Couchbase NoSQL Developer Workshop

ラボハンドブック

演習 5: K/V サブドキュメント操作

手順

このラボの目的は、新しい顧客住所の保存と既存の顧客住所の更新を可能にすることです。 この操作では、SDK のサブドキュメント操作を利用します。

ステップ 1: API にロジックを追加する

ドキュメント

- サブドキュメント操作

住所の追加

API リポジトリ ディレクトリで *repository.js* ファイルを開きます(付録の API プロジェクト構造を参照)。*saveAddress()* メソッドを検索します。 顧客ドキュメント内に住所を挿入 (または UPSERT)するために必要なロジックを追加するため、*saveAddress*() メソッドを編集します。

考える点がいくつかあります。

1. 顧客ドキュメント モデル
2. 指定されたパスが存在しない場合はどうなるか?
3. 親ドキュメントの監査プロパティの更新

*saveAddress( )*入力:

- custId: 整数 - 顧客の ID
- path: 文字列 - サブドキュメントパス
- address: オブジェクト - 顧客ドキュメントのアドレスオブジェクト
- callback

saveAddress() 出力:

- エラー オブジェクト (該当する場合)
- 顧客ドキュメント キー

*saveAddress()*メソッドの実装については、以下のコード スニペットを参照してください。
注意: NOP コード行をコメントアウトするか、新しいコードに置き換えます。

```
0:   async saveAddress(custId, path, address) {
1:     try {
2:       /**
3:        * Lab 5:  K/V sub-document operation(s):
4:        *  1.  generate key:  customer_<custId>
5:        *  2.  get customer addresses
6:        *  3.  create business logic to add new address
7:        *  4.  update customer address path
8:        *  5.  update customer modified date and modifiedBy
9:        *
10:       *
11:       *  When updating, think about pros/cons to UPSERT v. REPLACE
12:       */
13:      let key = `customer_${custId}`;
```

```
14:       let results = await this.collection.lookupIn(key, [
15:         couchbase.LookupInSpec.exists(path),
16:         couchbase.LookupInSpec.get(path),
17:       ]);
18:
19:       if (!results.content[0] || !results.content[1]) {
20:         return { success: false, error: null };
21:       }
22:
23:       let addresses = results.content[1].value;
24:       let { name, ...newAddress } = address;
25:       addresses[name] = newAddress;
26:       let modifiedDate = Math.floor(new Date() / 1000);
27:
28:       this.collection.mutateIn(key, [
29:         couchbase.MutateInSpec.upsert(path, addresses),
30:         couchbase.MutateInSpec.upsert("doc.modified", modifiedDate),
31:         couchbase.MutateInSpec.upsert("doc.modifiedby", custId),
32:       ]);
33:
34:       return { success: true, error: null };
35:     } catch (err) {
36:       //Optional - add business logic to handle error types
37:       outputMessage(err, "repository.js:saveAddress() - error:");
38:       return { success: null, error: err };
39:     }
41:   }
```

コードに関する注意事項:

- async/await (非同期/待機) 構文を使用します。
- 13 行目:ドキュメントのキーを作成しますが、キーが渡されるのではなく custId が渡されます。
- 14 行目: K/V 操作は、3.x SDK のコレクション レベルで行われます。
- 14-17 行: サブドキュメントの検索操作を使用して、パスが存在するかどうかを確認し、現在の顧客の住所を取得します。
- 23-26 行: address のパスが存在するかどうかを確認する場合は、customer ドキュメント モデルの構造に従い、address サブドキュメント オブジェクトに新しいキーを追加する必要があります。customer ドキュメントのサンプルについては、付録を参照してください。
- 26 行目:親ドキュメント監査プロパティを更新するために、現在の日付を設定します。
- 行 28-32:サブドキュメント操作を使用して、新しい address と監査用プロパティ(`modified`, `modifiedby`)を作成または更新(アップサート)します。
- *outputMessage()*: コンソールに情報を簡単に出力するためのヘルパーメソッドは、/library ディレクトリにあります (付録の API プロジェクト構造を参照してください)。
- try/catch & err オブジェクトの処理は、意図的に汎用的な方法で行われます。 ラボ参加者は、エラーを処理するさまざまな方法をテストするために、ロジックを自由に追加できます。

完了したら、*repository.js* ファイルが保存されていることを確認します。 API *Docker* コンテナは API の作業ディレクトリにマップされるため、API コードに対して行われたすべての更新はコンテナに反映される必要があります。 コードを保存すると、新しいアドレスを追加する機能が Web UI 内で利用可能になります。 以下の手順に従って、*saveAddress()*の動作を確認します。

