Alibaba Cloud Table Store

機能

Document Version20190730

目次

1 制限

次の表に、Table Store の制限事項を示します。一部の制限は、推奨値ではなく使用可能な最大値を示しています。 より良いパフォーマンスを確保するために、テーブル構造と単一行のデータサイズを適切に設定してください。

項目	制限	説明
Alibaba Cloud ユーザーアカウントで作成されたインスタンスの数	10	制限を増やすには、チケットを起票し、サポートセンターへお問い合わせください。
インスタンス内のテーブル数	64	制限を増やすには、チケットを起票し、サポートセンターお問い合わせください。
インスタンス名の長さ	3 ～ 16 バイト	大文字と小文字、数字、ハイフンを含めます。 文字で始まり、ハイフンでは終わりません。 「ali」、「ay」、「ots」、「taobao」、「admin」などの単語を含めないでください。
テーブル名の長さ	1 ～ 255 バイト	大文字と小文字、数字、アンダースコアを含めます。 文字またはアンダースコアで始めます。
列名の長さ	1 ～ 255 バイト	大文字と小文字、数字、アンダースコアを含めます。 文字またはアンダースコアで始めます。
プライマリキー列数	1 ～ 4 列	少なくとも 1 列です。

項目	制限	説明
文字列型プライマリキー列の値のサイズ	1 KB	単一のプライマリキー列の文字列型列値は 1 KB に制限されています。
文字列型属性列の値のサイズ	2 MB	単一属性列の文字列型列値は 2 MBに制限されています。
バイナリ型プライマリキー列値のサイズ	1 KB	単一のプライマリキー列のバイナリ型列値は 1 KB に制限されています。
バイナリ型属性列値のサイズ	2 MB	単一属性列のバイナリー型列値は 2 MB に制限されています。
単一行の属性列数	無制限	単一行には、無制限の量の属性列を含めることができます。
1 回のリクエストで書き込まれる属性列の数	1024 列	1 行に 1 つの PutRow 、UpdateRow または BatchWriteRow 要求によって書き込まれた属性列の数
単一行のデータサイズ	無制限	単一行のすべての列名と列値データの合計サイズは無制限です。
単一テーブルに対する予約済み読み書きスループット	0 ～ 5000	制限を増やすには、チケットを起票し、サポートセンターへお問い合わせください。
読み取り要求の columns_to_get パラメーターの列数	0 ～ 128	読み取り要求のデータ行で取得された列の最大数。
テーブルレベル操作 QPS	10	インスタンスに対するテーブルレベル操作の QPS は 10 を超えてはなりません。 表レベルの操作については、「表の操作」をご参照ください。
単一テーブルに対する UpdateTable オペレーションの数	Increase: 無制限 Lower: 無制限	各テーブルの予約済み読み書きスループットは、暦日 (UTC 時間の 00:00:00 から翌日の 00:00:00まで) 内で無制限に増減できます。

項目	制限	説明
単一テーブルの UpdateTable 頻度	2 分ごとに最大 1 回の更新	単一テーブルの予約済み読み書きスループットは、2 分に 1 回の頻度を超えて調整することはできません。
1回の BatchGetRow リクエストで読み取られる行数	100	N/A
1 回の BatchWriteRow 要求によって書き込まれる行数	200	N/A
1 回の BatchWriteRow リクエストのデータサイズ	4 MB	N/A
1 回の GetRange オペレーションによって返されるデータ	5,000 行または 4 MB	1 回の操作で返されるデータは 5000 行または 4 MB を超えることはできません。 そうでなければ、余分なデータは返されたトークンで読み込まれます。
HTTP リクエストボディのデータサイズ	5 MB	N/A

2 SearchIndex

2.1 概要

Table Store は Alibaba Cloud によって開発された分散型マルチモデルデータベースで、強力なデータ処理サービスを提供するように設計されています。 現在、Table Store は大規模なデータストレージと効率的なデータアクセスを提供しています。

SearchIndex は、属性クエリ、あいまい検索およびソートを補完するために導入されました。 SearchIndex がないと、プライマリキーでしかテーブルをクエリできません。 SearchIndex は、さまざまな高度なクエリをサポートする新しいデータ構造です。 これらの高度なクエリには、複数の属性に対するアドホッククエリ、あいまい検索、フルテキストクエリ、ソート、範囲クエリ、入れ子になったクエリおよび空間クエリがあります。

現在、SearchIndex ベータ版は特定のリージョンで利用可能です。 Alibaba Cloud オフィシャルサイトの Table Store コンソールでベータ版を申し込むことができます。 SearchIndex のこのベータ版は無料です。

インデックスの同期

SearchIndex 機能を有効にしたら、テーブルに SearchIndex を作成できます。 テーブルに書き込まれたデータはストレージモジュールに格納されます。 書き込み操作が成功したことを示す応答を受け取ります。 その間、スレッドは、SearchIndex を構築するためにストレージモジュールから非同期にデータを読み取ります。 プロセス全体のレイテンシはミリ秒から数秒です。 インデックス作成時にレイテンシをチェックするために、RPO (Recovery Point Objective) モニタリングを提供します。 SearchIndex を非同期的に作成しても、Table Store の書き込みパフォーマンスには影響しません。

制限事項

SearchIndex は一連の高度なクエリを提供します。 以前のキー値ストレージモデルと比較したとき、次の機能はサポートされていません。

- TTL: Time to Live (TTL) プロパティで指定されたテーブルに SearchIndexes を作成することはできません。
- MaxVersion: Max Versions プロパティで指定されたテーブルに SearchIndexes を作成することはできません。
- カスタムバージョン番号: 単一のバージョン番号で指定されたテーブルでは、書き込みごとにカスタムタイムスタンプを指定できます。 また、バージョン番号が大きいデータは、バージョ

ン番号が小さいデータよりも先に書き込まれます。 バージョン番号が大きいデータはインデックスに書き込まれ、バージョン番号が小さいデータで上書きされる可能性があります。

シナリオ

SearchIndex は、以下のシナリオで使用できます。

- メタデータ
- 時空間データ
- 時系列データ
- 全文検索

詳細については、「より高いクエリパフォーマンスを提供するために、SearchIndex がテーブルストアで利用可能になりました」をご参照ください。

2.2 特徴

現在、Table Store の SearchIndex には次の機能があります。

インデックス

リレーショナルデータベースと NoSQL データベースの両方に対する基本的なクエリ方法は、1 つ以上のプライマリキーに基づいています。 テーブルを属性でクエリする場合は、インデックスを作成する必要があります。 オンラインデータ処理をより強力で汎用性のあるものにするために、Table Store で転置インデックスおよびその他のインデックスを提供しています。

- 転置インデックスは、検索エンジン内の複数のクエリの基本データ構造です。 このデータ構造の目的は、クエリのパフォーマンスを向上させることです。 このため、Table Store は転置インデックスを提供しています。 テーブルの特定の属性に転置インデックスを作成すると、これらの転置インデックスに基づいた複数の属性を組み合わせてアドホッククエリを発行できます。 一方、性別、年齢、列挙などのカーディナリティの低い属性で作成された転置インデックスを使用して、テーブルをすばやくクエリすることもできます。
- 地理空間インデックスデータ構造は、地理的位置などの時空間データをクエリするために使用されます。 このインデックスは、時空間データをクエリするときのクエリパフォーマンスを大幅に向上させます。 このため、Table Store は地理空間インデックスを提供します。 Table Store は、近隣の人物、四角形の点、多角形の点など、さまざまな地理的な点をクエリする機能を備えているため、ビッグデータのふるい分け、Internet of Vehicles (IoV) およびモバイルアプリのための強力なワンストップデータ処理プラットフォームを実現できます。

ソート

ソートは、オンラインデータ処理の一般的な要件です。 最大数、最大、最小、最新など、特定の条件を満たすデータを順番に管理します。 Table Store は、順方向、逆方向、単一条件および複合条件を含む強力なソートも提供します。 これらのソートタイプを使用すると、Table Store に格納されているデータをさまざまな方法でソートできます。

全文検索

転置インデックスを用いて、Table Store でトークナイゼーション操作を実行できます。 この操作に基づいて、全文検索ができます。 ただし、データを呼び出すことしかできません。 相関分析はサポートされていません。

あいまい検索

あいまい検索はリレーショナルデータベースの強力な機能です。 like ステートメントを使用すると、さまざまなユースケースであいまい検索を実行できます。 あいまい検索は HBase などのリレーショナルデータベースではサポートされていません。 現在、Table Store はあいまい検索をサポートしています。 属性に対してあいまい検索を実行するには、その属性に対してのみ転置反転インデックスを作成できます。

プレフィックスクエリ

あいまい検索の他に、Table Store はプレフィックスクエリもサポートしています。

入れ子になったクエリ

オンラインデータ処理における平面データを除いて、画像のラベルのような多次元のデータを想像することができます。多数の画像がデータベースに格納され、それぞれの画像が家、自動車などの複数の要素を持つと想定します。 画像の各要素には、互いに異なるスコアがあります。 スコアは、画像内の要素のサイズと位置に従って評価されます。 つまり、それぞれの画像には複数のラベルがあります。 それぞれのラベルには名前と重みスコアがあります。 クエリ基準としてラベルを指定した場合、これは入れ子になったクエリを発行します。 入れ子になったクエリは、複数のディメンションを持つデータのクエリを容易にします。

カーディナリティ

前のクエリから返された結果セットで、カーディナリティの低い属性を取得することがあります。 カーディナリティを使用すると、高いカーディナリティを持つように属性値の出現回数を最大にするように指定できます。

総行数

SearchIndex を指定すると、行の総数が結果セットに返されます。 インデックスをクエリするときにクエリ基準を指定しないと、SearchIndex の行の総数が返されます。 データベースへの新しいデータの書き込みを停止してすべての属性にインデックスを作成すると、テーブルの行の総数が返されます。 この機能はデータ検証とデータ管理に適用されます。

SearchIndexes の使用中に問題が発生した場合は、次のいずれかの方法で Alibaba カスタマーサービスにお問い合わせください。

- Alibaba Cloud 公式サイトの管理コンソールでチケットを起票し、サポートセンターにお問い合わせください。
- 詳細については、ID 11789671 を入力して Table Store テクニカルサポートの DingTalk グループに参加してください。

