Web サービスを作る本の Firestore 移行マニュアル

渡邊達明　著

2018-09-10 版　　**THE SHIMESABUZZ** 発行

目次

第 1 章

はじめに

「Vue.js と Firebase で作るミニ Web サービス ~初めてのサーバーレスシングルーページアプリケーション~」（以下、Web サービス作る本）では Firebase を利用してユーザー登録やサイトホスティング、またユーザーデータの保存を利用していました。

本の内容でデータ保存には「Realtime Database（以下 RTDB）」を利用していましたが、後継として「Firestore」という別なデータベースがあります。執筆時にはまだ β 版で不安定だったため不採用にした Firestore ですが、最近は動作も安定しており本番環境で採用している企業も増えてきました。個人的には今年の末～来年の頭までには β 版が解除されるのではないかと思っています。（本マニュアル執筆:2018 年 9 月）

Firestore は RealtimeDB に比べてさまざまな機能が追加されており、Web サービス作る本で利用するような通常のデータベースであれば、今後は基本的に Firestore を使ったほうがよいと考えています。

本マニュアルでは RTDB との違いや、Web サービス作る本において Firestore を使った場合の実装への移行について書いていきますと思います。

第 2 章

RTDB と Firestore の違いについて

現在 Firestore では、東京リージョンが選択出来ない（予定はされています）ため、動作で少しタイムラグが発生します。そのため、RTDB はリアルタイムにデータのやりとりを行う場面では Firestore よりも優れているため、場面によって使い分けていただければと思います。

2.1 データの保存形式

RTDB では全体のデータを 1 つの大きな JSON ツリーとして保存します。シンプルなデータは非常に簡単に保存できますが、複雑で階層的なデータを処理する場合は色々と工夫が必要でした。

Firestore でも保存するデータは概ね、JSON に似ています。Firestore はコレクションというデータのまとまりにまとめ、内部データをドキュメントと呼び、コレクションごとに個別でクエリを発行して検索などができるなどが RTDB と大きく違う点です。

データを階層化しても、その階層化のデータをまたサブコレクションとすることで大量のデータも処理しやすくなりました。それにより RTDB のデータを利用する際に必要だった平滑化などのテクニックがなくともさまざまな処理をやりやすくなりました。

さらにデータに型を定義できたり、リファレンス型という他のコレクションやドキュメントのデータリファレンスを格納することができるようになり、リレーショナルなデータ格納が可能になりました。

2.2　ルール設定の違い

基本的に read と write のみで、varidation でそれを補う形でしたが、update・delete・create なども詳細に設定をすることができるようになったり、自分自身以外のデータを取得して比較するなども比較的楽になりました。

また、ルール内でカスタム関数を定義できるようになり、複雑化するルールのチェックをより簡潔に記述できるようになりました。

関数は `function` キーワードで定義され、0 個以上の引数を使用します。たとえば、上に示した例で使用した 2 つのタイプの条件を 1 つの関数にまとめることができます。

```
service cloud.firestore {
  match /databases/{database}/documents {
    // True if the user is signed in or the requested data is 'public'
    function signedInOrPublic() {
      return request.auth.uid != null || resource.data.visibility == 'public';
    }

    match /cities/{city} {
      allow read, write: if signedInOrPublic();
    }

    match /users/{user} {
      allow read, write: if signedInOrPublic();
    }
  }
}
```

▲図 2.1　カスタム関数の例

2.3　料金の違い

RTDB は帯域幅とストレージにのみ課金でしたが、Firestore はデータベースで実行されているオペレーション（読み取り、書き込み、削除）に課金され、帯域幅とストレージの課金レートは低くなります。また、Google App Engine と連携させて、1 日の限度額を設定できるようにもなりました。

2.4　その他の違い

RTDB では膨大なデータが保存されている場合にはスケールする際にシャーディングが必要となっていましたが、Firesoter ではスケーリングの際にそういった壁はありません。

