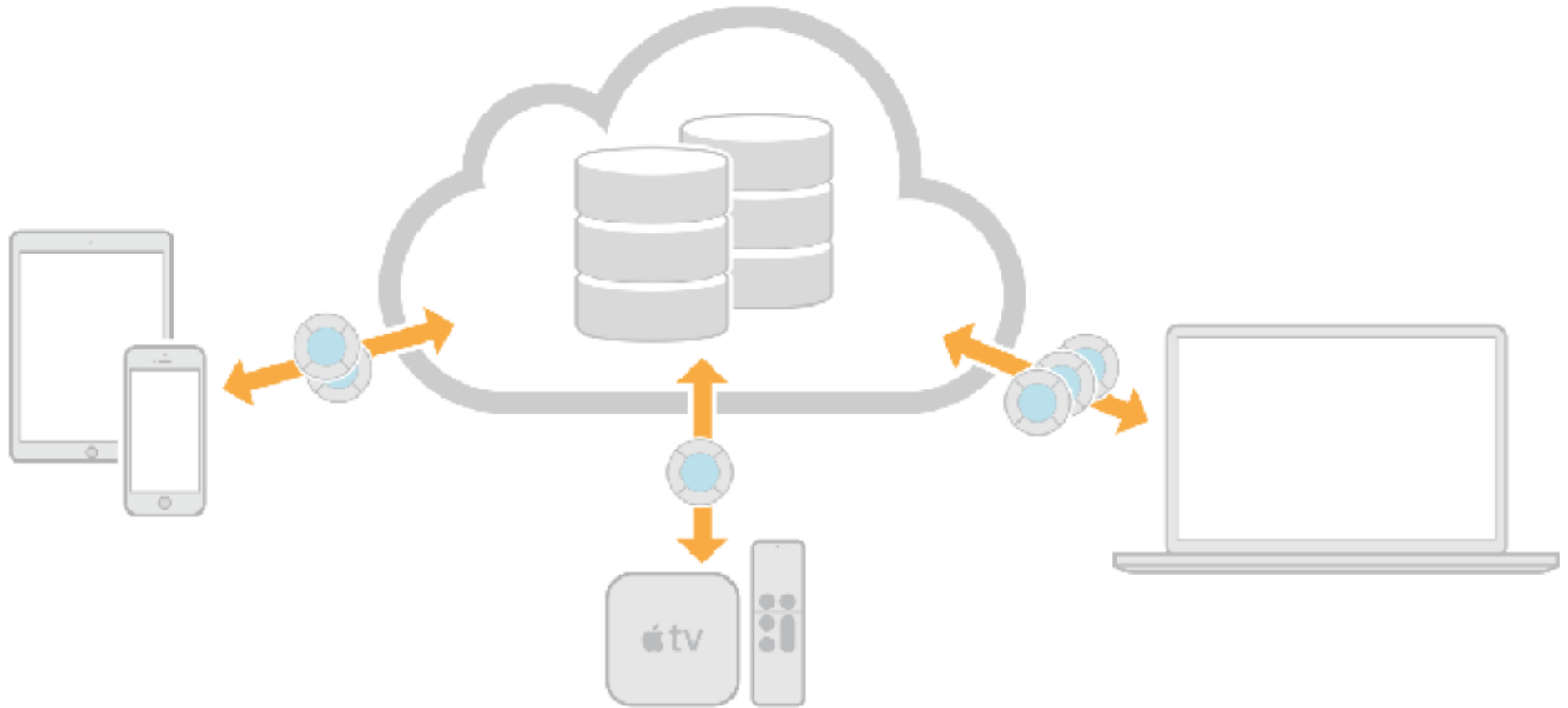


CloudKit



IZA, Martin Hrubý, FIT VUT, 2018

Přehled o ukládání dat

- (Flat) Soubory, (řízeno OS) dokumenty, Cloud documents, UserDefaults.
- CoreData — záznamy. Cloud CD.
- CloudKit — private / public / shared DB.
 - Key-Value Store.

DB nebo Dokumenty?

- Dokument — sada objektů, které k sobě patří.
Dokumenty nemají průnik objektů.
 - Je implementován souborem.
 - Životní cyklus dokumentu.
 - Může být exportován/importován na iCloud Drive.
 - Uvažujeme, že by dokument někdy mohl opustit rámec aplikace.
- DB — objektová paměť, neomezeně provázaná referencemi.

Jak koncepčně uložení...