3 グローバルセカンダリインデックス

3.1 概要

グローバルセカンダリインデックスの使用を開始する前に、次の基本概念、制限事項、および注意事項を理解しておく必要があります。

基本概念

用語	説明
インデックス	テーブルのいくつかの属性にインデックスを作成できます。 インデックスは読み取り専用です。
定義済みの属性	Table Store はスキーマフリーモデルを使用します。 スキーマに固定数の属性を指定する代わりに、無制限の数の属性を行に書き込むことができます。 テーブルを作成するときに、定義済みの属性とそのデータ型を指定することもできます。
シングルフィールドインデックス	1 つのテーブル属性にのみインデックスを作成できます。
複合インデックス	テーブルの複数の属性グループにインデックスを作成できます。 さらに、これらの属性で並べ替えることができます。
射影属性	定義済みの属性をインデックスにコピーすることができます。 これらの属性はいずれもプライマリキーとして指定されていません。
自動補完	テーブルのプライマリキーは、インデックスを作成すると自動的にインデックスにコピーされます。

制限事項

- 1 つのテーブルに最大 16 個のインデックスを作成できます。
- インデックスは最大 4 つのインデックス付き属性を持つことができます。 インデックス付き属性は、基本テーブルのプライマリキーと定義済みの属性で構成されています。
- データ型として、インデックス付き属性に Integer 型、String 型または Binary 型を指定できます。 属性の制限はプライマリキーの場合と同じです。

- 属性に Integer 型または Binary 型を指定した場合、これらのデータ型のサイズ制限は、ベース テーブルのプライマリキーの場合と同じです。
- 属性に String 型または Binary 型を指定した場合、属性の制限はベーステーブルの場合と同じです。
- TTL プロパティで設定されているテーブルにインデックスを作成することはできません。 そのようなインデックスを作成する場合は、DingTalk で Table Store テクニカルサポートに連絡してください。
- Max Versions プロパティで設定されているテーブルにインデックスを作成することはできません。
- インデックスに Stream を使用することはできません。
- 自動インクリメントのプライマリキーを使用してテーブルにインデックスを作成することはできません。 そのようなインデックスがある場合は、Table Store テクニカルサポートへのチケットを起票し、サポートセンターにお問い合わせください。

注意事項

- Table Store は各インデックスに対して自動補完演算を実行します。 インデックスをスキャンするときは、プライマリキー値の範囲を指定する必要があります。 一般に、範囲は負の無限大から正の無限大です。 たとえば、テーブルにプライマリキー `PK0` および `PK1` 、さらに定義済みの属性 `Defined0` が含まれているとします。

  `Defined0` 属性にインデックスを作成すると、Table Store は `Defined0` 、 `PK0` 、そしてプライマリキー `PK1` を持つインデックスを作成します。 `Defined0` および `PK1` 属性にインデックスを作成すると、Table Store は `Defined0` およびプライマリキー `PK1` 、 `PK0` を持つインデックスを作成します。 `PK` 属性にインデックスを作成すると、Table Store はプライマリキー `PK1` と `PK0` を持つインデックスを作成します。 インデックスを作成するときは、インデックス付けする属性のみを指定できます。 Table Store は後でインデックスに対して自動補完操作を実行します。 たとえば、プライマリキー PK0 と PK1 および事前定義された属性 Defined0 を持つテーブルだとします。

  - Defined0 属性にインデックスを作成すると、Table Store はプライマリキー Defined0、PK0 および PK1 を持つインデックスを作成します。
  - PK1 にインデックスを作成すると、Table Store はプライマリキー PK0 と PK1 を持つインデックスを作成します。
- テーブルの定義済み属性をインデックス属性として指定できます。 インデックス属性として事前定義属性を指定すると、ベース テーブルをクエリする代わりに、このインデックスをクエリして属性値を読み取ることができます。 この設定ではストレージコストが増加します。 事

前定義属性をインデックス属性として指定しなかった場合は、ベーステーブルを照会する必要があります。 要件とコストに基づいてクエリモードを選択できます。

- 設定によってインデックスの更新が遅くなる可能性があるため、インデックスのプライマリキーの最初の属性として時刻または日付に関連する属性を指定することは推奨しません。 ハッシュ関数の入力として時刻または日付に関連する属性を入力し、そのハッシュ関数によって処理された属性にインデックスを作成することを推奨します。 このような要件がある場合は、DingTalk で Table Store テクニカルサポートに連絡してください。
- インデックスのプライマリキーの最初の属性として、カーディナリティの低い属性、列挙可能な値を持つ属性を定義することは推奨しません。 たとえば、`gender` という設定値は、インデックスの水平方向のスケーラビリティを制限し、データ書き込みパフォーマンスの低下につながります。

3.2 はじめに

Table Store のグローバルセカンダリインデックスには、次の機能があります。

- テーブルとテーブルインデックス間の非同期データの同期をサポートします。 通常のネットワーク状態では、データ同期レイテンシはミリ秒単位です。
- シングルフィールドインデックス、複合インデックスおよびカバリングインデックスをサポートします。 事前定義属性は、テーブル内で事前に指定された属性です。 定義済みの属性またはテーブルのプライマリキーにインデックスを作成できます。 さらに、テーブルの定義済み属性をインデックス属性として指定するか、属性を指定しないことを選択できます。 インデックス属性として事前定義属性を指定すると、ベース テーブルをクエリする代わりに、このインデックスを直接クエリして基本テーブルからデータを取り出すことができます。 たとえば、

ベース テーブルには、3つのプライマリキー PK0、PK1 および PK2 が含まれています。 さらに、テーブルには 3 つの定義済み属性 Defined0、Defined1 および Defined2 があります。

- 属性を指定せずに、PK2 にインデックスを作成できます。
- PK2 にインデックスを作成し、属性として Defined0 を指定できます。
- 属性を指定せずに、PK3 および PK2 にインデックスを作成できます。
- PK3 と PK2 にインデックスを作成し、属性として Defined0 を指定できます。
- PK2、PK1 および PK3 にインデックスを作成し、属性として Defined0、Defined1 および Defined2 を指定できます。
- 属性を指定せずに Defined0 にインデックスを作成できます。
- Define0 と PK1 にインデックスを作成し、Defined1 を属性として指定できます。
- 属性を指定せずに Define1 と Define0 にインデックスを作成できます。
- Define1 と Define0 にインデックスを作成し、Defined2 を属性として指定できます。

・ スパースインデックスをサポートします。 ベース テーブルの事前定義属性をインデックス属性として指定できます。 事前定義された属性がベース テーブルの行から除外されているにもかかわらず、すべてのプライマリキーが存在する場合でも、この行は索引付けされます。 ただし、行が 1 つ以上のインデックス付き属性を除外すると、この行はインデックス付けされません。 たとえば、ベース テーブルには、PK0、PK1 および PK2 の 3 つのプライマリキーが含まれています。 さらに、テーブルには 3 つの定義済み属性 Defined0、Defined1 および Defined2 があります。 Defined0 と Defined1 にインデックスを作成し、Defined2 を属性として指定できます。

- インデックスには、Defined2 属性を除外し、定義済み属性 Defined0 および Defined1 を含むベース テーブルの行が含まれます。
- ベース テーブルの行が Defined1 を除外し、定義済みの属性 Defined0 と Defined2 を含む場合、この行はインデックスから除外されます。

・ 既存のベース テーブルに対するインデックスの作成および削除をサポートします。 今後のバージョンでは、ベース テーブルにこのインデックスを作成すると、ベース テーブルの既存のデータがインデックスにコピーされます。

・ インデックスをクエリしても、作成されたインデックスのベース テーブルに対するクエリは自動的には実行されません。 ベース テーブルをクエリする必要があります。 この機能は今後のバージョンでサポートされる予定です。

Table Store のグローバルセカンダリインデックス機能は、現在、中国 (張家口) リージョンでご利用いただけます。 トライアルについては DingTalk で Table Store テクニカルサポートに連絡するか、詳細については ID 111789671 を入力して DingTalk グループに参加してください。

3.3 シナリオ

グローバルセカンダリインデックスは、Table Store の新しい機能です。 テーブルを作成すると、プライマリインデックスはすべてのプライマリキーで構成されます。 Table Store はプライマリキーを使用して、一意にテーブル内の各行を識別します。 ただし、より多くのシナリオでは、属性、プライマリキーまたは最初の列からではないプライマリキーでテーブルをクエリする必要があります。 インデックスが不十分な場合、テーブル全体をスキャンしてフィルター条件を設定することによってのみ結果を取得できます。 大量のデータを含むテーブルをクエリした後に結果がほとんど得られない場合は、クエリによってリソースが過剰に消費される可能性があります。

Table Store のグローバルセカンダリインデックス機能は、 DynamoDB GSI および HBase Phoenix の機能と似ています。 指定した 1 つ以上の属性を使ってインデックスを作成できます。 さらに、作成したインデックス内のデータを指定した属性で並べ替えることができます。 ベース テーブルに書き込むすべてのデータは、ベース テーブルに作成されたインデックスと非同期的に同期されます。 ベース テーブルにデータを書き込むだけでよく、このベース テーブルに作成されたインデックスをクエリすることができます。 この設定により、ほとんどのシナリオでクエリのパフォーマンスが大幅に向上します。 たとえば、一般的な通話履歴クエリのベース テーブルを次のように作成できます。

CellNumber	StartTime (Unix タイムスタンプ)	CalledNumber	Duration	BaseStationNumber
123456	1532574644	654321	60	1
234567	1532574714	765432	10	1
234567	1532574734	123456	20	3
345678	1532574795	123456	5	2
345678	1532574861	123456	100	2
456789	1532584054	345678	200	3

- `CellNumber` および `StartTime` はプライマリキーで、それぞれ 通話番号 および 通話開始時刻を表しています。
- `CalledNumber` 、 `Duration` および `BaseStationNumber` は定義済み属性で、着信番号、通話時間および基地局番号を表しています。