第 3 章

RTDB からの移行作業

初版 P43 「5.4 Firebase Realtime DB の rule を設定する」を Firestore で設定する場面から順に進めていきます。

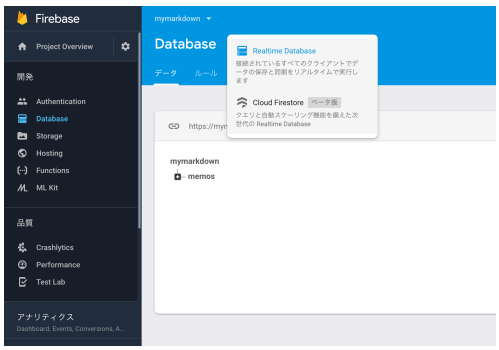
3.1　初期設定

まずは Firebase の管理画面から今回のプロジェクトで利用しているプロジェクトを開きます。

左のメニューから Databese を選択すると RTDB のデータ画面が表示されると思います。

※もし新規プロジェクトで作成する場合は Firestore が表示されていると思いますので、データベースの作成を選択してください。

データベースの管理画面の上部にある「Realtime Database」と表示してあるセレクトボックスをクリックすると「Cliud Firestore」がありますので、それを選択します。



▲図 3.1　セレクトボックスをクリックした状態

その後セキュリティルールの初期設定をどうするか表示されますが、こちらは後ほど変更するためロックモードのまま次へ進みます。

▲図 3.2　セキュリティルールの選択

これでデータベースとして利用する Firestore のセットアップは完了です。次に RTDB と同じようにルールの設定を行います。

3.2　ルール設定

Firestore では RTDB でのルール設定と比べて複雑なルールも比較的簡単に記述しやすくなりました。また、RTDB のみであったルール設定の確認のためのシミュレータも 2018 年 6 月から実装されたことでルールの検証がしやすくなりました。

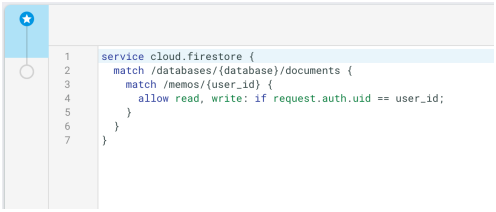
公式ドキュメントは次になります。

> Cloud Firestore セキュリティルールを使ってみる
> `https://firebase.google.com/docs/firestore/security/get-started`

RTDB で設定した場合と同じように memos というコレクションを作成し、その中にユーザー毎のドキュメントを作成するという形にしようと思います。

管理画面のルールタブを選択し、ルール設定画面を表示します。RTDB と同じように memos 配下に認証時のユーザー ID(（user_id）と同じドキュメント名配下は自分自身しか読み書き出来ないように設定します。

RTDB では"auth"に認証情報が格納されていましたが、Firestore のルール設定では"request.auth"になっているので注意が必要です。

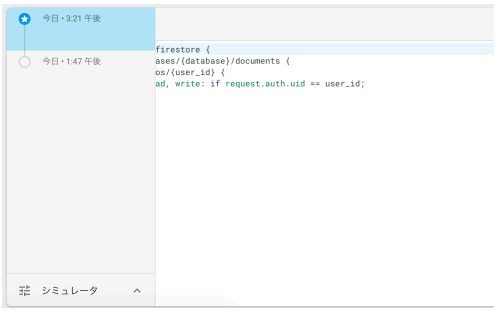


▲図 3.3　セキュリティルールの設定

3.3　ルールシュミレータの実行

Firestore でもルールシュミレータが実装されたので、試しに利用してみましょう。

ルール管理画面の左の星の辺りにマウスオーバーすると、ルール設定の履歴が閲覧できます。下にあるシミュレータボタンをタップしてみましょう。



▲図 3.4　シミュレータの実行

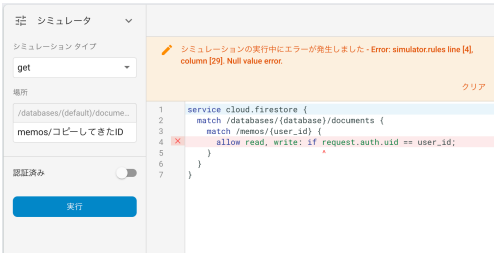
シミュレーションタイプでは get や create 等それぞれテストができます。今回は memos 配下の user_id のドキュメントが読み込み出来ないかのシミュレーションを試します。