- Potřeba pracovat s daty nějakého charakteru.
 - *Konstantní* — vytvořit pole v kódu, soubor v Bundle (načíst).
 - *Dynamická, malá, bez cloudSync* — soubor v sandbox / Documents, UserDefaults. Model s charakterem FRC.
 - *... s cloudSync* — např. Key-Value Storage (cloud UD).
 - *Dynamická, potenciálně velká, aplikačně interní, sync* — CloudKit/CD.
 - *... exportovatelná* (např. iCloudDrive) — Documents.

Notification Center

- Způsob komunikace objektů v aplikaci:
 - Přímé posílání zpráv — synchronní / asynchronní.
 - Zprávy posílané delegátovi prostřednictvím protokolu.
 - *Broadcast* — posílání zprávy celé aplikaci bez znalosti posluchačů (observers). Notifikace.
- Notification — centrum, observer, zpráva.

Notifications

- NotificationCenter — správce registrací observers, cíl pro posílání notifikace. Default.
- addObserver — název, zdrojový objekt, cílový objekt (ref, selektor).
- addObserverForName:object:queue:usingBlock:
- removeObserver — observer se musí explicitně odhlásit (deinit). Vhodný okamžik se odhlásit.
- post — poslání notifikace. Broadcast na posluchače *typicky probíhá vláknem*, kterým se zpráva iniciuje (post).

Broadcast, implementace

- Prostě synchronní — volá se post, posluchačům se v cyklu rozesílá notifikace.
- GCD — zpráva posluchačům se vkládá do DispatchQueue.
 - pak lze async.

UserDefaults

- Foundation.
- `UserDefaults.standard` — singleton UD:
 - persistence aplikačních Key-Value záznamů.
 - překlad parametrů příkazové řádky —key Value
 - `synchronize()` — uložení.
 - factory defaults.
- `Settings.bundle` — konfigurace app Settings pomocí plist.

UserDefaults

- register(defaults registrationDictionary: [String : Any])
- funkce pro čtení: string(forKey:) apod
- funkce pro zápis: set(_ value:String, forKey:)
 - data, dictionary, array

UD, implementace

- Zřejmě se exportuje / importuje NSCodering způsobem jeden soubor (do sandboxu app).
- Není synchronizováno mezi zařízeními.
- Vhodné pro menší kusy dat.
 - Kontext života aplikace (spuštěno poprvé apod.).
 - change token, přístupové kódy, ...
 - konfigurace.

Cloud Key-Value Storage

- Key-Value objektový kontejner: array, string, bool, data, dictionary, ...
- Zařízení musí mít přihlášen iCloud.
 - Aplikace musí registrovat požadavek na *ubiquity-kvstore-identifier*
- 1MB celkový limit. Key max 64B délka.
- Lokální soubor, synchronizace.

Zápis key-value

```
// API na key-value storage
let defs = NSUbiquitousKeyValueStore.default

// zapis
defs.set(Date().description, forKey: "mujdatum")
defs.set("ahoj", forKey: "pozdrav")

//
struct MData: Codable {
    //
    let name: String
    let age: Int
}

//
let person = MData(name: "Pepa", age: 123)
let encoded = try! JSONEncoder().encode(person)

// vlozeni objektu typu Data()
defs.set(encoded, forKey: "mdata")

// explicitni pokyn k zahajeni synchronizace K-V-Storage
defs.synchronize()
```

Aktualizace KV

```
// blok pro obsluhu zpravy od KV
// 1) doslo ke zmene KVS
// 2) posledni synchronizace havarovala
@objc func update(val:NSNotification) {
    //
    print("Udalost na key-value \(val)")
}

//
func startup() {
    // API na key-value storage
    let defs = NSUbiquitousKeyValueStore.default
    let nName =
NSUbiquitousKeyValueStore.didChangeExternallyNotification

    //
    let defNC = NotificationCenter.default
    // registrace
    defNC.addObserver(self,
                      selector: #selector(update(val:)),
                      name: nName,
                      object: defs)
}
```

Příklad notifikace

```
Udalost na key-value NSConcreteNotification 0x174245c40 {name =
NSUbiquitousKeyValueStoreDidChangeExternallyNotification; object =
<NSUbiquitousKeyValueStore: 0x17008f320>; userInfo = {
    NSUbiquitousKeyValueStoreChangeReasonKey = 0;
    NSUbiquitousKeyValueStoreChangedKeysKey =      (
        mujdatum
    );
}}
```

CloudKit

- Dokumenty, iCloud synchronizace.
- CoreData (SQLite3 db soubor), iCloud sync.
 - Technické potíže. *Co se vlastně synchronizuje?*
 - Alternativní DB kontejnery (Dropbox DB-API).
- CloudKit — *motivace, klient-server aplikace, kde server technicky zajišťuje iCloud.*
 - Má ovšem private sekci, kde lze úspěšně implementovat synchronizaci (dokumenty, CD).