通話を終了すると、通話情報がこのテーブルに書き込まれます。それぞれ CalledNumb er および BaseStatio nNumber にグローバルセカンダリインデックスを作成して、さまざまなクエリ要件を満たすことができます。インデックスの作成方法の詳細については、「付録」の例をご参照ください。

次のクエリ要件があるとします。

- CellNumber の値が 234567 と一致する行を取得します。

  Table Store のプライマリキーでデータを並べ替えることができます。さらに、 getRange メソッドを呼び出して、データを順次スキャンすることもできます。 getRange メソッドを呼び出す場合は、 234567 を PK0 (CellNumber) の最小値および最大値として指定する必要があります。一方、 0 をPK1 (StartTime) の最小値として指定し、 INT_MAX を PK1の最大値として指定する必要があります。その後、ベース テーブルをクエリできます。

```
 private   static   void   getRangeFr  omMainTabl  e ( SyncClient
 client ,  long   cellNumber )
{
     RangeRowQu  eryCriteri  a   rangeRowQu  eryCriteri  a  =  new
 RangeRowQu  eryCriteri  a ( TABLE_NAME );

    // プライマリキーを指定できます。
     PrimaryKey  Builder   startPrima  ryKeyBuild  er  =
 PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
     startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
 PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Value . fromLong ( cellNumber
 ));
     startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
 PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Value . fromLong ( 0 ));
     rangeRowQu  eryCriteri  a . setInclusi  veStartPri  maryKey (
 startPrima  ryKeyBuild  er . build ());

    // プライマリキーを指定できます。
     PrimaryKey  Builder   endPrimary  KeyBuilder  =  PrimaryKey
 Builder . createPrim  aryKeyBuil  der ();
     endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
 Y_NAME_1 ,  PrimaryKey  Value . fromLong ( cellNumber ));
     endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
 Y_NAME_2 ,  PrimaryKey  Value . INF_MAX );
     rangeRowQu  eryCriteri  a . setExclusi  veEndPrima  ryKey (
 endPrimary  KeyBuilder . build ());

     rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

     String   strNum  =  String . format ("% d ",  cellNumber );
     System . out . println (" A   cell   number  " +  strNum  + "
 makes   the   following   calls :");
     while  ( true ) {
         GetRangeRe  sponse   getRangeRe  sponse  =  client .
 getRange ( new   GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
         for  ( Row   row  :  getRangeRe  sponse . getRows ()) {
             System . out . println ( row );
        }
```

```
        //  nextStartP  rimaryKey  の値が  null  値でない場合は、ベー
ス テーブルから引き続きデータを読み取ることができます。
         if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
 =  null ) {
             rangeRowQu  eryCriteri  a . setInclusi  veStartPri
 maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
        }  else  {
             break ;
        }
    }
}
```

- CalledNumberの値が `123456` である行を取得する場合。

  Table Store はすべての行をプライマリキーでソートします。 CalledNumber は事前定義された属性であるため、この属性によってテーブルを直接照会することはできません。 そのため、 `CalledNumb er` に作成されたインデックスをクエリできます。

  `IndexOnBeC alledNumbe r` :

PK0	PK1	PK2
CalledNumber	CellNumber	StartTime
123456	234567	1532574734
123456	345678	1532574795
123456	345678	1532574861
654321	123456	1532574644
765432	234567	1532574714
345678	456789	1532584054

   注 :

  Table Store はインデックスのプライマリキーを自動補完します。 このインデックスを作成するとき、Table Store はベーステーブルのすべてのプライマリキーをこのベーステーブルに作成されたインデックスに追加します。 したがって、インデックスには 3 つのプライマリキーが含まれています。

  `IndexOnBeC alledNumbe r` は CalledNumber に作成されたインデックスなので、このインデックスを直接クエリして結果を取得できます。

```
private  static  void  getRangeFr  omIndexTab  le ( SyncClient
client ,  long  cellNumber ) {
    RangeRowQu  eryCriteri  a  rangeRowQu  eryCriteri  a  =  new
RangeRowQu  eryCriteri  a ( INDEX0_NAM  E );

   // プライマリキーを指定できます。
```

```
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_1 , PrimaryKey Value . fromLong ( cellNumber
));
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN );
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . INF_MAX );
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // プライマリキーを指定できます。
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_1 , PrimaryKey Value . fromLong ( cellNumber ));
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX );
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX );
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    String strNum = String . format ("% d ", cellNumber );
    System . out . println (" A cell number " + strNum + "
was called by the following numbers ");
    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            System . out . println ( row );
        }

        // nextStartP rimaryKey の値が null 値でない場合は、ベー
ス テーブルから引き続きデータを読み取ることができます。
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
= null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
   }
```

```
}
```

- BaseStationNumber の値が `002` に、また、StartTime の値が `1532574740` に一致する行を取得する場合。

  このクエリでは、`BaseStatio nNumber` および `StartTime` の両方を条件として指定しています。したがって、`BaseStatio nNumber` および `StartTime` に複合インデックスを作成できます。

`IndexOnBas eStation1`:

PK0	PK1	PK2
BaseStationNumber	StartTime	CellNumber
001	1532574644	123456
001	1532574714	234567
002	1532574795	345678
002	1532574861	345678
003	1532574734	234567
003	1532584054	456789

`IndexOnBas eStation1` インデックスをクエリできます。

```
private static void getRangeFr omIndexTab le ( SyncClient
client ,
                                        long baseStatio
nNumber ,
                                        long startTime ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX1_NAM E );

   // プライマリキーを指定できます。
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_3 , PrimaryKey Value . fromLong ( baseStatio
nNumber ));
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . fromLong ( startTime
));
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN );
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // プライマリキーを指定できます。
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_3 , PrimaryKey Value . fromLong ( baseStatio nNumber ));
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX );
```

```
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_1 ,  PrimaryKey  Value . INF_MAX );
    rangeRowQu  eryCriteri  a . setExclusi  veEndPrima  ryKey (
endPrimary  KeyBuilder . build ());

    rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

    String  strBaseSta  tionNum  =  String . format ("% d ",
baseStatio  nNumber );
    String  strStartTi  me  =  String . format ("% d ",  startTime
);
    System . out . println (" All  called  numbers  forwarded
  by  the  base  station " +  strBaseSta  tionNum  + "  that
start  from " +  strStartTi  me  + " are  listed  as  follows
:");
    while  ( true ) {
        GetRangeRe  sponse  getRangeRe  sponse  =  client .
getRange ( new  GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
        for  ( Row  row  :  getRangeRe  sponse . getRows ()) {
            System . out . println ( row );
        }

        //  nextStartP  rimaryKey  値が  null  値でない場合は、ベー
ス テーブルから引き続きデータを読み取ることができます。
        if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
=  null ) {
            rangeRowQu  eryCriteri  a . setInclusi  veStartPri
maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
        }  else  {
            break ;
        }
    }
}
```

- **BaseStationNumber の値 `003` が `1532574861` から `1532584054` の範囲内の StartTime 値と一致する行を取り出す場合。期間だけが行に表示されます。**

  このクエリでは、 `BaseStatio nNumber` および `StartTime` の両方を条件として指定します。 `Duration` のみ結果セットにが表示されます。最後のインデックスに対してクエリを発行してから、ベーステーブルをクエリして Duration を取得することができます。

```
private  static  void  getRowFrom  IndexAndMa  inTable (
SyncClient  client ,
                                            long  baseStatio
nNumber ,
                                            long  startTime ,
                                            long  endTime ,
                                            String  colName )
{
    RangeRowQu  eryCriteri  a  rangeRowQu  eryCriteri  a  =  new
RangeRowQu  eryCriteri  a ( INDEX1_NAM  E );

   // プライマリキーを指定できます。
    PrimaryKey  Builder  startPrima  ryKeyBuild  er  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
DEFINED_CO  L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio
nNumber ));
```

```
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Value . fromLong ( startTime
));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Value . INF_MIN );
    rangeRowQu  eryCriteri  a . setInclusi  veStartPri  maryKey (
startPrima  ryKeyBuild  er . build ());

   // プライマリキーを指定できます。
    PrimaryKey  Builder  endPrimary  KeyBuilder  =  PrimaryKey
Builder . createPrim  aryKeyBuil  der ();
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( DEFINED_CO
L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio  nNumber ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_2 ,  PrimaryKey  Value . fromLong ( endTime ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_1 ,  PrimaryKey  Value . INF_MAX );
    rangeRowQu  eryCriteri  a . setExclusi  veEndPrima  ryKey (
endPrimary  KeyBuilder . build ());

    rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

    String  strBaseSta  tionNum  =  String . format ("% d ",
baseStatio  nNumber );
    String  strStartTi  me  =  String . format ("% d ",  startTime
);
    String  strEndTime  =  String . format ("% d ",  endTime );

    System . out . println (" The  list  of  calls  forwarded
  by  the  base  station " +  strBaseSta  tionNum  + " from "
+  strStartTi  me  + " to " +  strEndTime  + " is  listed  as
follows :");
    while  ( true ) {
        GetRangeRe  sponse  getRangeRe  sponse  =  client .
getRange ( new  GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
        For  ( Row  row :  fig .  getrows  (){
            PrimaryKey  curIndexPr  imaryKey  =  row . getPrimary
Key ();
            PrimaryKey  Column  mainCalled  Number  =  curIndexPr
imaryKey . getPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_1 );
            PrimaryKey  Column  callStartT  ime  =  curIndexPr
imaryKey . getPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_2 );
            PrimaryKey  Builder  mainTableP  KBuilder  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
            mainTableP  KBuilder . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_1 ,  mainCalled  Number . getValue ());
            mainTableP  KBuilder . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_2 ,  callStartT  ime . getValue ());
            PrimaryKey  mainTableP  K  =  mainTableP  KBuilder .
build (); //  You  can  specify  primary  keys  for  the
base  table .

            // ベース テーブルをクエリすることができます。
            SingleRowQ  ueryCriter  ia  criteria  =  new
SingleRowQ  ueryCriter  ia ( TABLE_NAME ,  mainTableP  K );
            criteria . addColumns  ToGet ( colName ); //  You  can
  read  the  Duration  attribute  value  of  the  base
table .
            // 最新のデータバージョンが読み込まれることを示すために  1  を指定
できます。
            criteria . setMaxVers  ions ( 1 );
            GetRowResp  onse  getRowResp  onse  =  client . getRow
( new  GetRowRequ  est ( criteria ));
            Row  mainTableR  ow  =  getRowResp  onse . getRow ();
```

```
            System . out . println ( mainTableR  ow );
        }

        //  nextStartP  rimaryKey  値が  null  値でない場合は、ベー
ス テーブルから引き続きデータを読み取ることができます。
         if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
 =  null ) {
            rangeRowQu  eryCriteri  a . setInclusi  veStartPri
 maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
        }  else  {
            break ;
        }
    }
}
```

