\Orchestrating a brighter world NEC

NECモバイルバックエンド基盤 ver 7
ご利用にあたっての注意事項

ver 7.0.0

2018/3/15

NEC IoT基盤開発本部

Orchestrating a brighter world

未来に向かい、人が生きる、豊かに生きるために欠かせないもの。
それは「安全」「安心」「効率」「公平」という価値が実現された社会です。

NECは、ネットワーク技術とコンピューティング技術をあわせ持つ
類のないインテグレーターとしてリーダーシップを発揮し、
卓越した技術とさまざまな知見やアイデアを融合することで、
世界の国々や地域の人々と協奏しながら、
明るく希望に満ちた暮らしと社会を実現し、未来につなげていきます。

改版履歴

バージョン	日付	変更内容
ver 3.0.0	2015/12/2	初版
ver 3.0.1	2016/1/22	クライアントSDK品質保証範囲を追記 (p5) サーバ品質保証範囲を追記 (p11) インデックス設定時の注意事項を追記 (p14)
ver 4.0.0	2016/4/5	モバイルバックエンド基盤 v4 向け修正 品質保証範囲を修正
ver 4.0.1	2016/4/19	モバイルバックエンド基盤 v4.0.1 向け修正 • 論理削除オブジェクトの定期自動削除追記 • .NET SDK リクエストタイムアウト・キャンセル記述追記 • APNs Production 証明書設定不具合の記載解除
ver 4.1.0	2016/5/13	モバイルバックエンド基盤 v4.1.0 向け修正 • JavaScript SDK API 変更について追記 (p5)
ver 5.0.0	2016/11/11	モバイルバックエンド基盤 v5.0.0 向け修正 • オブジェクトクエリ注意事項を追記(p7) • Cloud Functions 関連注意事項を追記 • リクエストトークンに関する注意事項を追記(p12) • Push通知に関する注意事項を追記(p20) • 文字種・文字列長の制限事項を追記(p35-)
ver 6.0.0	2017/3/16	• 品質保証範囲を変更 (p4) • 文字種・文字列長の制限事項をデベロッパーマニュアルに移動(p35-)
ver 6.5.0	2017/9/22	• 品質保証範囲を変更 (p4) • Java9対応について追記 (p22)
ver 7.0.0	2018/3/5	• 品質保証範囲を変更 (p4) • Java9以降の対応方針について追記 (p22)

品質保証範囲

以下は品質保証範囲外(β扱い)となります。

- JavaScript SDK の一部
  - SDE4SD 利用時に使用できる、ファイルシステム上ファイルの直接アップロード・ダウンロード機能 (FileBucket.uploadNewFile(), FileBucket.downloadFile())
- .NET SDK の一部
  - バケット管理系 API
- iOS SDK：全体

クライアント SDK 関連

JavaScript SDK v4.1.0 API変更

- JavaScript SDK v4.1.0 において、v4.0.x 以前のバージョンと互換性のない API 変更が行われています。
  - アプリケーションの実装に変更が必要になります。

- 影響のある変更の概要は以下のとおりです。
  - jQuery.Deferred サポートを廃止し、ES6 Promise に移行
  - コールバックの引数個数(主にエラーコールバック)を変更 (1個に統一)
  - User.current() API を同期APIに変更
  - ObjectBucket.query() API を分割

- 詳細は「JavaScript SDK ガイド」内の「マイグレーションガイド」を参照ください。

オブジェクトストレージ：オブジェクトクエリ (1)

ソート条件

- オブジェクトクエリにソート条件を指定する場合、原則ソートに使用するキーに対してインデックスを指定してください。
- インデックスを設定しない場合、ソート対象のデータ量は32MB以内に収める必要があります。この制限を超えると、クエリはエラーとなります。