CloudKit

- CK je tedy architektura Client—Server.
- iCloud (server) je centrálním prvkem synchronizace.
 - Latence. Kvóty.
 - Tzn.: existuje konkrétní naprosto jasný způsob implementace synchronizace dat (na rozdíl od CoreData v iCloud).
- Programátor má možnost určit způsob provedení synchronizace dat mezi aplikacemi.

Způsob používání CK

- Rozhodně (!!!) se nepředpokládá konvenční Client-Server přístup jako "tenký klient".
- Rozhodně (!!!) se očekává, že app tvoří lokální kopii dat a přenáší pouze změny.
 - uvažujme cílovou platformu iPhone a mobilní data.
 - Latence a kvóty. Veškeré DB operace jsou NSOperation.
- Programy Apple (v macOS, iOS) jsou založeny na CK (např. Notes).

Způsob používání CK

- CK jako správa dokumentů — obdoba UIDocument.
- CK jako synchronizační prvek CoreData lokálních uložišť.
- CK jako archivační medium.
- CK jako nástěnka pro sdílení dat uživatelů aplikace (public/shared).
 - skupina uživatelů — např. firemní aplikace.

CloudKit

- Aplikace musí mít oprávnění (XCode).
- Založí se vzdálený kontejner — sdílení různými aplikacemi.
- V kontejneru se založí 2 (3) DB:
 - privátní — speciální sync režim.
 - public — volný přístup všech uživatelů. Nelze diffFetch.
 - shared — omezení přístupu na vybrané uživatele.
- Kvóty při přístupu. Kapacita DB.

API CloudKitu

- CKContainer — kontejner zřízený aplikací.
- CKDatabase — private, public.
- CKRecordZone — složka v DB.
- CKRecord — záznam do DB. CKAsset.
- CK Operace — přístup přes NSOperation.
- Metadata — uživatel, nastavení události (subscriptions).

CKRecord

- Jeden DB záznam (Entita).
- Berme CKRecord jako dočasný objekt pro přenos dat aplikace -> iCloud, iCloud -> app:
 - založíme CKRecord pro nový záznam (dostane recordID),
 - ... pro aktualizaci (určíme recordID),
 - obdržíme CKRecord jako výsledek query / fetch.
- CKRecordID — recordName, zoneID.

Aplikace s dokumenty v CK

- Koncept vychází z UIDocuments.
 - Datová složka,
 - třída s řídicími metodami — save, load.
- Metadatové operace — fetch všech záznamů, verzování.

Demo, datová složka

```
//  
struct MyCKDocumentCont : Codable {  
    //  
    var text: String  
    var notes: [String]  
}
```

ICloud.eu.cdplan-solutions.CKDocumentsIZA > Development Data

MARTIN HRUBÝ

ZONES RECORDS RECORD TYPES INDEXES SUBSCRIPTIONS SUBSCRIPTION TYPES SECURITY ROLES

DEFAULT TYPES

Users

CUSTOM TYPES

DocRecordType

Record types will be automatically created in the development environment when you use the native API to create records of that type.

Create New Type

| Field Name | Field Type | Indexes | |
|------------|------------------------|-----------------------------------|---|
| docAsset | Asset | | X |
| docData | Bytes | View Indexes > | X |
| docName | String | Queryable | X |
| recordName | SYSTEM FIELD Reference | Queryable | |
| createdBy | SYSTEM FIELD Reference | | |
| createdAt | SYSTEM FIELD DateTime | | |
| modifiedBy | SYSTEM FIELD Reference | | |
| modifiedAt | SYSTEM FIELD DateTime | | |
| changeTag | SYSTEM FIELD String | | |

Add Field

Created: Apr 8 2018 11:14 AM by _1d80972c9627783c919494b1885b96ac
Modified: Apr 8 2018 11:58 AM by _1d80972c9627783c919494b1885b96ac