クエリのパフォーマンスを向上させるために、 `BaseStatio nNumber` および `StartTime` に複合インデックスを作成できます。 `Duration` をこのインデックスの属性として指定できます。

以下のインデックスが作成されます。

`IndexOnBas eStation2` :

PK0	PK1	PK2	Defined0
BaseStatio nNumber	StartTime	CellNumber	Duration
001	1532574644	123456	600
001	1532574714	234567	10
002	1532574795	345678	5
002	1532574861	345678	100
003	1532574734	234567	20
003	1532584054	456789	200

`IndexOnBas eStation2` インデックスをクエリできます:

```
private   static   void   getRangeFr  omIndexTab  le ( SyncClient
client ,
                                        long   baseStatio
nNumber ,
                                        long   startTime ,
                                        long   endTime ,
                                        String   colName ) {
    RangeRowQu  eryCriteri  a   rangeRowQu  eryCriteri  a  =  new
RangeRowQu  eryCriteri  a ( INDEX2_NAM  E );

   // プライマリキーを指定できます。
    PrimaryKey  Builder   startPrima  ryKeyBuild  er  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
```

```
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
DEFINED_CO  L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio
nNumber ));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Value . fromLong ( startTime
));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Value . INF_MIN );
    rangeRowQu  eryCriteri  a . setInclusi  veStartPri  maryKey (
startPrima  ryKeyBuild  er . build ());

   // プライマリキーを指定できます。
    PrimaryKey  Builder  endPrimary  KeyBuilder  =  PrimaryKey
Builder . createPrim  aryKeyBuil  der ();
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( DEFINED_CO
L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio  nNumber ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_2 ,  PrimaryKey  Value . fromLong ( endTime ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_1 ,  PrimaryKey  Value . INF_MAX );
    rangeRowQu  eryCriteri  a . setExclusi  veEndPrima  ryKey (
endPrimary  KeyBuilder . build ());

   // 読み込む属性名を指定できます。
    rangeRowQu  eryCriteri  a . addColumns  ToGet ( colName );

    rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

    String  strBaseSta  tionNum  =  String . format ("% d ",
baseStatio  nNumber );
    String  strStartTi  me  =  String . format ("% d ",  startTime
);
    String  strEndTime  =  String . format ("% d ",  endTime );

    System . out . println (" The  duration  of  calls
forwarded  by  the  base  station " +  strBaseSta  tionNum  + "
from " +  strStartTi  me  + " to " +  strEndTime  + " is  listed
as  follows :");
    while  ( true ) {
        GetRangeRe  sponse  getRangeRe  sponse  =  client .
getRange ( new  GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
        for  ( Row  row  :  getRangeRe  sponse . getRows ()) {
            System . out . println ( row );
       }

       //  nextStartP  rimaryKey  値が  null  値でない場合は、ベー
ス テーブルから引き続きデータを読み取ることができます。
        if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
=  null ) {
            rangeRowQu  eryCriteri  a . setInclusi  veStartPri
maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
       }  else  {
            break ;
       }
   }
}
```

したがって、 `Duration` をインデックス属性として指定しないと、ベーステーブルをクエリして Duration を取得する必要があります。 しかし、 `Duration` をインデックス属性と

して指定した場合、この属性データはベース テーブルとインデックスに保管されます。 この設定では、ディスクスペースの消費を犠牲にしてクエリのパフォーマンスが向上します。

- 結果セットから、合計通話時間、平均通話時間、最大通話時間および最小通話時間の値を取得する場合。この結果セットは、StartTime の値が `1532574861` から `1532584054` の間にある `003` という BaseStationNumber の値です。

  最後のクエリと比較して、各通話時間に対してリターンは必要ありません。 ただし、期間統計には リターンが必要です。 最後のクエリと同じ方法で結果を取得できます。 その後、Duration 計算をして必要な結果を得ることができます。 さらに、SQL-on-OTS で SQL ステートメントを実行して統計を取得することもできます。 SQL-on-OTS を有効にする方法の詳細については、「Table StoreのOLAP: Data Lake Analytics でのサーバーレス SQL ビッグデータ分析」をご参照ください。 SQL-on-OTS では、ほとんどの MySQL 構文を使用できます。 さらに、SQL-on-OTS を使用すると、ビジネスに適した複雑な計算を簡単に処理できます。

3.4 グローバルセカンダリインデックス用の Java SDK

このセクションでは、Java SDK で `createTable` メソッドと `scanFromIndex` メソッドを呼び出して、次の操作を実行することができます。

- ベーステーブルとこのベーステーブルのインデックスを同時に作成できます。

```
 private   static   void   createTabl  e ( SyncClient   client ) {
     TableMeta   tableMeta  =  new   TableMeta ( TABLE_NAME );
     tableMeta . addPrimary  KeyColumn ( new   PrimaryKey  Schema (
 PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Type . STRING )); // ベーステーブ
ルのプライマリキーを指定できます。
     tableMeta . addPrimary  KeyColumn ( new   PrimaryKey  Schema (
 PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Type . INTEGER )); // #主表#置
 PK 列
     tableMeta . addDefined  Column ( new   DefinedCol  umnSchema (
 DEFINED_CO  L_NAME_1 ,  DefinedCol  umnType . STRING )); // ベース
テーブルに事前定義属性を指定できます。
     tableMeta . addDefined  Column ( new   DefinedCol  umnSchema (
 DEFINED_CO  L_NAME_2 ,  DefinedCol  umnType . INTEGER )); // ベース
テーブルに事前定義属性を指定できます。
     tableMeta . addDefined  Column ( new   DefinedCol  umnSchema (
 DEFINED_CO  L_NAME_3 ,  DefinedCol  umnType . INTEGER )); // ベース
テーブルに事前定義属性を指定できます。

     int   timeToLive  = - 1 ; // データが期限切れにならないように、- 1
  を  Time   To   Live  ( TTL ) の値として指定することができます。
     int   maxVersion  s  =  1 ; // 最大バージョン番号 ベース テーブルに
  1  つ以上のインデックスがある場合は、バージョン値として  1  だけを指定できま
す。

     TableOptio  ns   tableOptio  ns  =  new   TableOptio  ns (
 timeToLive ,  maxVersion  s );
```

```
    ArrayList < IndexMeta >  indexMetas  =  new   ArrayList <
 IndexMeta >();
    IndexMeta   indexMeta  =  new   IndexMeta ( INDEX_NAME ); // イ
ンデックスを作成できます。
    indexMeta . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_1
 ); // ベーステーブルの  DEFINED_CO  L_NAME_1  をインデックスのプライマリ
キーとして指定できます。
    indexMeta . addDefined  Column ( DEFINED_CO  L_NAME_2 ); // ベー
ステーブルの  DEFINED_CO  L_NAME_2  をインデックスのプライマリキーとして指定
できます。
    indexMetas . add ( indexMeta ); // インデックスをベーステーブルに追加
できます。

    CreateTabl  eRequest   request  =  new   CreateTabl  eRequest (
 tableMeta ,  tableOptio  ns ,  indexMetas ); // ベーステーブルを作成で
きます。

    client . createTabl  e ( request );
}
```

- ベース テーブルにインデックスを作成できます。

```
 private   static   void   createInde  x ( SyncClient   client ) {
    IndexMeta   indexMeta  =  new   IndexMeta ( INDEX_NAME ); // イ
ンデックスを作成できます。
    indexMeta . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_2 ); //
 DEFINED_CO  L_NAME_2  をインデックスのプライマリキーの最初の属性として指定で
きます。
    indexMeta . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_1 ); //
 EFINED_COL  _NAME_1  をインデックスのプライマリキーの  2  番目の属性として
指定できます。
    CreateInde  xRequest   request  =  new   CreateInde  xRequest (
 TABLE_NAME ,  indexMeta ,  false ); // ベーステーブルにインデックスを作成
できます。
    client . createInde  x ( request ); // インデックスを作成できます。
}
```

注：

現時点では、ベーステーブルにインデックスを作成すると、ベーステーブルの既存のデータはインデックスにコピーされません。 このインデックスを作成した後、新しく作成されたインデックスには増分データのみが含まれます。 増分データの詳細については、DingTalk の Table Store テクニカルサポートにお問い合わせください。

- インデックスを削除できます。

```
 private   static   void   deleteInde  x ( SyncClient   client ) {
    DeleteInde  xRequest   request  =  new   DeleteInde  xRequest (
 TABLE_NAME ,  INDEX_NAME ); // ベーステーブルとインデックスの名前を指定で
きます。
    client . deleteInde  x ( request ); // インデックスを削除できます。
```

```
}
```

- インデックスからデータを読み取ることができます。

  結果に返される属性がインデックスに含まれている場合は、インデックスから直接データを取得できます。

```
private static void scanFromIn dex ( SyncClient client ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX_NAME ); // インデックス名前を指定できます。

    // 開始プライマリキーを指定できます。
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_1 , PrimaryKey Value . INF_MIN ); // インデックスのプライマリキーの最小値を指定できます。
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN ); // ベース テーブルのプライマリキーの最小値を指定できます。
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . INF_MIN ); // ベース テーブルのプライマリキーの最小値を指定できます。
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

    // 終了プライマリキーを指定できます。
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_1 , PrimaryKey Value . INF_MAX ); // インデックス属性の最大値を指定できます。
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX ); // ベーステーブルのプライマリキーの最大値を指定できます。
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX ); // ベーステーブルのプライマリキーの最大値を指定できます。
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    System . out . println (" The results returned from an
  index are as follows :");
    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            System . out . println ( row );
        }

        // nextStartP rimaryKey 値が null 値でない場合は、ベーステーブルから引き続きデータを読み取ることができます。
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
= null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
```

```
    }
}
```

