CloudKit I.

(zatím ne zcela dořešené koncepty použití)

Martin Hrubý
Seminář iOS, FIT VUT v Brně

Úvod

- Představen na WWDC 2014
  - Základní popis funkcionality
  - iCloud DB Server — kontejnery, entity, záznamy
  - Uživatelé konkrétní aplikace mohou sdílet data — **jak jim to aplikace umožní**
- Zařazen do IOS 8
- Velmi specifická (až neobvyklá) koncepce
  - Časem se uvidí, jak CK vývojáři pochopí a integrují do svých aplikací

Co CloudKit *NENÍ*

- Databázový server SQL typu
  - Vybavovací doby jsou příliš velké
  - Infrastrukuta CKRecord zřejmě nepředpokládá vetší přenosy dat v jednom balíku
  - Přístupová práva — **všechno řídí aplikace**
- Univerzální platforma pro mobilní aplikace (omezení zatím na iOS/OS X).

K čemu může být CK

- Informační systém pro malé skupiny uživatelů (firmy).
- Sociální síť.
- Zálohovací server pro uživatele (dat z jeho aplikace)
  - Jak je privátní kontejner spojen s instalací aplikace?

Databáze v kontejneru

- Privátní — z názvu plyne, že je individuální pro každého uživatele. Limitován velikostí iCloud prostoru uživatele.
  - Uživatel musí mít iCloud přístup.
- Veřejný (public) — zde mohou uživatelé sdílet záznamy. Masivní kvóta.
  - Pro čtení netřeba iCloud přístup. Pouze pro zápis.
  - Kvóta roste s počtem zapojených uživatelů.

Zóny

- Další “složka” pro organizaci CKRecords:
  - Kontejner -> Databáze -> Zóna
- Private DB — Default + Custom zones
- Public DB — Default
  - Public DB neumožňuje vlastní zóny !!! Tady by se zrovna hodily. Nutno implementovat explicitním atributem záznamu.

Záznam (CKRecord)

- Má svůj typ (recordType) — NSString.
  - typ by měl mít charakter DB Entity, tj. nějakou koncepci.
- Charakter NSDictionary s omezeným rozsahem typů pro value (key-value) — protokol CKRecordValue.
- Záznam lze:
  - save (insert, update), fetch (**znám** ID), query (neznám ID), delete.
  - metadata — datum vytvoření, uživatel.

Hodnoty atributů CKRecord

- setObject:forKey:
  - NSString, NSDate, NSNumber, NSArray
  - NSData — ovšem maké kousky
  - CLLocation — geografická souřadnice WGS84
  - CKAsset — souborová příloha
  - CKReference — vazba na jiný CKRecord (ve stejném kontejneru)

Instanciace CKRecord

- initWithRecordType:
- initWithRecordType: recordID: (CKRecordID *)
  - RecordID je generováno rnd nebo uživatelské.
- Jak zacházet s CKRecord:
  - Berme CKRecord jako **dočasný objekt** pro komunikaci s CK
  - recordID — jednoznačná identifikace záznamu. Lze ji uložit (.recordName) do lokálních dat aplikace a pak použít pro další operace (fetch, delete).

CK Databáze je “objektová”

- CKRecord je identifikován — .recordID
- Vazby (reference) mezi CKRecords.
  - Hodnota CKReference (typ vlastnictví)
  - Vazby 1:N (NSArray s CKReference *)
- Nepřehánět to s provázaností mezi objekty.
- Atribut určující verzi DB modelu.

Smysl privátní databáze

- CKServerChangeToken — udělá značku v private kontejneru nad stavem záznamů
  - pak lze provádět rozdílové operace
  - zjisti změny: obdrží seznam změn, ty lze zapracovat do stavu aplikace
- Vzdálené odkladiště záznamů jednoho uživatele
  - Aplikace v kopiích na iOS, OS X zařízeních může synchronizovat svá data s jejich vzdáleným uložením
  - Lze určitě udělat jinak — např. iCloud Core Data