Delete Record Type Save Record Type

Demo, třída CKDocument

```
// CloudKit Document
class MyCKDocument {
    // datova cast
    var content: MyCKDocumentCont
    // metadata: nazev dokumentu (idealne unikatni)
    let name: String
    // ckrecordId pro ucely aktualizace save:update
    var ckRecordID: CKRecordID? = nil
    // ...
    var ckAsset: MyCKDCAsset? = nil
    // handle na privatni DB CK kontejneru
    var db: CKDatabase {
        //
        return CKContainer.default().privateCloudDatabase
    }
}
```


CKRecord — create, update

```
// instanciacie CKRecord. Bud nova nebo
// se znamym recordID
var ckRecordSave: CKRecord {
    //
    if let _id = ckRecordID {
        //
        return CKRecord(recordType: "DocRecordType", recordID: _id)
    } else {
        // _defaultZone
        return CKRecord(recordType: "DocRecordType")
    }
}
```

Demo, instanciacie

```
// instanciacie, vytvoreni dokumentu
  init(name: String) {
    //
    self.name = name
    self.content = MyCKDocumentCont(text: "", notes: [])
  }
// instanciacie, z ulozenych CKRecord dat
init?(withRecord: CKRecord) {
  //
  guard
    let _name = withRecord["docName"] as? String,
    let _data = withRecord["docData"] as? NSData,
    let _dataDecoded = try?
JSONDecoder().decode(MyCKDocumentCont.self, from: _data as Data)
    else { return nil }

  //
  self.name = _name
  self.content = _dataDecoded
  self.ckRecordID = withRecord.recordID
}
```

Demo, save document

```
//  
func save() {  
    //  
    let _ckRec = ckRecordSave  
    //  
    _ckRec["docName"] = self.name as NSString  
    //  
    if let _dt = try? JSONEncoder().encode(content) as NSData {  
        //  
        _ckRec["docData"] = _dt  
    }  
    // !!! async  
    db.save(_ckRec) { (resCKR, err) in  
        //  
        if let _resCKR = resCKR, err == nil {  
            //  
            self.ckRecordID = _resCKR.recordID  
  
            //  
            print("Save uspech, \(self.ckRecordID)")  
        }  
    }  
}
```

Demo, vytvoření dokumentu

```
//  
func newDoc() -> MyCKDocument {  
    //  
    let doc = MyCKDocument(name: "dokument-1")  
  
    //  
    doc.content.text = "Nejaky text"  
    doc.content.notes = ["1", "2", "1234"]  
  
    //  
    doc.save()  
  
    //  
    return doc  
}
```

Demo, načtení obsahu

```
func loadMeta() {  
    //  
    let _db = CKContainer.default().privateCloudDatabase  
    let _q = CKQuery(recordType: "DocRecordType",  
                    predicate: NSPredicate(value: true))  
    //  
    _db.perform(_q, inZoneWith: nil) { (records, error) in  
        //  
        if let _recs = records {  
            //  
            for _r in _recs {  
                //  
                let _doc = MyCKDocument(withRecord: _r)  
  
                //  
                mydocs.append(_doc)  
            }  
        }  
        //  
        DispatchQueue.main.async {  
            tableView.reloadData()  
        }  
    }  
}
```

Demo, zhodnocení

- Při spuštění aplikace se načítá celý obsah tabulky DocRecordType.
 - přesun metadat včetně obsahu datové složky (velikost).
- Vylepšení:
 - CKAsset — datová složka jako separátní syncovaný dokument.
 - Lokální databáze / cache CKRecords. Rozdílový fetch.

CKAsset

- Datová (souborová) příloha CKRecord.
- Je to wrapper nad URL — založení CKAsset, čtení CKAsset.
- Lze předpokládat, že je to v CK referencovaný objekt (refCount). Automatické mazání z cache / DB.
- Při Query se nejspíš nestahuje obsah CKAsset.

```

// Generator CKAsset z MyCKDocumentCont
class MyCKDCAsset {
  //
  var assetURL: URL
  var ckasset: CKAsset
  // ... generovani
  init?(withCont: MyCKDocumentCont) {
    //
    guard let _dt = try? JSONEncoder().encode(withCont) else {
      //
      return nil
    }
    // dejme tomu nejake TMP jmeno
    let __tmpName = NSUUID().uuidString + ".json"
    let _tmps = FileManager.default.temporaryDirectory
    // toto bude url pro CKAsset
    assetURL = _tmps.appendingPathComponent(__tmpName)
    // zapsani do app:sandboxTMP
    do { try _dt.write(to: assetURL) } catch { return nil }
    // instanciaci
    self.ckasset = CKAsset(fileURL: assetURL)
  }
  // destruktor maze docasny soubor
  deinit {
    try? FileManager.default.removeItem(at: assetURL)
  }
}