結果に返される属性がインデックスに含まれていない場合は、ベーステーブルを照会する必要があります。

```
private static void scanFromIn dex ( SyncClient client ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX_NAME ); // インデックス名を指定できます。

    // 開始プライマリキーを指定できます。
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_1 , PrimaryKey Value . INF_MIN ); // インデックスのインデックス付き属性の最小値を指定できます。
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN ); // ベーステーブルのプライマリキーの最小値を指定できます。
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . INF_MIN ); // ベーステーブルのプライマリキーの最小値を指定できます。
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

    // 終了プライマリキーを指定できます。
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_1 , PrimaryKey Value . INF_MAX ); // インデックスのインデックス付き属性の最大値を指定できます。
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX ); // ベース テーブルのプライマリキーの最大値を指定できます。
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX ); // ベーステーブルのプライマリキーの最大値を指定できます。
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            PrimaryKey curIndexPr imaryKey = row . getPrimary
Key ();
            PrimaryKey Column pk1 = curIndexPr imaryKey .
getPrimary KeyColumn ( PRIMARY_KE Y_NAME1 );
            PrimaryKey Column pk2 = curIndexPr imaryKey .
getPrimary KeyColumn ( PRIMARY_KE Y_NAME2 );
            PrimaryKey Builder mainTableP KBuilder =
PrimaryKey Builder . createPrim aryKeyBuil der ();
            mainTableP KBuilder . addPrimary KeyColumn (
PRIMARY_KE Y_NAME1 , pk1 . getValue ());
            mainTableP KBuilder . addPrimary KeyColumn (
PRIMARY_KE Y_NAME2 , ke2 . getValue ());
            PrimaryKey mainTableP K = mainTableP KBuilder .
build (); // ベーステーブルのインデックスプライマリキーを指定できます。
```

```
            // ベーステーブルをクエリできます。
             SingleRowQ  ueryCriter  ia   criteria  =  new
 SingleRowQ  ueryCriter  ia ( TABLE_NAME ,  mainTableP  K );
             criteria . addColumns  ToGet ( DEFINED_CO  L_NAME3
 ); //  You   can   read   the   DEFINED_CO  L_NAME3   attribute
 from   the   base   table .
            // 最新のデータバージョンを取得できます。
             criteria . setMaxVers  ions ( 1 );
             GetRowResp  onse   getRowResp  onse  =  client . getRow
 ( new   GetRowRequ  est ( criteria ));
             Row   mainTableR  ow  =  getRowResp  onse . getRow ();
             System . out . println ( row );
        }

        //  nextStartP  rimaryKey  の値が  null  値でない場合は、ベース
テーブルから引き続きデータを読み取ることができます。
         if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
 =  null ) {
             rangeRowQu  eryCriteri  a . setInclusi  veStartPri
 maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
        }  else  {
             break ;
        }
    }
}
```

3.5 付録

次のようにテーブルとインデックスを作成できます。

```
private   static   final   String   TABLE_NAME  = " CallRecord  Table
";
    private   static   final   String   INDEX0_NAM  E  = "
IndexOnBeC  alledNumbe  r ";
    private   static   final   String   INDEX1_NAM  E  = "
IndexOnBas  eStation1 ";
    private   static   final   String   INDEX2_NAM  E  = "
IndexOnBas  eStation2 ";
    private   static   final   String   PRIMARY_KE  Y_NAME_1  = "
CellNumber ";
    private   static   final   String   PRIMARY_KE  Y_NAME_2  = "
StartTime ";
    private   static   final   String   DEFINED_CO  L_NAME_1  = "
CalledNumb  er ";
    private   static   final   String   DEFINED_CO  L_NAME_2  = "
Duration ";
    private   static   final   String   DEFINED_CO  L_NAME_3  = "
BaseStatio  nNumber ";

    private   static   void   createTabl  e ( SyncClient   client ) {
        TableMeta   tableMeta  =  new   TableMeta ( TABLE_NAME );
        tableMeta . addPrimary  KeyColumn ( new   PrimaryKey  Schema
( PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Type . INTEGER ));
        tableMeta . addPrimary  KeyColumn ( new   PrimaryKey  Schema
( PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Type . INTEGER ));
        tableMeta . addDefined  Column ( new   DefinedCol  umnSchema
( DEFINED_CO  L_NAME_1 ,  DefinedCol  umnType . INTEGER ));
        tableMeta . addDefined  Column ( new   DefinedCol  umnSchema
( DEFINED_CO  L_NAME_2 ,  DefinedCol  umnType . INTEGER ));
        tableMeta . addDefined  Column ( new   DefinedCol  umnSchema
( DEFINED_CO  L_NAME_3 ,  DefinedCol  umnType . INTEGER ));
```

```
        int   timeToLive  = - 1 ; // データの有効期限が切れるまでの時間です。 Time   To   Live  ( TTL ) として - 1  を指定して、データの有効期限が切れないようにすることができます。 単位は秒です。 テーブルに  1  つ以上のインデックスがある場合は、 TTL  値として - 1  を指定する必要があります。
        int   maxVersion  s  =  1 ; //  最大バージョン数です。 テーブルに 1  つ以上のインデックスがある場合は、値として  1  を指定する必要があります。

        TableOptio  ns   tableOptio  ns  =  new   TableOptio  ns ( timeToLive ,  maxVersion  s );

        ArrayList < IndexMeta >  indexMetas  =  new   ArrayList < IndexMeta >();
        IndexMeta   indexMeta0  =  new   IndexMeta ( INDEX0_NAM  E );
        indexMeta0 . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_1 );
        indexMetas . add ( indexMeta0 );
        IndexMeta   indexMeta1  =  new   IndexMeta ( INDEX1_NAM  E );
        indexMeta1 . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_3 );
        indexMeta1 . addPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_2 );
        indexMetas . add ( indexMeta1 );
        IndexMeta   indexMeta2  =  new   IndexMeta ( INDEX2_NAM  E );
        indexMeta2 . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_3 );
        indexMeta2 . addPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_2 );
        indexMeta2 . addDefined  Column ( DEFINED_CO  L_NAME_2 );
        indexMetas . add ( indexMeta2 );

        CreateTabl  eRequest   request  =  new   CreateTabl  eRequest ( tableMeta ,  tableOptio  ns ,  indexMetas );

        client . createTabl  e ( request );
    }
```

4 HBase

4.1 Table Store HBase Client

SDK と RESTful API に加えて、Table Store HBase Client は、オープンソースHBase API 上に構築された Java アプリケーションを介して Table Store にアクセスするために使用できます。 Table Store バージョン 4.2.x 以降用の Java SDK に基づいて、Table Store HBase Client は HBase バージョン 1.x.x 以降用のオープンソース API をサポートします。

Table Store HBase Client は、次の 3 つのチャンネルのいずれかから入手できます。

- GitHub tablestore-hbase-client プロジェクト
- 圧縮パッケージ
- Maven

```
< dependenci  es >
        < dependency >
            < groupId > com . aliyun . openservic  es </ groupId >
            < artifactId > tablestore - hbase - client </ artifactId
>
            < version > 1 . 2 . 0 </ version >
        </ dependency >
    </ dependenci  es >
```

Table Store は、完全に管理された NoSQL データベースサービスです。 TableStore HBase Client を使用している場合、HBase Server を単に無視することができます。 代わりに、クライアントによって公開された API を使用してテーブルまたはデータ操作を実行するだけで済みます。

自作 HBase サービスと比較して、Table Store には以下の利点があります。

項目	Table Store	自作 HBase クラスター
コスト	請求は実際のデータ量に基づいています。 高性能と大容量のインスタンスを提供することで、Table Store はすべてのシナリオに合わせて調整できます。	トラフィックのピークに基づいてリソースを割り当てます。 オフピーク時にはリソースがアイドル状態のままになるため、運用と保守のコストが高くなります。

項目	Table Store	自作 HBase クラスター
セキュリティ	Alibaba Cloud RAM を統合し、複数の認証および承認メカニズム、VPC およびプライマリ / RAM ユーザーアカウント管理をサポートします。 認可の細分性は、テーブルレベルと API レベルの両方で定義できます。	追加のセキュリティメカニズムが必要です。
信頼性	自動冗長データバックアップとフェイルオーバーをサポートします。 データの可用性は 99.9% 以上で、データの信頼性は 99.99999999% です。	クラスタの信頼性に依存します。
スケーラビリティ	Table Store の Server Load Balancer は、単一テーブルからの PB レベルのデータ転送をサポートしています。 数百万バイトのデータが同時に格納されている場合でも、手動でサイズ変更する必要はありません。	クラスタが使用容量に達すると、オンラインサービスに大きな影響を与える可能性があるため、複雑なオンライン/オフラインプロセスが必要になります。

4.2 Table Store HBase Client でサポートされている機能

Table Store と HBase で異なる API サポート

Table Store と HBase は、データモデルの点では似ていますが、異なるAPI を持っています。以下のセクションでは、Table Store HBase Client API と HBase API の違いについて詳しく説明します。

Table Store HBase Client API でサポートされている機能:

- CreateTable

  Table Store では、すべてのデータが同じ ColumnFamily にあると見なすことができるため、ColumnFamily をサポートしません。 これは、Table Store の TTL および最大バー

ジョンがテーブルレベルであることを意味します。 したがって、Table Store は次の機能いくつかをサポートしています。

機能	サポートされているかどうか
family max version	テーブルレベルの最大バージョンがサポートされています。 デフォルト値：1
family min version	サポートされていない
family ttl	テーブルレベルのTTLがサポートされている
is/set Readonly	RAM のサブアカウントでサポートされている
Pre-partitioning	サポートされていない
blockcach	サポートされていない
blocksize	サポートされていない
BloomFilter	サポートされていない
column max version	サポートされていない
cell ttl	サポートされていない
Control parameter	サポートされていない