クエリ条件式

- ブラウザから JavaScript SDK を使用してクエリ式を発行する際、クエリ式が非常に大きいと(概ね2KB程度以上)、URLが長くなりすぎブラウザによってはリクエスト送信が失敗する場合があります。
  - この場合は、v5 より新規追加されたロングクエリ API を使用してください。ロングクエリ API では、HTTP GET の代わりに POST が使用されるため、クエリ式の長さの制限はありません。
- クエリ条件式は URL に含まれサーバのアクセスログに保存されるため、機密情報を条件式に含めないようにしてください。
  - どうしても機密情報を含めざるを得ない場合は、ロングクエリAPIを使用してください。

オブジェクトストレージ：オブジェクトクエリ (2)

大量データのクエリ

- 大量のデータに対してクエリを行う場合は、適切にインデックスを設定してください。
- インデックスを設定せずに大量データにクエリをかけると、MongoDB サーバに多大な負荷がかかり、応答が返ってこない場合があります。
- このようなクエリをかけてしまった場合の対処として、MongoDB のデフォルトクエリタイムアウト時間を設定することを推奨します。
  - タイムアウト時間は、デベロッパーコンソールの「システム設定」より設定できます。
  - クエリ処理がこの時間を超えた場合、クエリは中断されクライアントには 408 Request Timeout エラーが返却されます。

オブジェクトストレージ: オブジェクト削除

- オブジェクトストレージのオブジェクト削除動作は、SDK によって動作が異なります。
  - Java / Android SDK: 論理削除のみ
  - JavaScript SDK: オフライン機能非使用時は物理削除、オフライン機能使用時は論理削除
  - iOS SDK: 物理削除のみ
  - .NET SDK: 物理削除・論理削除ともサポート (API引数で指定)

- 物理削除した場合は MongoDB 上から即時にオブジェクトが削除されますが、論理削除したオブジェクトは MongoDB 上にはデータとして残ったままの状態となります。
  - 論理削除状態のオブジェクトデータは、オフライン機能使用時に削除状態をクライアントに通知するために必要となります。
  - 論理削除状態のオブジェクトが残っていても動作上問題はありませんが、ストレージ容量を専有します。物理的に消去するには、定期削除機能を使用するか、明示的に物理削除 API を呼び出して削除する必要があります。詳細は次ページを参照ください。

オブジェクトストレージ: 論理削除オブジェクト (自動削除)

論理削除オブジェクトの自動定期削除

- モバイルバックエンド基盤サーバ ver 4.0.1 より、論理削除オブジェクトの自動定期削除機能が追加されています。
- 1時間に1回の頻度で、論理削除オブジェクトの削除処理が実行されます。
- デベロッパーコンソールより、テナント単位で論理削除オブジェクトの有効期限を設定できます。なお、デフォルトでは削除は行わないよう(有効期限が無限大)になっていますので、定期削除を行いたい場合は設定を変更してください。

オブジェクトストレージ: 論理削除オブジェクト (手動削除)

論理削除オブジェクトの手動一括削除手順

- 論理削除オブジェクトを手動削除する手順を示します。
- .NET SDK では一括削除 API が用意されていますので、こちらを使用してください。他の SDK は現時点では提供していませんので、直接 REST API を発行してください。
- 削除条件としては以下の指定を行ってください。
  - 論理削除されていること (“_deleted” フィールドが true である)
- 以下の指定を行うことで、一定期間経過したオブジェクトを指定して削除できます。
  - “updatedAt” が特定の日時以前である。

- 以下、例を示します。この例では 2016/3/1 00:00:00 (UTC) 以前の論理削除オブジェクトを物理削除します。

```
//.NET SDK の場合の例
var query = new NbQuery().DeleteMark(true).LessThan("updatedAt", "2016-03-01T00:00:00.000Z");
await bucket.DeleteAsync(query, false);
```

```
// REST API 呼び出し(curl)の場合の例
// {} 内の文字列は適宜適切な値を代入してください
curl –X DELETE ¥
  -H "X-Application-Id: {appId}" ¥
  -H "X-Application-Key: {masterKey}" ¥
  --data-urlencode 'where={"_deleted": true, "updatedAt": {"$lt": "2016-03-01T00:00:00.000Z"}} ' ¥
  https://{server}/api/1/{tenant_id}/objects/{bucketName}
```