Ověření uživatelova přístupu

- Ubiquity token. Může se měnit. Sledovat v mainThread.

```
NSFileManager* fileManager = [NSFileManager defaultManager];

id currentiCloudToken = fileManager.ubiquityIdentityToken;


[[NSNotificationCenter defaultCenter]

    addObserver: self

       selector: @selector (iCloudAccountAvailabilityChanged:)

           name: NSUbiquityIdentityDidChangeNotification

         object: nil];
```

Smysl public databáze

- Nelze dělat značky (ServerChangeToken)
  - Nelze aktuálně odhadnout provoz/cvrkot, značky jsou nejspíš technicky nerealizovatelné.
  - Navíc asi nelze smysluplně zformulovat smysl rozdílové operace ve více-uživatelském prostředí.
- Public DB berme jako ***nástěnku*** pro *všechny* uživatele *jedné* aplikace
  - Berme CK-public jako perzistentní komunikační kanál.
- Lze rychle přesouvat CKRecs mezi databázemi?

Režimy “nástěnky”

- Přihlásit se jako observer změn v databázi, pak přijímat Push notifikace.
  - Může být značný tok notifikací. Lze **garantovat** korektní zpracování všech notifikací?
  - Model: chatovací aplikace — broadcast události v CK.
- Občasný refresh stavu tabulek.
  - Mazání záznamů.
- Ne-perzistence na straně čtenáře záznamu.

Můj doporučený koncept

- CK předpokládá nějakou formu **lokální kopie dat**, např. Core Data.
  - Musíme spravovat DVĚ databáze: CK a CoreData.
- Lokální data => dává smysl aplikaci provozovat bez připojení k Internetu.
  - V přechodu do online režimu provést formu synchronizace.
- V public je nutno **definovat vlastníka záznamu**.
  - … dle charakteru aplikace.

Původce a čtenář

- Původce vytvoří záznam. Je zodpovědný za jeho smazání z CK. Má zápisová práva.
  - lokálně (CD) — má atribut NSString pro ref. na CKRecord a Status objektu (exportován, modifikován, smazán).
  - Synchronizace — odeslání změn. Triviální.
- Čtenář — synchronizace. Komplikovaná.
  - Nepředpokládá se update záznamu.
  - Query. Subscription na smazání.

Základní konstrukce

- V XCode Capabilities — iCloud
  - jméno kontejneru — iCloud.eu.domena.appNazev
  - V simulátoru může haprovat (přihlašte svůj AppleID)
- Inicializace CK API
- Vytvoření CKRecord
- Provádění dotazů/příkazů na CK

Inicializace API

```
-(BOOL) initCloudKitWithContainerID: (NSString *) containerID
{
    self.ckContainer = [CKContainer containerWithIdentifier:
containerID];
    if (_ckContainer == nil)
        return NO;

    self.ckDBPrivate = [_ckContainer privateCloudDatabase];
    self.ckDBPublic = [_ckContainer publicCloudDatabase];

    return YES;
}
```

Vytvoření CKRecord

- Instanciace objektu CKRecord
- Naplnění daty
- Provedení přístupové operace:
  - Řada přístupových operací.
  - Jsou to NSOperation!
  - Sledování callback bloků a NSError

Operace nad daty

- Fetch
  - záznamů — rozdílová (Changes), normální.
  - subscriptions.
- Modify — záznamy, subscriptions.
- Query.
- User info — specifické objekty o uživatelích.
- Notifikace — fetch, mark.

Simple varianta

```
CKRecord *rec = [[CKRecord alloc] initWithRecordType: @"recType1"];

[rec setObject: @"Pepa" forKey: @"name"];
[rec setObject: [NSDate date] forKey: @"narozen"];

[self.ckPublic saveRecord: rec completionHandler:^(CKRecord *record,
NSError *error) {
        NSLog(@"Saved %@", error);
    }];
```

Uživatelské RecordID

```
CKRecordID *appDefID = [[CKRecordID alloc] initWithRecordName:
@"Hlavnipolozka"];
CKRecord *appDef = [[CKRecord alloc] initWithRecordType: @"Main"
recordID:appDefID];
[appDef setObject: @"Martin" forKey: @“jmeno"];

[self.ckPublic saveRecord: appDef completionHandler:^(CKRecord *record,
NSError *error) {
        NSLog(@"Record AppDef, error: %@", error);
    }];
```