- Put

機能	サポートされているかどうか
一度に複数列のデータを書き込む	サポートされている
タイムスタンプを指定する	サポートされている
タイムスタンプが指定されていない場合、デフォルトでシステム時刻を使用	サポートされている
単一行 ACL	サポートされていない
ttl	サポートされていない
セルの可視性	サポートされていない
タグ	サポートされていない

- Get

  Table Store は高いデータ一貫性を保証します。 データが API に書き込まれた後に HTTP 200 ステータスコード (OK) が返された場合、データはすべてのコピーに永続的に書き込まれ、Get によってすぐに読み取ることができます。

機能	サポートされているかどうか
データ行を読み取る	サポートされている
ColumnFamily のすべての列を読み取る	サポートされている
指定された列からデータを読み取る	サポートされている
指定されたタイムスタンプでデータを読み込む	サポートされている
指定した数のバージョンのデータを読み込む	サポートされている
タイムレンジ	サポートされている
ColumnfamilyTimeRange	サポートされていない
RowOffsetPerColumnFamily	サポートされている
MaxResultsPerColumnFamily	サポートされていない
checkExistenceOnly	サポートされていない
nearestRowBefore	サポートされている
属性	サポートされていない
cacheblock: true	サポートされている
cacheblock: false	サポートされていない
IsolationLevel: READ_COMMITTED	サポートされている
IsolationLevel:READ_UNCOMMITTED	サポートされていない
IsolationLevel:STRONG	サポートされている
IsolationLevel:TIMELINE	サポートされていない

- Scan

  Table Store は高いデータ一貫性を保証します。 データが API に書き込まれた後に HTTP 200 ステータスコード (OK) が返された場合、そのデータはすべてのコピーに恒久的に書き込まれます。これは Scan ですぐに読み取ることができます。

機能	サポートされているかどうか
指定された開始と停止に基づいてスキャン範囲を決定する	サポートされている

機能	サポートされているかどうか
スキャン範囲が指定されていない場合、データをグローバルにスキャンする	サポートされている
プレフィックスフィルタ	サポートされている
Get と同じロジックを使用してデータを読み取る	サポートされている
逆順にデータを読み取る	サポートされている
キャッシュ	サポートされている
バッチ	サポートされていない
maxResultSize、返されるデータボリュームの最大サイズを示す	サポートされていない
small	サポートされていない
バッチ	サポートされていない
cacheblock:true	サポートされている
cacheblock:false	サポートされていない
IsolationLevel:READ_COMMITTED	サポートされている
IsolationLevel:READ_UNCOMMITTED	サポートされていない
IsolationLevel:STRONG	サポートされている
IsolationLevel:TIMELINE	サポートされていない
allowPartialResults	サポートされていない

- Batch

機能	サポートされているかどうか
Get	サポートされている
Put	サポートされている
Delete	サポートされている
batchCallback	サポートされていない

- Delete

機能	サポートされているかどうか
行を削除する	サポートされている
指定された列のすべてのバージョンを削除する	サポートされている

機能	サポートされているかどうか
指定された列の指定されたバージョンを削除する	サポートされている
指定された ColumnFamily を削除する	サポートされていない
タイムスタンプが指定されている場合、deleteColumn はそのタイムスタンプと等しいバージョンを削除する	サポートされている
タイムスタンプが指定されている場合、deleteFamily および deleteColumn はタイムスタンプより前のバージョンまたは等しいタイムスタンプを削除する	サポートされていない
タイムスタンプが指定されていない場合、deleteColumn は最新バージョンを削除する	サポートされていない
タイムスタンプが指定されていない場合、deleteFamily と deleteColumn は現在のシステム時刻のバージョンを削除する	サポートされていない
addDeleteMarker	サポートされていない

- checkAndXXX

機能	サポートされているかどうか
CheckAndPut	サポートされている
checkAndMutate	サポートされている
CheckAndDelete	サポートされている
列の値が条件を満たしているかどうかを確認する 存在する場合、checkAndXXXは列を削除する	サポートされている
値が指定されていない場合はデフォルト値を使用する	サポートされている
行 A をチェックして行 B を実行する	サポートされていない

- Exist

機能	サポートされているかどうか
1つ以上の行が存在するかどうかを確認し、内容を返さない	サポートされている

- Filter

機能	サポートされているかどうか
ColumnPaginationFilter	columnOffset と count はサポートされていない
SingleColumnValueFilter	サポートされている: LongComparator、BinaryComparator および ByteArrayComparable サポートされていない: RegexStringComparator、SubstringComparator および BitComparator

Table Store HBase Client API でサポートされていない機能

- 名前空間

  Table Store はインスタンスを使用してデータテーブルを管理します。 インスタンスは、Table Store の最小課金単位です。 インスタンスは Table Store コンソールで管理できます。 したがって、次の機能はサポートされていません。

  - createNamespace(NamespaceDescriptor descriptor)
  - deleteNamespace(String name)
  - getNamespaceDescriptor(String name)
  - listNamespaceDescriptors()
  - listTableDescriptorsByNamespace(String name)
  - listTableNamesByNamespace(String name)
  - modifyNamespace(NamespaceDescriptor descriptor)

- リージョン管理

  データパーティションは、Table Store のデータ保存と管理の基本単位です。 Table Store は、データボリュームとアクセス条件に基づいてデータパーティションを自動的に分割または結合します。 したがって、Table Store は HBase のリージョン管理に関連する機能をサポートしていません。

- スナップショット

  Table Store はスナップショット、またはスナップショットの関連機能をサポートしていません。

・テーブル管理

Table Store は、テーブル内のデータパーティションを自動的に分割、結合、および圧縮します。 したがって、次の機能はサポートされていません。

- getTableDescriptor(TableName tableName)
- compact(TableName tableName)
- compact(TableName tableName, byte[] columnFamily)
- flush(TableName tableName)
- getCompactionState(TableName tableName)
- majorCompact(TableName tableName)
- majorCompact(TableName tableName, byte[] columnFamily)
- modifyTable(TableName tableName, HTableDescriptor htd)
- split(TableName tableName)
- split(TableName tableName, byte[] splitPoint)

・コプロセッサ

Table Store はコプロセッサをサポートしていません。 したがって、次の機能はサポートされていません。

- coprocessorService()
- coprocessorService(ServerName serverName)
- getMasterCoprocessors()

・分散プロシージャー

テーブルストアは分散プロシージャーをサポートしません。 したがって、次の機能はサポートされていません。

- execProcedure(String signature, String instance, Map props)
- execProcedureWithRet(String signature, String instance, Map props)
- isProcedureFinished(String signature, String instance, Map props)

・インクリメントと追加

Table Store は、アトミック増加/減少またはアトミック追加をサポートしません。

4.3 Table Store と HBase の違い

ここでは、Table Store HBase Client の機能を紹介し、HBase と比較したときに制限されサポートされる機能について説明します。 機能は以下のとおりです。

テーブル

Table Store は単一の ColumnFamilies のみをサポートします。つまり、複数の ColumnFamilies はサポートしません。

行とセル

- Table Store は ACL 設定をサポートしません。
- Table Store はセルの表示設定をサポートしません。
- Table Store はタグ設定をサポートしません。

GET

Table Store は単一の ColumnFamilies のみをサポートします。 そのため、ColumnFamily 関連の API はサポートしていません。

- setColumnFamilyTimeRange(byte[] cf, long minStamp, long maxStamp)
- setMaxResultsPerColumnFamily(int limit)
- setRowOffsetPerColumnFamily(int offset)

SCAN

GET と同様に、Table Store は ColumnFamily 関連の API をサポートしていないため、次のような部分最適化 API の設定には使用できません。

- setBatch(int batch)
- setMaxResultSize(long maxResultSize)
- setAllowPartialResults(boolean allowPartialResults)
- setLoadColumnFamiliesOnDemand(boolean value)
- setSmall(boolean small)

バッチ

Table Store は BatchCallback をサポートしません。

変更と削除

- Table Store は、指定された ColumnFamily の削除をサポートしません。
- Table Store は最新のタイムスタンプを持つバージョンの削除をサポートしていません。
- Table Store は、指定されたタイムスタンプより前のすべてのバージョンの削除をサポートしません。

インクリメントと追加

Table Store は Increment 機能や Append 機能をサポートしていません。

フィルター

- Table Store は ColumnPaginationFilter をサポートしています。
- Table Store は FilterListをサポートしています。
- Table Store は SingleColumnValueFilter を部分的にサポートし、 BinaryComparator のみをサポートします。
- Table Store は他のフィルタをサポートしません。

最適化

HBase API の中には、アクセスとストレージの最適化を含むものがあります。 これらの API は現在開かれていません。

- blockcache: デフォルト値は "true" です。これは変更できません。
- blocksize: デフォルト値は "64 KB" です。これは変更できません。
- IsolationLevel: デフォルト値は "READ_COMMITTED" です。これは変更できません。
- 整合性: デフォルト値は "STRONG" です。これは変更できません。

管理者

`org.apache.hadoop.hbase.client.Admin` HBaseのAPIは管理と制御に使用され、そのほとんどは Table Store には必要ありません。

Table Store はクラウドサービスであるため、運用、保守、管理、制御などの作業は自動的に実行され、これらの作業を気にかける必要がありません。 Table Store は現在、いくつかの API をサポートしていません。

- CreateTable

  Table Store は単一の ColumnFamilies のみをサポートします。 したがって、テーブルを作成するときに作成できる ColumnFamily は1つだけです。 ColumnFamily は、MaxVersions パラメーターと TimeToLive パラメーターをサポートしています。

- メンテナンス作業

  Table Store では、タスクメンテナンスに関連する以下の API が自動的に処理されます。

  - abort(String why, Throwable e)
  - balancer()
  - enableCatalogJanitor(boolean enable)
  - getMasterInfoPort()
  - isCatalogJanitorEnabled()
  - rollWALWriter(ServerName serverName) -runCatalogScan()
  - setBalancerRunning(boolean on, boolean synchronous)
  - updateConfiguration(ServerName serverName)
  - updateConfiguration()
  - stopMaster()
  - shutdown()
- 名前空間