```


save/load s CKAsset

```
func save() {
    //
    let _ckRec = ckRecordSave
    //
    _ckRec["docName"] = self.name as NSString
    // generator ckasset
    if let _assik = MyCKDCAsset(withCont: self.content) {
        // pokud uspeje, pak mam url/ckasset
        _ckRec["docAsset"] = _assik.ckasset
        // objekt musi prezit export, pak maze
        // docasny soubor
        ckAsset = _assik
    }
    //
    db.save(_ckRec) { (resCKR, err) in
        //
        if let _resCKR = resCKR, err == nil {
            //
            self.ckRecordID = _resCKR.recordID
            //
            print("Save uspech, \(self.ckRecordID)")
        }
    }
}
```

```

// instanciacie, z ulozenych CKRecord dat
  init?(withRecord: CKRecord) {
    //
    guard
      let _name = withRecord["docName"] as? String,
      let _asset = withRecord["docAsset"] as? CKAsset
    else { return nil }
    //
    self.name = _name
    self.ckRecordID = withRecord.recordID
    self.ckAsset = nil
    self.content = MyCKDocumentCont(text: "", notes: [])
    //
    if let _dt = try? Data(contentsOf: _asset.fileURL) {
      //
      if let _cont = try?
JSONDecoder().decode(MyCKDocumentCont.self,
                                                                from: _dt)
      {
        //
        self.content = _cont
      }
    }
  }
}

```

shrnutí / revize

- Přistupujeme do databáze (public / private), která představuje datový kontejner (entity, záznamy).
- CKRecord tedy existuje v konkrétní databázi.
- Přistupujeme pomocí NSOperation. Vše je asynchronní.
- Bude nás zajímat dynamika, události o změnách.
- Koncept dynamiky v aplikaci — tlačítko "refresh", zprávy od Cloudu o změnách.

Tlačítko "refresh"

- Cílem je synchronizovat lokální paměť (zatím ne perzistentní) s cloudem.
 - Záznamy: aktualizované, přidané, smazané.
 - 1) Kompletní fetch entity. Porovnání podle CKRecord.recordID.
 - 2) Z metadat vyhodnocovat novější záznamy, query, predicate — načti CKRecord modifikované po okamžiku T. Pozn: "modifiedAt" musí být "queryable".
 - 3) Rozdílový fetch — systém tokenů.

Rozdílový fetch

- Funguje pouze u private DB. V public DB by byl asi obtížně implementovatelný.
- Časová značka nad private DB (token).
- Fetch: dej mi události od mé časové značky.
- Lokální paměťový prvek: časová značka.
 - serverChangeToken.
 - budeme potřebovat RecordZone

Implementace diffFetch

- Potřebujeme zavést uživatelské zóny v PrivateDB.
- Ukládat changeToken (UserDefaults) — předpoklad pro diffFetch.
- Registrovat subscriptions + poslouchání Push Notifikací.

Správa uživ. zón v DB

```
// singleton spravy CK
class CKCfg {
    //
    static let shared = CKCfg()
    // kontejner a db
    lazy var container = CKContainer.default()
    lazy var db = container.privateCloudDatabase
    // aplikacni zona: CKRecordZone, CKRecordZoneID
    lazy var zoneID = CKRecordZone(zoneName: "pokusy").zoneID
    //
    func startMyZone() {
        //
        let _zone = CKRecordZone(zoneID: zoneID)
        let operation = CKModifyRecordZonesOperation(recordZonesToSave:
[_zone], recordZoneIDsToDelete: [])
        //
        operation.modifyRecordZonesCompletionBlock = { _, _, error in
            //
            print("Zalozeni zony \(error)")
        }
        //
        db.add(operation)
    }
}
```

Sledování zón

iCloud.eu.cdplan-solutions.CKDocumentsIZA > Development Data MARTIN HRUBÝ

ZONES RECORDS RECORD TYPES INDEXES SUBSCRIPTIONS SUBSCRIPTION TYPES SECURITY ROLES

LOAD ZONES FROM:

Private Database
mhafan@gmail.com

Fetch zone changes since...