Komplexní varianta

```
CKModifyRecordsOperation *op = [[CKModifyRecordsOperation alloc]
initWithRecordsToSave: @[rec] recordIDsToDelete: @[]];

    op.completionBlock = ^{
        NSLog(@"Operace komplet dokoncena");
    };

    op.perRecordCompletionBlock = ^(CKRecord *record, NSError *error) {
        NSLog(@"Zaznam proveden, chyba: %@", error);
    };

    op.modifyRecordsCompletionBlock = ^( NSArray *savedRecords, NSArray
*deletedRecordIDs, NSError *error) {
        NSLog(@"Completed uloz: %ld smaz: %ld", savedRecords.count,
deletedRecordIDs.count);
    };

    [self.ckPublic addOperation: op];
```

Query

```
-(void) fetchRecords1
{
    // MUSI byt zadan
    NSPredicate *qPredicate = [NSPredicate predicateWithValue: YES];
    CKQuery *query = [[CKQuery alloc] initWithRecordType: @"recType1"
                                                predicate: qPredicate];

    CKQueryOperation *op = [[CKQueryOperation alloc] initWithQuery:
query];

    op.recordFetchedBlock = ^(CKRecord *rec) {
        NSLog(@"Natazen objekt: %@", [rec objectForKey: @"name"]);
    };

    op.queryCompletionBlock = ^(CKQueryCursor *cursor, NSError
*operationError){
        self.ckCursor = cursor;
    };

    [self.ckPublic addOperation: op];
}
```

Query+cursor

```
op.queryCompletionBlock = ^(CKQueryCursor *cursor, NSError
*operationError){
        if (cursor != nil) {

            dispatch_async(dispatch_get_main_queue(), ^{
                [self fetchRecords1C: cursor];
            });
        }
    };

-(void) fetchRecords1C:(CKQueryCursor *)withCursor
{
    CKQueryOperation *op = [[CKQueryOperation alloc] initWithCursor:
withCursor];

    op.recordFetchedBlock = ^(CKRecord *rec) {
        NSLog(@"Natazen objekt: %@", [rec objectForKey: @"name"]);
    };

    [self.ckPublic addOperation: op];
}
```

Vazby mezi objekty

```
CKRecord *rec1 = [[CKRecord alloc] initWithRecordType: @"recType1"];
CKRecord *rec2 = [[CKRecord alloc] initWithRecordType: @"recType2"];
CKReference *rec2ref = [[CKReference alloc]
  initWithRecord: rec2 action: CKReferenceActionNone];

    [rec1 setObject: @"Honza" forKey: @"name"];
    [rec1 setObject: rec2ref forKey: @"studuje"];

    [rec2 setObject: @"FIT" forKey: @"fakulta"];
    [rec2 setObject: @"VUT" forKey: @"skola"];

    CKModifyRecordsOperation *op = [[CKModifyRecordsOperation alloc]
initWithRecordsToSave: @[rec1, rec2] recordIDsToDelete: @[]];

    [self.ckPublic addOperation: op];
```

Subscription

- CKSubscription — perzistentní query na straně serveru. Posílá push-notifikace (povolení notifikací).
- Subscriptions:
  - Globální pro aplikaci (zavedou se při prvním startu aplikace).
  - Zaměřené na zadaný NSPredicate.
  - Jsou to objekty (záznamy) v databázi! Hodilo by se je po upotřebení mazat.
  - Začnou fungovat po zaindexovaní serverem.