  Table Store では、インスタンス名は HBase の名前空間と似ています。 したがって、次のような名前空間関連の API はサポートされていません。

  - createNamespace(NamespaceDescriptor descriptor)
  - modifyNamespace(NamespaceDescriptor descriptor)
  - getNamespaceDescriptor(String name)
  - listNamespaceDescriptors()
  - listTableDescriptorsByNamespace(String name)
  - listTableNamesByNamespace(String name)
  - deleteNamespace(String name)

・ リージョン

Table Store は自動的に地域関連の操作を実行します。 したがって、次の API はサポートされていません。

- assign(byte[] regionName)
- closeRegion(byte[] regionname, String serverName)
- closeRegion(ServerName sn, HRegionInfo hri)
- closeRegion(String regionname, String serverName)
- closeRegionWithEncodedRegionName(String encodedRegionName, String serverName)
- compactRegion(byte[] regionName)
- compactRegion(byte[] regionName, byte[] columnFamily)
- compactRegionServer(ServerName sn, boolean major)
- flushRegion(byte[] regionName)
- getAlterStatus(byte[] tableName)
- getAlterStatus(TableName tableName)
- getCompactionStateForRegion(byte[] regionName)
- getOnlineRegions(ServerName sn)
- majorCompactRegion(byte[] regionName)
- majorCompactRegion(byte[] regionName, byte[] columnFamily)
- mergeRegions(byte[] encodedNameOfRegionA, byte[] encodedNameOfRegionB, boolean forcible)
- move(byte[] encodedRegionName, byte[] destServerName)
- offline(byte[] regionName)
- splitRegion(byte[] regionName)
- splitRegion(byte[] regionName, byte[] splitPoint)
- stopRegionServer(String hostnamePort)
- unassign(byte[] regionName, boolean force)

スナップショット

Table Store はスナップショット関連の API をサポートしません。

レプリケーション

Table Store はレプリケーション関連の API をサポートしません。

コプロセッサ

Table Store は、コプロセッサ関連の API をサポートしません。

分散プロシージャー

Table Store は、分散プロシージャー関連の API をサポートしません。

テーブル管理

テーブルストアは自動的にテーブル関連の操作を実行しますので、これを気にする必要はありません。 したがって、Table Store は次の API をサポートしません。

- compact(TableName tableName)
- compact(TableName tableName, byte[] columnFamily)
- flush(TableName tableName)
- getCompactionState(TableName tableName)
- majorCompact(TableName tableName)
- majorCompact(TableName tableName, byte[] columnFamily)
- modifyTable(TableName tableName, HTableDescriptor htd)
- split(TableName tableName)
- split(TableName tableName, byte[] splitPoint)

制限事項

Table Store はクラウドサービスであるため、最適な全体的なパフォーマンスを保証するために、一部のパラメーターは制限されており、再設定できません。 制限の詳細については、「制限」をご参照ください。

4.4 HBase から Table Store への移行

以下は HBase を Table Store に移行する方法を説明しています。

依存関係

Table Store HBase Client v1.2.0 は、HBase Client v1.2.0 および Table Store Java SDK v4.2.1 に依存します。 `pom . xml` の設定は以下のとおりです。

```
< dependenci  es >
        < dependency >
            < groupId > com . aliyun . openservic  es </ groupId >
            < artifactId > tablestore - hbase - client </ artifactId >
            < version > 1 . 2 . 0 </ version >
        </ dependency >
```

```
    </ dependenci  es >
```

別の HBase Client または Table Store の Java SDK バージョンを使用する場合は、除外タグを使用する必要があります。 次の例では、HBase Client v1.2.1 と Table Store Java SDK v4.2.0 が使用されています。

```
  < dependenci  es >
       < dependency >
           < groupId > com . aliyun . openservic  es </ groupId >
           < artifactId > tablestore - hbase - client </ artifactId >
           < version > 1 . 2 . 0 </ version >
           < exclusions >
               < exclusion >
                   < groupId > com . aliyun . openservic  es </
groupId >
                   < artifactId > tablestore </ artifactId >
               </ exclusion >
               < exclusion >
                   < groupId > org . apache . hbase </ groupId >
                   < artifactId > hbase - client </ artifactId >
               </ exclusion >
           </ exclusions >
       </ dependency >
       < dependency >
           < groupId > org . apache . hbase </ groupId >
           < artifactId > hbase - client </ artifactId >
           < version > 1 . 2 . 1 </ version >
       </ dependency >
       < dependency >
           < groupId > com . aliyun . openservic  es </ groupId >
           < artifactId > tablestore </ artifactId >
           < classifier > jar - with - dependenci  es </ classifier >
           < version > 4 . 2 . 0 </ version >
       </ dependency >
   </ dependenci  es >
```

Table Store HBase Client v1.2.x は HBase Client v1.2.x とのみ互換性があります。これは、API の変更が HBase Client v1.2.x 以前に存在するためです。

HBase Client バージョン v1.1.x を使用したい場合は、Table Store HBase Client バージョン v1.1.x を使用してください。

HBase クライアントバージョン v0.x.x を使用する場合は、「以前のバージョンの HBase の移行」をご参照ください。

ファイルを設定する

HBase Client から Table Store HBase Client にデータを移行するには、設定ファイル内の次の 2 つの項目を変更します。

- HBase 接続タイプ

  Connection を TableStoreConnection に設定します。

```
    < property >
```

```
        < name > hbase . client . connection . impl </ name >
        < value > com . alicloud . tablestore . hbase . Tablestore
Connection </ value >
   </ property >
```

- Table Store の設定項目

  Table Store はクラウドサービスであり、厳密な権限管理を提供します。 Table Store は厳格な権限管理を提供します。 Table Store にアクセスするには、AccessKey などのアクセス情報を設定する必要があります。

  - Table Store にアクセスする前に、次の 4 つの項目を設定する必要があります。

```
 < property >
   < name > tablestore . client . endpoint </ name >
   < value ></ value >
  </ property >
  < property >
   < name > tablestore . client . instancena  me </ name >
   < value ></ value >
  </ property >
  < property >
   < name > tablestore . client . accesskeyi  d </ name >
   < value ></ value >
  </ property >
  < property >
   < name > tablestore . client . accesskeys  ecret </ name >
   < value ></ value >
  </ property >
```

  - 設定できるオプション項目は以下のとおりです。

```
  < property >
   < name > hbase . client . tablestore . family </ name >
   < value > f1 </ value >
  </ property >
  < property >
   < name > hbase . client . tablestore . family .$ tablename </
name >
   < value > f2 </ value >
  </ property >
  < property >
   < name > tablestore . client . max . connection  s </ name >
   < value > 300 </ value >
  </ property >
  < property >
   < name > tablestore . client . socket . timeout </ name >
   < value > 15000 </ value >
  </ property >
  < property >
   < name > tablestore . client . connection . timeout </ name >
   < value > 15000 </ value >
  </ property >
  < property >
   < name > tablestore . client . operation . timeout </ name >
   < value > 2147483647 </ value >
  </ property >
  < property >
   < name > tablestore . client . retries </ name >
   < value > 3 </ value >
```

```
  </ property >
```

- hbase.client.tablestore.family および hbase.client.tablestore.family.$ tablename
  - Table Store は単一の ColumnFamilies のみをサポートします。 HBase API を使用するときは、ファミリの内容を入力する必要があります。

    `hbase . client . tablestore . family` はグローバル設定を示します。 `hbase . client . tablestore . family .$ tablenae` は、単一テーブルの設定を示します。
  - ルール: 名前が T のテーブルの場合、最初に `hbase . client . tablestore . family . T` を検索してください。 ファミリが存在しない場合は、 `hbase . client . tablestore . family` を検索してください。 そのファミリが存在しない場合は、既定値 f を使用します。
- tablestore.client.max.connections

  最大接続数です。 デフォルト値は 300 です。
- tablestore.client.socket.timeout

  ソケットタイムアウト時間です。 デフォルト値は、15 秒です。
- tablestore.client.connection.timeout

  接続タイムアウト時間です。 デフォルト値は、15 秒です。
- tablestore.client.operation.timeout

  API タイムアウト時間です。 デフォルト値は Integer.MAX_VALUE です。これは、API がタイムアウトしないことを示します。
- tablestore.client.retries

  要求が失敗したときの再試行回数です。 デフォルト値は 3 です。

4.5 以前のバージョンの HBase を移行する

Table Store HBase Client は HBase Client 1.0.0 以降のバージョンの API をサポートします。

以前のバージョンと比較して、HBase Client 1.0.0 には、以前のバージョンの HBase Client と互換性のない大きな変更があります。

バージョン 0.x.x (つまり、1.0.0 より前のバージョン) の HBase Client を使用している場合、ここでは HBase Client のバージョンを Table Store と統合する方法について説明します。

接続 API

HBase 1.0.0 以降のバージョンは HConnection API をキャンセルし、代わりに `org . apache . hadoop . hbase . client . Connection Factory` シリーズを使用します。 Connection API を提供し、ConnectionManager と HConnectionManager を ConnectionFactoryに置き換えです。

接続 API の作成は比較的コストがかかりますが、接続 API はスレッドの安全性を保証します。 Connection API を使用する場合、プログラム内に生成できる Connection オブジェクトは1つだけです。 複数のスレッドがこのオブジェクトを共有できます。

また、Connection のライフサイクルを管理し、使用後に閉じる必要があります。

最新のコードは次のとおりです。

```
Connection connection = Connection Factory . createConn ection
( config );
// ...
connection . close ();
```

TableName シリーズ

HBase バージョン 1.0.0 以前では、テーブルを作成するときに String 型の名前を使用できます。 それ以降のバージョンの HBase では、 `org . apache . hadoop . hbase . TableName` を使用できます。

最新のコードは次のとおりです。

```
String tableName = " MyTable ";
// or byte [] tableName = Bytes . toBytes (" MyTable ");
TableName tableNameO bj = TableName . valueOf ( tableName );
```

Table、BufferedMutator および RegionLocator の各 API

HBase Client v1.0.0 から、HTable API は Table、BufferedMutator および RegionLocator の各 API に置き換えられました。