List Zones

Create New Zone...

| Zone Name | Owner RecordName | Change Token | Atomic |
|----------------|-----------------------------------|-----------------------------------|--------|
| ▶ pokusy | _1d30972c9527763c9f9494b1885b9... | AQAAAAAAAAAAU////////+HawSpONI... | true |
| ▶ _defaultZone | _1d30972c9527763c9f9494b1885b9... | | false |

ZONES RECORDS RECORD TYPES INDEXES SUBSCRIPTIONS SUBSCRIPTION TYPES SECURITY ROLES

Private Database
mhafan@gmail.com

Fetch Subscriptions

| ID | type | filterBy | firesOn | firesOnce | shouldSend... | shouldBadge | alertBody |
|--------------|----------|----------|---------|-----------|---------------|-------------|-----------|
| ▼ my-updates | database | | | | true | false | ✕ |

Registrace subscription

- Aplikace provádí pouze jednou (na iCloudu).
 - Další požadavky z ostatních zařízení jsou ignorovány.
- Když jedno zařízení uloží změnu, iCloud automaticky testuje subscriptions a posílá ostatním zařízením PushNotif.
 - registrace pro poslouchání PushN.
 - obsluha PushN události — tichá PushN/s hlášením uživateli.
Na pozadí/Na popředí.
 - subscription nad DB, nad CK zónou

Smysl CKRecord zón

- Mají oddělovat tematické celky záznamů (ne nutně entit).
 - Např. provozně kritické záznamy, středně důležité...
- Subscriptions pak definovat nad zónami.

Registrace subscription

```
func startSubscriptions() {
    // pojmenovani subscription. Lze dodat NSPredicate
    let cks = CKDatabaseSubscription(subscriptionID: "my-updates")
    //
    let notifInfo = CKNotificationInfo()
    // varianta notifikace bez UI (ticha notifikace)
    notifInfo.shouldSendContentAvailable = true
    cks.notificationInfo = notifInfo
    //
    let operation =
CKModifySubscriptionsOperation(subscriptionsToSave: [cks],
subscriptionIDsToDelete: [])
    //
    operation.modifySubscriptionsCompletionBlock = { _, _, error in
        //
        print("Error \(error)")
    }
    //
    operation.qualityOfService = .utility
    db.add(operation)
}
```

```

//
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool
{
    //
    UIApplication.shared.registerForRemoteNotifications()
    //
    return true
}
// prijimani PushNotifikaci
func application(_ application: UIApplication,
didReceiveRemoteNotification userInfo: [AnyHashable : Any],
fetchCompletionHandler completionHandler: @escaping
(UIBackgroundFetchResult) -> Void)
{
    // konverze na CKNotifikaci
    let notif = CKNotification(fromRemoteNotificationDictionary:
userInfo)
    // ... zahajit rozdilovy fetch
    if notif.subscriptionID == "my-updates" {
        //
        CKCfg.shared.doDiffFetches()

        completionHandler(UIBackgroundFetchResult.newData)
    }
}
}

```

Provedení rozdílového fetch

- Vstupní changeToken — aplikace ukládá (UD).
- Výstupní changeToken — vrací iCloud.
- Může být rozloženo na několik skoků mezi tokeny.
- Výsledkem je seznam CKRecord pro lokální uložení a smazání.
 - Výstupní token — uložit.
 - Token nutno NS-kódovat. NSSecureCoding.

Správa changeTokens

```
extension UserDefaults {
    var serverChangeToken: CKServerChangeToken? {
        get {
            guard let data = self.value(forKey: "ChangeToken") as? Data else
            {
                return nil
            }

            guard let token = NSKeyedUnarchiver.unarchiveObject(with: data)
as? CKServerChangeToken else {
                return nil
            }

            return token
        }
        set {
            if let token = newValue {
                let data = NSKeyedArchiver.archivedData(withRootObject:
token)

                self.set(data, forKey: "ChangeToken")
            } else {
                self.removeObject(forKey: "ChangeToken")
            }
        }
    }
}
```