ここで自身の認証ユーザー ID が知りたいので、一度 Authentication タブを開きます。そこでは認証済のユーザーの一覧が見られますので、自分のメールアドレスの「ユーザー UID」をコピーします。

▲図 3.5　認証情報一覧

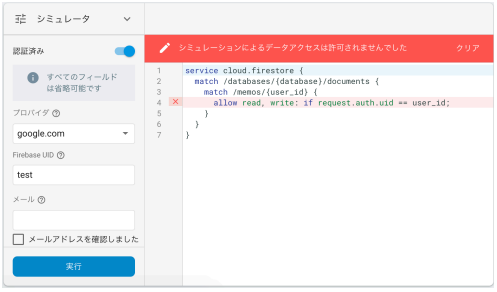
もう一度ルールシミュレータに戻り「場所」の部分に「memos/コピーしてきたユーザー ID」を入力します。認証済みボタンを OFF のまま触らず「実行」ボタンを押してみてください。

「 Error: simulator.rules line [4], column [29]. Null value error.」という表示がされたと思います。これは認証がされていないユーザーが DB の読み込みを行おうとして、認証情報が入っている"request.auth"が空のためエラーになって読み込みに失敗（データが正しく保護されている）しています。



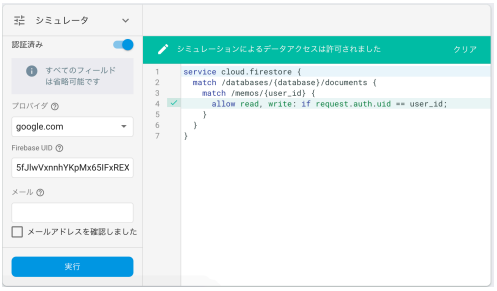
▲図 3.6　認証していないユーザーのアクセス

次は「認証済み」のボタンを ON にして、Firebase UID の部分にコピーしてきた ID と別な ID を入力して、実行してみましょう。今度は「シミュレーションによるデータアクセスは許可されませんでした」と、出ました。認証情報が一致しないことで読み込みに失敗しています。



▲図 3.7　自分以外のユーザーのアクセス

最後に認証済みのユーザーを DB の ID と一致させて実行してみて下さい。「シミュレーションによるデータアクセスは許可されました」と表示されたらチェック OK です。データが正しく保護されていることを確認できました。



▲図 3.8　アクセス成功

ではルールの設定ができたので、データの読み取りと保存の実装に移りましょう。

3.4　データ保存の実装

初版 P45「5.5 メモの保存と読み込み機能の作成」を Firestore に移行します。

まず Firestore を利用するためには、Firestore 用の SDK を別途読み込む必要があります。

> Cloud Firestore を使ってみる
> `https://firebase.google.com/docs/firestore/quickstart`

こちらの Firestore を使ってみるの「開発環境の設定」で「ウェブ」のタブにある Script

をコピーして、index.html の script を置き換えましょう。

▼リスト 1.1　/dist/index.html の firebase.js の読み込み部分を置き換え

```
<script src="https://www.gstatic.com/firebasejs/4.12.1/firebase.js"></script>
<script src="https://www.gstatic.com/firebasejs/4.12.1/firebase-firestore.js">
</script>
```

