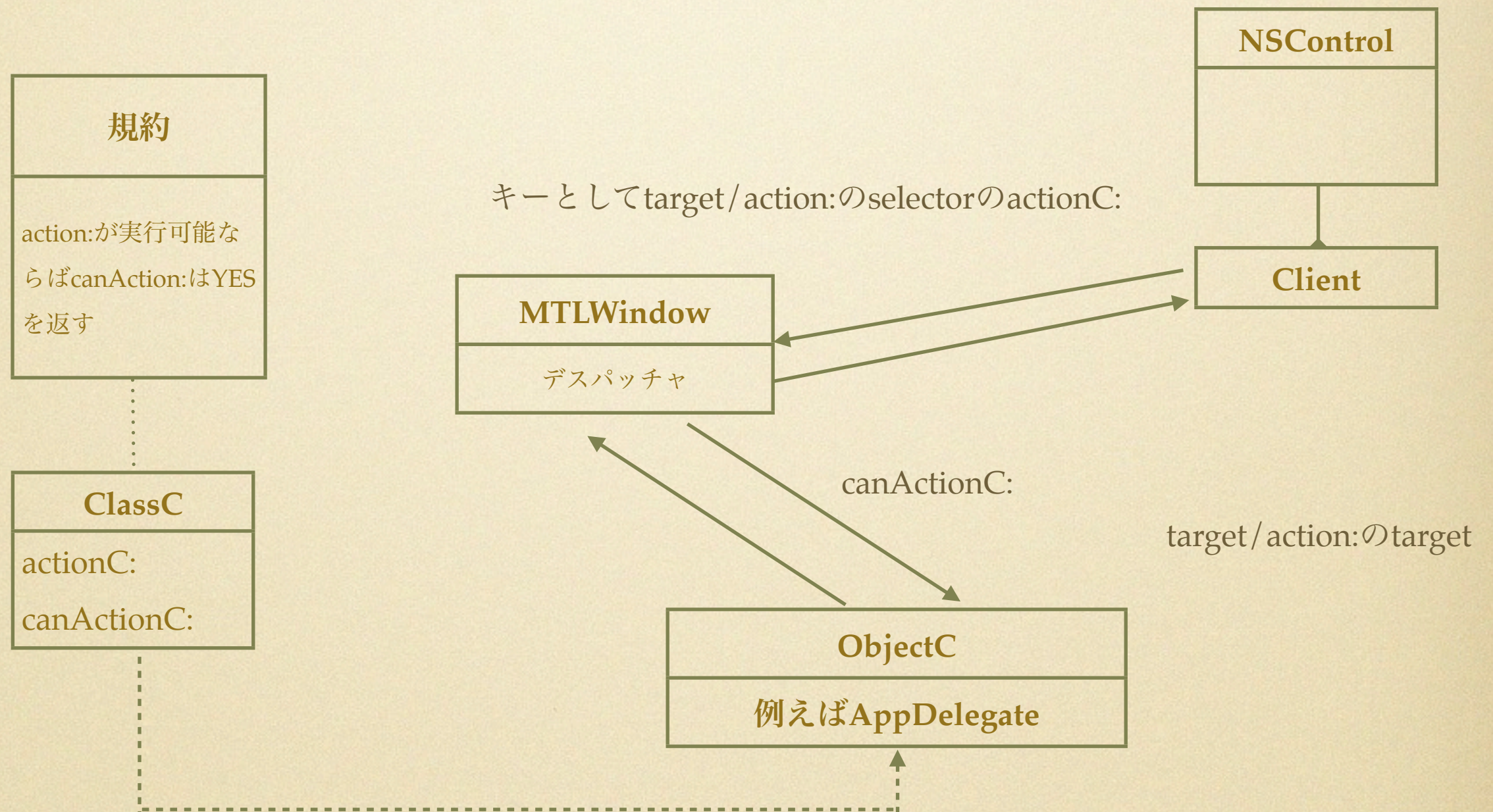
- KVC/KVO
- 成田のControlEnabler

構造(KVC)

“A”や“B”が規約用のキー
valueForKey:がディスパチャ
valueAやsetValueA:が規約に従ったメソッド



構造(ControlEnabler)



協調関係

- クライアントがデスパッチャにキーを渡す
- デスパッチャはキーからメソッド名を生成
- 生成したメソッド名を使ってメソッド群を呼出す

結果

- 定型的なコーディング量の減少と 可読性の向上
- 言語から conformsToProtocol 等の支援が無い
- インスタンス変数が減る
- コーディング規約から動作を想像しにくい
- 文字列操作が入る為に実行コストが高い

サンプルコード

- 成田の以前の 5 年後のControlEnablerを見てく
ださい
- キーリマッパーのアプリ側にもコードが入って
ます。

使用例

- AppleのKVC
- 成田のControlEnabler
- MSのVisualBasicのGUIとコードのbinding
 - UIエディターで設定したボタン名に対して、“onボタン名()”のようなメソッドを定義して振る舞いを設定できたらしい

応用 1

sender無しdelegate

- NSTableViewDelegate等でtableViewへのポインタをなくす事が出来る。nibでtableViewに名前を付けられるので、delegate側でポインタを持つ必要が無くなる
- - (void)**tableViewName**WillDisplayCell:(id)cell
forTableColumn:(NSTableColumn *)tableColumn
row:(NSInteger)row;
 - (BOOL)**tableViewName**ShouldEditTableColumn:(NSTableColumn *)tableColumn
row:(NSInteger)row;

妄想です！

応用 2

KVCの先行評価や遅延評価の

- valueForKey:やsetValueForKeyを上書きして、アクセスの前後や変更の前後に規約に従ったメソッドを呼び出すようにする。
- モデルクラスを書く時に、以下のような規約があると遅延評価や先行評価の仕組みを組み込みやすくなる。

- (BOOL) willChangeValueFor_KEY_;
- (BOOL) didChangeValueFor_KEY_;
- (BOOL) willAccessValueFor_KEY_;
- (BOOL) didAccessValueFor_KEY_;

妄想です！

関連するパターン

- DSLを作成する意味ではインタプリタパターンと似ている

終わり