オブジェクトストレージ: リクエストトークン

リクエストトークンの仕様

- 直接 REST API を用いて、オブジェクトの作成・バッチリクエストを行う場合、リクエストトークンを指定することができます。
- クライアント側から requestToken を指定した場合、サーバはレスポンスをキャッシュします。同一 requestToken を持つリクエストが再送された場合、サーバは処理を行わずキャッシュしたレスポンスのみを返却します。

リクエストトークンを利用する場合の注意点

- 同一requestTokenでのリクエストが非常に短い間隔で再送された場合には、処理が重複して行われる可能性があります。
- これは、ネットワークエラーなどによりレスポンス取得に失敗した場合の再送処理対応が本来の目的であり、短時間での連続送信は目的外であるためです。
- SDKを利用する場合は、SDK内で考慮されるため問題ありません。

ファイルストレージ

ファイルアップロード

- Java / Android SDK で、InputStream からファイルアップロード(新規・更新)する場合、アップロード中にクライアントの電源断が発生するとアップロード途中のファイルがサーバに残る場合があります。
  - InputStream ではなくファイルパスを指定した場合は問題ありません。
  - JavaScript/SDE 環境でファイルパス指定する場合も問題ありません。
- Pure JavaScript SDK で、ブラウザからファイルを指定してアップロードを行う場合、クライアントの電源断が発生するとアップロード途中のファイルがサーバに残る場合があります。
  - 具体的には、HTTP の Chunked Encoding でファイル転送が行われる場合に発生します。

- 対策
  - Java/Android SDK でファイルアップロードを行う場合は、ファイルパスを指定するようにしてください。
  - ファイルパスを指定できない場合（ブラウザからアップロードする場合)は、回避策はありません。電源断が発生した場合は、手動でファイルを削除するようにしてください。

Android SDK：Android 5.1.1 使用時の注意点

Android 5.1.1 の特定環境で、メモリリークが発生する可能性があります

- 具体的には、SDE for SmartDevice に組み込み、WebView を使用する場合。
- Android 5 の ART に関連して、ネイティブヒープがリークすることを確認しています。
  - モバイルバックエンド基盤の API を連続発行し続ける状態で、1時間に100KB程度。
- Android の ART / WebView の潜在バグと思われます。
- Android 4.x および 6.0 環境では問題はありません。

- 対策
  - 通常利用で問題になることはないと思われますが、Android 5.1.1 + WebView 環境での利用時には注意が必要です。
  - 問題になるようであれば、1日に1回程度アプリを再起動するなどの対策を行って下さい。

JavaScript SDK：Android 5.x 以降使用時の注意点

Android 5.x 以降で jQuery Mobile を使用する際、画面が真っ白で何も表示されない現象が発生する場合があります。

- 具体的には、WebView を Google Play から最新版に更新した場合。
- 本件は、jQuery Mobile 側の問題です。

- 対策
  - 本現象が発生した場合は、jQuery Mobile の読み込み前に以下のコードを挿入することで回避可能です。

```
<script>
   $(document).on("mobileinit", function () {
      $.mobile.pushStateEnabled = false;
   });
</script>
<!-- jQuery Mobile 読み込み -->
<script type="text/javascript" src="jquery.mobile.js"></script>
```

.NET SDK

REST API 実行にネットワーク切断が発生した場合、結果通知(正常・異常ともに)がされない場合があります。

- 特にネットワーク回線が細い場合に発生しやすくなります。
- ネットワーク接続が回復したタイミングで結果通知がされます。

- オフライン同期を行っている最中に本現象が発生した場合、以下の問題があります。
  - 内部的に結果通知が OS から返ってくるまで、内部的に同期処理が停止します。
  - 同期処理中は、オフラインバケットに対する読み書きはすべてエラーとなります。