Zavedení subs

```
NSUserDefaults *ud = [NSUserDefaults standardUserDefaults];
[ud registerDefaults: @{ FIRST_RUN : @(YES)}];

if ([ud boolForKey: FIRST_RUN]) {
  [self registerDeleteSubscr: @"recType1"];
}

-(void) registerDeleteSubscr:(NSString *)recType
{
    CKSubscription *delS = [[CKSubscription alloc] initWithRecordType:
recType predicate: [NSPredicate predicateWithValue: YES]
  options: CKSubscriptionOptionsFiresOnRecordDeletion];

    [self.ckPublic saveSubscription:delS
completionHandler:^(CKSubscription *subscription, NSError *error) {
        NSLog(@"Subscription zaveden: %@", error);
        if (error == nil) {
            NSUserDefaults *ud = [NSUserDefaults
             standardUserDefaults];
            [ud setBool: NO forKey: FIRST_RUN];
            [ud synchronize];
        }
    }];
}
```

Přijetí push-notifikace

- Aplikace je musí chtít přijímat.
- NotificationInfo.

Rozdílový Fetch

- Pouze v privátní DB, v custom zóně
- Vytvoření zóny
- Zjištění předchozího ChTokenu
- Provedení rozdílového Fetch
- Uložení nového ChTokenu

Vytvoření zóny

```
CKRecordZone *zonePok = [[CKRecordZone alloc] initWithZoneName: @"Pok"];

[self.ckPrivate saveRecordZone:zonePok completionHandler:^(CKRecordZone
*zone, NSError *error) {
        NSLog(@"Save zone, error %@", error);
    }];
```

Předchozí token, UDefaults

```
NSUserDefaults *ud = [NSUserDefaults standardUserDefaults];
NSData *dt = [ud dataForKey: @"changeToken"];
CKServerChangeToken *oldToken = nil;

if (dt != nil)
  oldToken = [NSKeyedUnarchiver unarchiveObjectWithData: dt];
```

```
CKRecordZone *zone = [[CKRecordZone alloc] initWithZoneName: @"Pok"];
CKFetchRecordChangesOperation *op = [[CKFetchRecordChangesOperation
alloc] initWithRecordZoneID:[zone zoneID] previousServerChangeToken:
oldToken];

    op.recordChangedBlock = ^(CKRecord *rec) {
        NSLog(@"New/Modified object %@", rec);
    };

    op.recordWithIDWasDeletedBlock = ^(CKRecordID *recID) {
    };

    op.fetchRecordChangesCompletionBlock = ^(CKServerChangeToken
*serverChangeToken, NSData *clientChangeTokenData, NSError
*operationError) {

        if (operationError == nil) {
            [ud setObject: [NSKeyedArchiver archivedDataWithRootObject:
serverChangeToken]
                   forKey: @"changeToken"];
            [ud synchronize];
        } else {
            NSLog(@"Error %@", operationError);
        }
    };

    [self.ckPrivate addOperation: op];
```

CKCDManagedObject

- Bázová třída pro CK-CD entity. Režijní attr.
- Vytvoření CD objektu.
- Synchronizace:
  - všechny nové a modifikované.
  - všechny označené jako smazané.
  - vytvoří se seznamy CKRecord

```
-(CKRecord *) ckcdCreateCKRecord
{
    CKRecord *rec = nil;

    if (self.ckcdRecordName == nil) {
        rec = [[CKRecord alloc] initWithRecordType: [self
ckcdEntityName]];
    } else {
        rec = [[CKRecord alloc] initWithRecordType: [self
ckcdEntityName] recordID: [self ckcdGetCKRecordID]];
    }
    for (NSString *key in [self ckcdKeys]) {
        id objectAtKey = [self valueForKey: key];
        if ([objectAtKey isKindOfClass: [CKCDManagedObject class]]) {
            // je to reference
            CKCDManagedObject *ckcdReferencedObject = objectAtKey;
            CKRecordID *refID = [[CKRecordID alloc] initWithRecordName:
ckcdReferencedObject.ckcdRecordName];
            CKReference *ref = [[CKReference alloc] initWithRecordID:
                                    refID action: CKReferenceActionNone];
            [rec setObject: ref forKey: key];
        } else
            [rec setObject: objectAtKey forKey: key];
    }

    return rec;
}
```

Závěr I. dílu CK

- Private/Public DB
  - Rozdílový fetch v private DB
  - Práce v public DB
- Vždy se očekává lokální kopie dat
- Je třeba vytvořil robustní modely činnosti programu v různých režimech aplikací
  - Implementace náhražky rozdílového fetch nad public DB