次に Editor.vue の saveMemos 関数の中身を Firestore 対応したものに置き換えます。

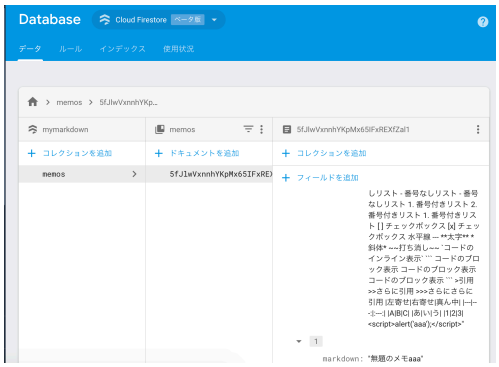
▼リスト 1.2　/src/components/Editor.vue

```
saveMemos: function() {
  firebase
    .firestore()
    .collection("memos")
    .doc(this.user.uid)
    .set({ memos: this.memos });
},
```

RTDB の場合は ref でデータベースのリファレンスを直接指定していましたが、Firestore ではコレクション. ドキュメントという形式での指定が必要です。

また、このとき.set の中身はドキュメントへの保存となり、オブジェクトである必要があるため this.memos をそのまま保存しようとするとエラーになってしまいます。そのため、memos というキーのオブジェクトに保存するように設定しました。

管理画面でデータが Firestore へ保存されていることが確認出来たら次は読み取りへ移ります。



▲図 3.9　管理画面での Firestore のデータが保存されたことの確認

3.5 データ読み込みの実装

次は読み込み部分です、保存の際と同じようにリファレンスを作成してかた get する形になります。また、保存の際に memos というキーでオブジェクトへ変換したので、読み取り時にもオブジェクトのデータを読みに行く必要があります。

▼リスト 1.3　/src/components/Editor.vue

```
created: function() {
  firebase
    .firestore()
    .collection("memos")
    .doc(this.user.uid)
    .get()
    .then(doc => {
      if (doc.exists && doc.data().memos) {
        this.memos = doc.data().memos;
      }
    });
},
```

RTDB と同じようにリロードしてもデータが自動的に読み取り、表示されましたでしょうか。以上で Firestore への移行作業は終了となります。お疲れ様でした。

3.6 おわりに

いかがでしたでしょうか。RTDB と基本的な使い方は大きく変わらないため、移行作業自体は結構すぐに終わったと思います。

今後も Firebase では Firestore 以外にも便利な機能が増えていくと思われます。様々な機能を使いこなし得手不得手を理解して取捨選択しながら便利に利用して、面白い Web サービスを開発していっていただければと思います。

本マニュアルがそのための一助になれば幸いです。最後までお読みいただきありがとうございました。

なにか誤字・誤植等見つけられた方は Twitter で@nabettu 宛にご連絡いただければと思います。

著者紹介

渡邊 達明 @nabettu

株式会社クリモ取締役副社長。

1988 年宮城県生まれ。仙台高専専攻科を卒業後、富士通株式会社にて WindowsOS のカスタマイズ業務に従事する。その後面白法人カヤックにて受託開発部門を経験後、ブロガーの妻と二人で株式会社クリモを設立。WEB フロントエンド中心の受託開発や保育園問題の解決のためのメディアを運営。

「三度の飯よりものづくり」と言っていたら BMI が 17 になり健康診断で毎回ひっかかるのが悩み。

Twitter

https://twitter.com/nabettu

運営 Web サービス

ためしがき

日本語のフリーフォントをまとめて試せるサービス

https://tameshigaki.jp

アプリーチ

アプリを紹介するブログパーツがすぐ作れるサービス

http://mama-hack.com/app-reach/

技術ブログ

https://nabettu.hatenablog.com

Web サービスを作る本の Firestore 移行マニュアル

2018 年 9 月 10 日　1 版 v1.0.0
著　者　渡邊達明
発行所　THE SHIMESABUZZ

(C) 2018 TATSUAKI WATANABE