- 対策
  - .NET SDK v4.0.1 より、HTTPリクエストのタイムアウト設定、およびリクエストキャンセルAPIを追加しています。
  - 一定時間内に必ず何らかの応答が必要な場合は、タイムアウト値を設定してください。
  - オフライン同期中に本現象が発生した場合は、リクエストキャンセルAPIを使用して処理を中断してください。

サーバ：運用時の注意事項

サーバ: ホットデプロイ時の注意

- Tomcat 上に WAR ファイルをホットデプロイした際は、必ず Tomcat を再起動するようにしてください。
  - 再起動しなくても動作上は大きな問題はありませんが、ホットデプロイを繰り返すと一部のクラスのメモリが解放されずメモリ消費量が上昇したままになる場合があります。

データベース関連：MongoDB

MongoDB データベースのサイズが非常に大きい場合、テナント削除がすぐに完了しない場合があります。

- コンソール上では、テナント削除エラーとなります。
- 対策
  - 削除エラーとなった場合は、しばらく時間をあけて再度削除を実施してください。

ディスク容量が不足すると、MongoDB プロセスが異常終了する場合があります。

- mongod が異常終了します。
- 対策
  - 監視ソフトウェアを使用して、ディスク容量不足に陥らないよう監視を行うことを推奨します。
  - 不足しそうな場合は、ディスクを追加するか、シャーディング使用時はシャード追加を行ってデータを分散させてください。

Push通知関連

到達確認

- FCM/GCM/APNs を利用したPush通知機能を提供しておりますが、いずれも到達保障はされません。FCM/GCM／APNs の仕様になります。
- SSE Pushについても到達確認機能は提供しておりません。

- メッセージの到達確認を行うためには、オブジェクトストレージなどを併用して別途作り込む必要があります。

SSE Push サーバ

- RabbitMQ サーバへの接続が完了するまで、SSE Push サーバの起動がブロックする仕様となっています。
- 同一APサーバ上の他のアプリケーションの起動もブロックされますので注意してください。

Cloud Functions 関連

動作中に Function やコードの変更・削除を行った場合、一定時間(数秒程度) Function の呼び出しができない期間が発生します

- ロジックサーバの再起動などを実行する必要があるため
- この期間中は、Function を呼び出す API は 500 Internal Server Error が返ります

Docker コンテナは Function 毎に1個起動されるため、消費メモリ量に注意してください

- コンテナで使用するメモリ量は Function 定義の memorySize で指定します
- サーバは、起動するファンクション数 x 指定メモリ量分のメモリを搭載している必要があります。
- 指定メモリ量が少なすぎるとコンテナ内でスラッシングが発生し性能が大きく劣化する場合があります。Function 実装毎に、メモリサイズ指定を変えながら最適値を探すことを推奨します。

Function 実行中にサーバ(RabbitMQサーバ、Cloud Functionsサーバ)を停止・再起動した場合、Function の処理がエラーとなる場合があります

- タイミングによって、クライアント側には 400 Bad Request もしくは 504 Gateway Timeout が返却されます

Java 9 以降での動作について

2018/3月時点では Java 9 以降には対応していません。

- v7.0 のJava 9 での動作は確認しておりますが、Java 9 は 2018年3月末に EOL になるため、正式対応はしておりません。
- Java 10 以降の動作検証は行っておりません。
- v6.0 以前のサーバは、Java9 では動作しません(サーバ起動が失敗します)
- v6.5 については、JVM起動時に --add-modules java.xml.bind,java.activation オプションの指定が必要になる場合があります。

OpenJDK 11 および Oracle JDK 11/LTS については後日対応の予定です。

- OpenJDK11, Oracle JDK11/LTSリリース(2018年9月予定)後、モバイルバックエンド基盤 v7.0.x での対応を予定しています (v6.x 以下は対象外となります)