Sestavení diffFetchOp

```
//  
func doDiffFetches() {  
    // pole pro uložení získaných CKRecord  
    var recordsToSave = [CKRecord]()  
    var recordIDsToDelete = [CKRecordID]()  
    // options pro jednotlivé RecordZone  
    var optsDict = [CKRecordZoneID :  
CKFetchRecordZoneChangesOptions]()  
    // máme 1 zonu  
    let myZoneID = CKCfg.shared.zoneID  
    let options = CKFetchRecordZoneChangesOptions()  
    // nastavení původního tokenu  
    options.previousServerChangeToken =  
UserDefaults.standard.serverChangeToken  
    // ...  
    optsDict[myZoneID] = options  
    // alokace NSOperation  
    let operation = CKFetchRecordZoneChangesOperation(recordZoneIDs:  
[myZoneID], optionsByRecordZoneID: optsDict)
```

```

// hlaseni o CKRecord, ktere mam lokalne uložit
// (jsou pro me nove)
operation.recordChangedBlock = { record in
    //
    recordsToSave.append(record)
}
// zaznamy pro smazani z lokalni kopie
operation.recordWithIDWasDeletedBlock = { recordID, _ in
    //
    recordIDsToDelete.append(recordID)
}
// komplexni zprava o dokonceni fetche
operation.recordZoneFetchCompletionBlock = {
    // id zony (zname); moreComing
    zone, newToken, _, more, error in
    //
    if let _nt = newToken, error == nil {
        // ulozeni noveho tokenu
        UserDefaults.standard.serverChangeToken = _nt
        // ...
        self.processUpdates(toSave: recordsToSave, toDel:
recordIDsToDelete)
    }
}
//
CKCfg.shared.db.add(operation)

```


Životní cyklus s diffFetch

- Na startu aplikace — provést diffFetch.
 - moreComing.
- Na pushNotifikaci — diffFetch.
 - manuální refresh tlačítko.
- Kdykoli lze resetovat přechodem na token=nil.
 - doporučoval bych možnost resetovat diffFetching v aplikaci.
 - originál (platná / archivní podoba) dat je na Cloudu. Lokální cache jsou pouze pro efektivnější přístup.

Provázání na CoreData

- Koncept: aplikace je koncepčně CD-typu, tj. vytváří Managed objekty. Chceme změny odesílat na iCloud.
- ... a přijímat hlášení o změnách z iCloud a promítat ho do CD objektů.
- k diskuzi: *synchronizovat na MOC.save() nebo průběžně.*
- Potřebujeme sledovat změny na MOC.
 - specificky vznik nových objektů.
 - Zapisujeme do MO, MOC generuje události, FRC přeposílá UI.

Rozšíření DB schématu pro CK

ENTITIES

- CKDoc

FETCH REQUESTS

CONFIGURATIONS

- Default

▼ Attributes

| Attribute | Type |
|------------|-------------|
| ckencoded | Binary Data |
| ckrecordID | String |
| content | Binary Data |
| name | String |

+ -

ENTITIES

- CKDoc
- CKEntity

FETCH REQUESTS

CONFIGURATIONS

- Default

▼ Attributes

| Attribute | Type |
|------------|-------------|
| ckencoded | Binary Data |
| ckrecordID | String |

+ -

▼ Relationships

Vytvoření CD ManagedObj

```
extension CKDoc {
  // CKDoc: NSManagedObject
  // udalost generovana pri instanciaci
  override public func awakeFromInsert() {
    // transformace "self" na CKRecord
    let ck = CKRecord(recordType: "DocRecordType", zoneID:
CKCfg.shared.zoneID)
    //
    ck["docName"] = self.name as CKRecordValue?
    // operace do private db
    CKCfg.shared.db.save(ck) { ckrec, error in
      //
      if let _ck = ckrec, error == nil {
        //
        self.ckrecordID = _ck.recordID.recordName
      }
    }
  }
}
```

Managed to CKRecord

- V MO je nutno uchovat kódovanou podobu CKRecord. Při update záznamu se CKRecord rekonstruuje z kódu.
- atribut *ckencoded* v MO. Zřejmě nestačí pouze recordID (recordName, zoneID).
- záznam je identifikován recordName v zoneID. Při uložení pod stejným recordName může CK tvrdit "záznam už existuje". Další režijní atributy.

MO -> CKRecord, insert

```
//  
// extension CKDoc  
func ckInsert() {  
    //  
    let ck = CKRecord(recordType: "DocRecordType", zoneID:  
CKCfg.shared.zoneID)  
    //  
    ck["docName"] = self.name as CKRecordValue?  
    // operace do private db  
    CKCfg.shared.db.save(ck) { ckrec, error in  
        //  
        if let _ck = ckrec, error == nil {  
            let archivedData = NSMutableData()  
            let archiver = NSKeyedArchiver(forWritingWith: archivedData)  
            archiver.requiresSecureCoding = true  
            // !!!  
            _ck.encodeSystemFields(with: archiver)  
            archiver.finishEncoding()  
            // ulozeny atribut v CKDoc  
            self.ckencoded = archiver.encodedData  
        }  
    }  
}
```