注 ***

saveAddress () を使用するにはログイン認証が必要です。

1. http://localhost:8080 に移動
2. ログインしていない場合、ログインします。
3. 右上のカート アイコンをクリックします。
    a. または、ユーザー アイコンの横にある [Hello {名}] をクリックし、ドロップダウン メニューの [*Cart*] をクリックし、Web UI がカートページにリダイレクトする必要があります。
4. カートページで、*Shipping(配送)*または *Billing(請求)*セクションの「+」をクリックします。
5. フォームのフィールドに入力します (一部は必須フィールドです)
    a. フォーマットの検証は行われません。
6. [SAVE] をクリック します。
7. アドレスが正しく保存されると、" *Successfully saved new address*" というメッセージが表示されます。

住所の更新

API リポジトリ ディレクトリで *repository.js* ファイルを開きます (付録の API プロジェクト構造を参照)。*updateAddress( )*メソッドを検索します。 顧客ドキュメントを更新するために必要なロジックを追加するため、*updateAddress*() メソッドを編集します。

考える点がいくつかあります。

1. 顧客住所のパス全体または一部を更新する
2. アップサート 対置換
3. 指定されたパスが存在しない場合はどうしますか?

updateAddress() 入力:

- custId: 整数 - 顧客の ID
- パス: 文字列 - サブドキュメントパス
- 住所: オブジェクト - 顧客ドキュメントアドレス オブジェクト
- コールバック

updateAddress() 出力:

- エラー オブジェクト (該当する場合)
- 顧客ドキュメント キー
-

*updateAddress()*メソッドの実装については、以下のコード スニペットを参照してください。
注意: NOP コード行 をコメントアウトするか、新しいコードに置き換えます。

```
0:  updateAddress(custId, path, address, callback) {
1:    try {
2:      /**
3:       * Lab 5:  K/V sub-document operation(s):
4:       *  1.  generate key:  customer_<custId>
5:       *  2.  update customer document address path
6:       *  3.  update customer document modified date and modifiedBy
7:       *
```

```
8:          *  When updating, think about pros/cons to UPSERT v. REPLACE
9:          */
10:         let key = `customer_${custId}`;
11:         let modifiedDate = Math.floor(new Date() / 1000);
12:
13:         this.collection.mutateIn(key, [
14:           couchbase.MutateInSpec.upsert(path, address),
15:           couchbase.MutateInSpec.upsert("doc.modified", modifiedDate),
16:           couchbase.MutateInSpec.upsert("doc.modifiedby", custId),
17:         ]);
18:
19:         return { success: true, error: null };
20:       } catch (err) {
21:         //Optional - add business logic to handle error types
22:         outputMessage(err, "repository.js:updateAddress() - error:");
23:         callback(err, null);
24:       }
25:     }
```

コードに関する注意事項:

- async/await(非同期/待機)構文を使用します。
- 10 行目:渡された custId キーを用いてドキュメントのキーを作成します。
- 13 行目: K/V 操作は、3.x SDK の Collection レベルで実行されます。
- 13-17 行: サブドキュメント変更操作を使用して address(パス)全体を更新します。また併せて、親ドキュメントの監査用プロパティ(modified, modifiedby)を更新します。
- *outputMessage()*: コンソールに情報を簡単に出力するためのヘルパーメソッドは、/library ディレクトリにあります (付録の API のプロジェクト構造を参照ください)。
- try/catch & err オブジェクトの処理は、意図的に汎用的な方法で行われます。 ラボ参加者は、エラーを処理するさまざまな方法をテストするために、それに応じてロジックを自由に追加できます。

完了したら、*repository.js* ファイルが保存されていることを確認します。 API *Docker* コンテナは API の作業ディレクトリにマップされており、API コードに対して行われたすべての更新はコンテナに反映されるはずです。 コードを保存すると、住所検索機能が Web UI 内でアクティブになるはずです。
以下の手順に従って、*updateAddress*() ロジックを確認します。

注 ***

updateAddress () を使用するにはログイン認証が必要です。

1. http://localhost:8080 に移動
2. ログインしていない場合、ログインします。
3. ユーザー アイコンの横にある [Hello {名}] をクリックし、ドロップダウンメニューの [*UserProfile*]をクリックし、Web UI が*[UserProfile]* ページにリダイレクトされます。
4. [*UserProfile*]ページで、いずれかのアドレス (たとえば、HOME) の [EDIT] をクリックします。
5. 都市と州を更新します。
6. 「SAVE」をクリックします。
7. 画面の上部に" *Successfully saved new address*"というメッセージが表示されます。

クリーンアップ

コンテナーを停止する

黄色で強調表示されているコンテナ名は、停止するコンテナの名前に変更する必要があります。 この例では、web コンテナが停止されます。 コンテナの作成時に --rm オプションが使用された場合、コンテナを停止するとコンテナ自体が削除されます。

```
$ docker stop web
web
```