サーバ：冗長構成時の注意事項

APIサーバ冗長構成

Nebula API サーバをロードバランサを使用して複数台構成としている場合、以下の点に注意が必要です。

- APIサーバは、DBアクセスを効率化するためにいくつかの情報をメモリ上にキャッシュしています。これらの情報を変更した場合、他APIサーバへの情報伝播に時間を要する場合があります。
- 情報伝播前のサーバにアクセスすると、古い情報が返却される場合があります。
  - 特にバケットの ACL変更や、グループメンバ変更がすぐに反映されない点にご注意ください。
- キャッシュする情報と、最大キャッシュ有効時間は以下の表の通りです。

キャッシュする情報	有効時間(秒)	キャッシュするエントリ数
テナント	180	1000
アプリケーション	180	1000
バケット	60	1000
ユーザ	30	5000
グループ	30	5000
セッショントークン	30	5000
API定義	30	1000
ファンクション定義	30	1000

対策

- 本件は通常使用では問題になることはありませんが、最新の情報が必須の場合は適宜ウェイトを入れて下さい。
- キャッシュ有効時間を短くするようカスタマイズすることは可能です。必要な場合はお問い合わせ下さい。

MongoDB シャーディング

MongoDB シャーディング環境では以下の点に注意が必要です。

- シャード間でデータのマイグレーションを行っている最中にバケットを削除することはできません(エラーになります)。
  - マイグレーションが稼働中かどうかは、mongo シェルの sh.status() で確認できます。
- シャードキー設定後、全APサーバへのシャード情報の伝搬には最大60秒かかります。この間、バケットへのアクセスが制限される場合があります。
- Configサーバがすべて正常稼働している状態でないと、テナントの追加・削除処理を行うことができません。
  - Config サーバ3台のうち1台でも停止していると、テナント追加・削除はエラーになります。その他の操作(バケット作成など)は問題ありません。

- 対策
  - テナント追加・削除、シャードキー設定、バケット作成・削除などの管理操作は、開発時とサービス運用停止時(メンテナンス時)にのみ行うようにしてください。

オフライン・同期機能関連

同期機能の注意事項

- オフライン・同期機能を使用する際は、次の点に注意が必要です。

- 1) ACL
- 2) 同期範囲の設定
- 3) 同期範囲の変更
- 4) 別ユーザログイン
- 5) 論理削除オブジェクト

- 詳細は次ページ以降で説明します。

1) ACL

- ローカルキャッシュに関係するアクセス権変更があっても、同期ではローカルキャッシュへは自動的に反映されません。ユーザのアクセス可能なオブジェクトが増加した場合、増加したオブジェクトを取得できない場合があります。
  - 具体例
    - ユーザの所属するグループの追加や変更
    - バケットに対するアクセス権の追加
  - 解説
    - オブジェクトの ACL を変更した場合は、当該オブジェクトは変更されたとみなされ同期によりサーバからクライアントへ送付されるため問題はありません。
    - グループにユーザを追加すると、オブジェクトには変更がないにも関わらずアクセス不可⇒可にアクセス権限が変化する場合があります。この場合、当該オブジェクトは差分同期の対象とならないため、クライアントへ送付されません。
    - 逆にグループからユーザを削除すると、オブジェクトがアクセス可⇒不可 に変化する場合があります。この場合、当該オブジェクトはローカルキャッシュから削除されず、更新もされないオブジェクトとなってしまいます。
  - 対策
    - ユーザのアクセス権が変わるような操作を行う場合には、ローカルキャッシュを削除するようにしてください。

1) ACL (続き)

具体例の解説

- User A の所属グループが(Group1,Group2)から(Group1,Group3)に変更になった場合、
  - アクセス権が失われた Object Y は、ローカルキャッシュから消えずに残り、アップデートもされなくなってしまいます。
  - アクセス権が追加された Object Z は、クライアントにオブジェクトが送付されず、アプリから存在が見えません。

オブジェクト	オブジェクトにアクセス可能なグループ	User A 変更前のアクセス権	User A 変更後のアクセス権	挙動
Object X	Group1	○	○	正常
Object Y	Group2	○	×	消えなくなる
Object Z	Group3	×	○	見えない