MO -> CKRecord, update

```
//  
func ckUpdate() {  
    //  
    let unarchiver = NSKeyedUnarchiver(forReadingWith: self.ckencoded!)  
    unarchiver.requiresSecureCoding = true  
    // rekonstrukce CKRecord z kodu  
    let ck = CKRecord(coder: unarchiver)  
    // aktualizace hodnot  
    ck!["docName"] = self.name as CKRecordValue?  
    //  
    let op = CKModifyRecordsOperation(recordsToSave: [ck!],  
recordIDsToDelete: [])  
    //  
    op.modifyRecordsCompletionBlock = { _, error in  
        //  
        print("Update error \(error)")  
    }  
    //  
    CKCfg.shared.db.add(op)  
}
```

MO -> CKRecord

```
// zprava poslana tesne pred ulozenim obj
// do persistentstore, tj. "na disk"
// insert/update
override public func willSave() {
    //
    super.willSave()

    //
    if let _ = self.ckencoded {
        //
        ckUpdate()
    } else {
        //
        ckInsert()
    }
}
```


Synchronizace na MOC.save()

- Lze odchyťávat volání willSave na MO.
 - průběžně generovat zápisové operace do KC,
 - sebrat seznam modifikovaných objektů, po dokončení MOC.save() generovat souhrnnou operaci do KC.

Synchronizace na změny MOC

- Poslouchat notifikaci o změnách z MOC
 - `NSManagedObjectContextObjectsDidChange`
- Dekódovat `userInfo`, pak generovat souhrnnou operaci na KC

Centrální generování MO-CKR

```
@objc(CKEntity)
public class CKEntity: NSObject {
    //
    func encodeTo(ckRecord: CKRecord) -> Bool {
        //
        return true
    }
}

@objc(CKDoc)
public class CKDoc: CKEntity {
    //
    override func encodeTo(ckRecord: CKRecord) -> Bool {
        //
        if let _name = name {
            //
            ckRecord["docName"] = _name as CKRecordValue
        }

        //
        return true
    }
}
```

```

func generateCK() -> CKRecord? {
    //
    var ck:CKRecord
    //
    if let _ckenco = self.ckencoded {
        //
        let unarchiver = NSKeyedUnarchiver(forReadingWith: _ckenco as Data)
        unarchiver.requiresSecureCoding = true
        // rekonstrukce CKRecord z kodu
        ck = CKRecord(coder: unarchiver)! // !!
    } else {
        //
        ck = CKRecord(recordType: entity.name!, zoneID: CKCfg.shared.zoneID)

        //
        let archivedData = NSMutableData()
        let archiver = NSKeyedArchiver(forWritingWith: archivedData)
        archiver.requiresSecureCoding = true
        // !!!
        ck.encodeSystemFields(with: archiver)
        archiver.finishEncoding()
        //
        self.ckencoded = archivedData
    }
    // lze prepsat v navazujicich M0
    if encodeTo(ckRecord: ck) == false { return nil }
    //
    return ck
}

```

Událost na MOC, CK operace

```
//
@objc func mocEvent(_ notif:Notification) {
    if let _uinfo = notif.userInfo {
        //
        let _upa = _uinfo[NSUpdatedObjectsKey] as! [CKEntity]
        let _ui = _uinfo[NSInsertedObjectsKey] as! [CKEntity]
        let _ud = _uinfo[NSDeletedObjectsKey] as! [CKEntity]

        // transformace do CKRecords...
//////////
        let op = CKModifyRecordsOperation(recordsToSave: forSaving,
                                          recordIDsToDelete:
forDeleting)
    }
}
```

Poznámka k public CK-DB

- Obsah vidí všichni uživatelé aplikace.
 - Mohou vkládat. Synchronizace (broadcast) prostřednictvím subscriptions a push-notifikací.
- Smysl:
 - zřejmě nelze uvažovat plnou synchronizaci lokální a serverové DB.
 - zřejmě jenom prostředek pro broadcast zpráv (CKRecord).
 - kdo však má záznamy mazat???

Závěr

- CK je jednoznačně univerzální robustní řešení pro synchronizaci dat mezi platformami.
 - nahrazuje UIDocuments.
- CoreData + CloudKit.
- Nevýhody — latence, asynchronnost.
- Zanedbáno: CKReference, polohové informace, uživatelská metadata / oprávnění, shared DB