2) 同期範囲の設定

同期範囲を決定するために利用するオブジェクトフィールドは書き換えが起こらないフィールドのみとしてください。

- 具体例
  - 特定のフィールドにIDを入れるなどして、同期範囲を設定する場合、その フィールドの値を変更しないでください。
  - 端末Aでローカルキャッシュにオブジェクトを作成・同期したあと、別の端末Bから そのオブジェクトの ID を端末Aの同期範囲外に変更すると、当該オブジェクトが端末Aのローカルキャッシュから削除されなくなります。
- 解説
  - ID 変更を行ったオブジェクトが同期範囲から外れると、同期時に当該オブジェクトがクライアントに送信されないため、端末Aのローカルキャッシュから削除されなくなります。
- 対策
  - 同期範囲として利用するフィールドの値変更は行わないように注意してください。

3) 同期範囲の変更

同期範囲を変更すると、最終同期日時がリセットされます。

- 最終同期日時がリセットされると、次回同期は差分同期ではなく全データ同期となるため、同期に要する時間が長くなります。
- 最終同期日時の考え方は、p33「オフライン・同期アルゴリズムの解説」を参照ください。

4) 別ユーザログイン

- ローカルキャッシュを取得した際のユーザと異なるユーザでのログインを行うと、 ACLや同期範囲設定の影響で、アクセス可能なオブジェクトへアクセスできない場合があります。
  - 具体例
    - ユーザAでアプリの利用を開始し同期実行後、ユーザBでログインし同期を行うケースを考えます。ユーザAにはアクセス権がなくユーザBにアクセス権があるオブジェクトが存在する場合、同期時にクライアントにはそのオブジェクトが送信されません。
    - ユーザ毎に同期範囲を変える設定にしていた場合、同期範囲設定がローカルキャッシュと異なる条件となってしまうため、同期範囲に使う ID を変更するのと同じ結果となります。
  - 解説
    - ユーザAで同期を行った際には、サーバ上でこのオブジェクトに更新があったとしてもクライアントには送信されません(ユーザAのアクセス権限がないため)。この後、ユーザBでログインして同期しても、当該オブジェクトの更新時刻は最終同期日時より古いため更新されていないとみなされ、同期されません。
  - 対策
    - 別ユーザでログインした場合、キャッシュは消すように実装してください。
    - あるいは、ユーザ単位で ACL の設定を変える場合、運用で異なるユーザでのログインを行わないように管理してください。

5) 論理削除オブジェクト

- サーバ上で論理削除されたオブジェクトは、そのままでは物理的には削除されません。古くなったオブジェクトは定期的に削除する処置が必要です。
  - 詳細は「オブジェクトストレージ: オブジェクト削除」のページを参照ください。

補足: オフライン・同期アルゴリズムの解説

同期はバケット単位に実行されます。

バケット毎に「Pull」⇒「Push」処理が順に実行されます。

- Pull / Push 処理ともに、更新データのみを同期する「差分同期」が行われます。

Pull 処理

- Pull 処理では、サーバ上の更新データがクライアント側に転送、同期されます。
- Pull 時には、サーバに対して a)「同期範囲内」かつ b)「更新時刻が最終同期日時以降」となるデータをクエリし、同期します。
  - 初回同期時は b) の条件は使用されません。
- Pull 処理が行われた際、衝突処理も実行されます。

Push 処理

- Push 処理では、クライアント上の更新データがサーバ側に転送されます。
- ローカルキャッシュ上には、オブジェクト毎に更新フラグ(dirtyフラグ)があり、dirty 状態のオブジェクトのみがサーバに同期されます。
- なお、Push 時には同期範囲は使用されません (全範囲同期)。

文字種・文字列長の制約

文字種・文字列長制約

- モバイルバックエンドサーバにはいくつか文字種・文字列長についての制約があります。
- 詳細は「デベロッパーマニュアル」内の「制限値・スレショルド・文字列制約」を参照ください。


Orchestrating a brighter world
NEC