- `org . apache . hadoop . hbase . client . Table` は、単一のテーブルに対する読み取り、書き込み、その他の要求を操作するために使用されます。
- `org . apache . hadoop . hbase . client . BufferedMu tator` は、非同期バッチ書き込みに使用されます。 このAPIは、以前のバージョンの HTableInterface API の `setAutoFlu sh ( boolean )` に対応しています。
- `org . apache . hadoop . hbase . client . RegionLoca tor` は、テーブルパーティション情報を示します。

Table、BufferedMutator および RegionLocator の各 API は、スレッドの安全性を保証するものではありません。 ただし、これらは軽量であり、各スレッドのオブジェクトを作成するために使用できます。

管理 API

HBase Client v1.0.0 から、HBaseAdmin API は `org.apache.hadoop.hbase.client.Admin` に置き換えられました。 Table Store はクラウドサービスであり、ほとんどの運用および保守 API は自動的に処理されるため、ほとんどの管理 API はサポートされていません。 詳細については、「Table Store と HBase の違い」をご参照ください。

Connection インスタンスを使用して管理者インスタンスを作成します。

```
Admin admin = connection.getAdmin();
```

4.6 Hello World

このトピックでは、Table Store HBase Client を使用して単純な Hello World プログラムを実装する方法について説明します。次の操作が含まれます。

- プロジェクトの依存関係を設定する
- Table Store に接続する
- テーブルを作成する
- データを書き込む
- データを読み込む
- データをスキャンする
- テーブルを削除する

コード位置

このサンプルプログラムは HBase API を使用して Table Store にアクセスします。 完全なサンプルプログラムは、 Githubのaliyun-tablestore-hbase-client プロジェクトにあります。 ディレクトリは、 "src/test/java/samples/HelloWorld.java" です。

HBase API を使用する

- プロジェクトの依存関係を設定する

  次のように Maven の依存関係を設定します。

```
<dependencies>
    <dependency>
        <groupId>com.aliyun.openservices</groupId>
        <artifactId>tablestore-hbase-client</artifactId>
</
```

```
            < version > 1 . 2 . 0 </ version >
        </ dependency >
    </ dependenci  es >
```

詳細設定の詳細については、「HBase から Table Store への移行」をご参照ください。

- ファイルを設定する

  以下の設定項目を hbase-site.xml に追加します。

```
< configurat  ion >
    < property >
        < name > hbase . client . connection . impl </ name >
        < value > com . alicloud . tablestore . hbase . Tablestore
 Connection </ value >
    </ property >
    < property >
        < name > tablestore . client . endpoint </ name >
        < value > endpoint </ value >
    </ property >
    < property >
        < name > tablestore . client . instancena  me </ name >
        < value > instance_n  ame </ value >
    </ property >
    < property >
        < name > tablestore . client . accesskeyi  d </ name >
        < value > access_key  _id </ value >
    </ property >
    < property >
        < name > tablestore . client . accesskeys  ecret </ name >
        < value > access_key  _secret </ value >
    </ property >
    < property >
        < name > hbase . client . tablestore . family </ name >
        < value > f1 </ value >
    </ property >
    < property >
        < name > hbase . client . tablestore . table </ name >
        < value > ots_adapto  r </ value >
    </ property >
</ configurat  ion >
```

詳細設定の詳細については、「HBase から Table Store への移行」をご参照ください。

- テーブルストアに接続

  Table Store に接続するための TableStoreConnection オブジェクトを作成します。

```
   Configurat  ion   config  =  HBaseConfi  guration . create ();

  //  Tablestore   Connection を作成する
   Connection   connection  =  Connection  Factory . createConn
 ection ( config );

  // 管理者は作成、管理、削除に使用される
```

```
        Admin admin = connection . getAdmin ();
```

- テーブルを作成する

  指定されたテーブル名を使ってテーブルを作成します。MaxVersions と TimeToLive にはデフォルトのテーブル名を使用してください。

```
  // ColumnFami ly を 1 つだけ含む HTableDesc riptor を作成する
   HTableDesc riptor descriptor = new HTableDesc riptor (
TableName . valueOf ( TABLE_NAME ));

  // ColumnFami ly を作成する。 Max Versions および TimeToLive
  にはデフォルトの ColumnFami ly 名を使用してください。 Max Versions
  のデフォルトの ColumnFami ly 名は 1 、 TimeToLive のデフォルトの
ColumnFami ly 名は Integer . INF_MAX です。
  descriptor . addFamily ( new HColumnDes criptor ( COLUMN_FAM
ILY_NAME ));

  // 管理者の createTabl e API を使用してテーブルを作成する
  System . out . println (" Create table " + descriptor .
getNameAsS tring ());
  admin . createTabl e ( descriptor );
```

- データを書き込む

  1 行のデータを Table Store に書き込みます。

```
  // 単一のテーブルに対する読み取り、書き込み、更新、削除、およびその他の操作用
の Table Store テーブルを作成する
   Table table = connection . getTable ( TableName . valueOf (
TABLE_NAME ));

  // プライマリキー row_1 で Put オブジェクトを作成する
  System . out . println (" Write one row to the table
");
        Put put = new Put ( ROW_KEY );

  // 列を追加する Table Store は単一の ColumnFami lies のみをサ
ポートします。 ColumnFami ly の名前は " hbase - site . xml " で設定さ
れています。 ColumnFami ly の名前が設定されていない場合、デフォルトの名前は
 " f " です。 この場合、データの書き込み時に COLUMN_FAM ILY_NAME の値が
null 値になることがあります。
  put . addColumn ( COLUMN_FAM ILY_NAME , COLUMN_NAM E ,
COLUMN_VAL UE );

  // テーブルに対して put を実行し、 HBase API を使用して Table
Store にデータ行を書き込む
        table . put ( put );
```

- データを読み込む

  指定された行のデータを読み込みます。

```
  // プライマリキーが ROW_KEY である行を読み取るための Get オブジェクトを
作成する
   Result getResult = table . get ( new Get ( ROW_KEY ));
   Result result = table . get ( get );

  // 結果を出力する
```

```
  String   value  =  Bytes . toString ( getResult . getValue (
COLUMN_FAM  ILY_NAME ,  COLUMN_NAM  E ));
  System . out . println (" Get   one   row   by   row   key ");
  System . out . printf ("\ t % s  = % s \ n ",  Bytes . toString (
ROW_KEY ),  value );
```

- スキャンデータ

  指定した範囲のデータを読み込みます。

```
テーブルのすべての行のデータをスキャンする
 System . out . println (" Scan   for   all   rows :");
 Scan   scan  =  new   Scan ();

 ResultScan  ner   scanner  =  table . getScanner ( scan );

// 結果を周期的に印刷する
 for  ( Result   row  :  scanner ) {
  byte []  valueBytes  =  row . getValue ( COLUMN_FAM  ILY_NAME ,
COLUMN_NAM  E );
  System . out . println ('\ t ' +  Bytes . toString ( valueBytes
));
}
```

- テーブルを削除する

  管理 API を使用してテーブルを削除します。

```
  print (" Delete   the   table ");
  admin . disableTab  le ( table . getName ());
  admin . deleteTabl  e ( table . getName ());
```

コードの完了

```
package   samples ;

import   org . apache . hadoop . conf . Configurat  ion ;
import   org . apache . hadoop . hbase . HBaseConfi  guration ;
import   org . apache . hadoop . hbase . HColumnDes  criptor ;
import   org . apache . hadoop . hbase . HTableDesc  riptor ;
import   org . apache . hadoop . hbase . TableName ;
import   org . apache . hadoop . hbase . client .*;
import   org . apache . hadoop . hbase . util . Bytes ;

import   java . io . IOExceptio  n ;

public   class   HelloWorld  {

    private   static   final   byte []  TABLE_NAME  =  Bytes .
toBytes (" HelloTable  store ");
    private   static   final   byte []  ROW_KEY  =  Bytes . toBytes
(" row_1 ");
    private   static   final   byte []  COLUMN_FAM  ILY_NAME  =
Bytes . toBytes (" f ");
    private   static   final   byte []  COLUMN_NAM  E  =  Bytes .
toBytes (" col_1 ");
    private   static   final   byte []  COLUMN_VAL  UE  =  Bytes .
toBytes (" col_value ");

    public   static   void   main ( String []  args ) {
        helloWorld ();
```

```
    }

    private   static   void   helloWorld () {

        try   {
            Configurat  ion   config  =  HBaseConfi  guration .
create ();
            Connection   connection  =  Connection  Factory .
createConn  ection ( config );
            Admin   admin  =  connection . getAdmin ();

            HTableDesc  riptor   descriptor  =  new   HTableDesc
riptor ( TableName . valueOf ( TABLE_NAME ));
            descriptor . addFamily ( new   HColumnDes  criptor (
COLUMN_FAM  ILY_NAME ));

            System . out . println (" Create   table  " +  descriptor
. getNameAsS  tring ());
            admin . createTabl  e ( descriptor );

            Table   table  =  connection . getTable ( TableName .
valueOf ( TABLE_NAME ));

            System . out . println (" Write   one   row   to   the
table ");
            Put   put  =  new   Put ( ROW_KEY );
            put . addColumn ( COLUMN_FAM  ILY_NAME ,  COLUMN_NAM  E
,  COLUMN_VAL  UE );
            table . put ( put );

            Result   getResult  =  table . get ( new   Get ( ROW_KEY
));
            String   value  =  Bytes . toString ( getResult .
getValue ( COLUMN_FAM  ILY_NAME ,  COLUMN_NAM  E ));
            System . out . println (" Get   a   one   row   by   row
  key ");
            System . out . printf ("\ t % s  = % s \ n ",  Bytes .
toString ( ROW_KEY ),  value );

            Scan   scan  =  new   Scan ();

            System . out . println (" Scan   for   all   rows :");
            ResultScan  ner   scanner  =  table . getScanner ( scan
);
            for  ( Result   row  :  scanner ) {
                byte []  valueBytes  =  row . getValue ( COLUMN_FAM
ILY_NAME ,  COLUMN_NAM  E );
                System . out . println ('\ t ' +  Bytes . toString (
valueBytes ));
           }

            System . out . println (" Delete   the   table ");
            admin . disableTab  le ( table . getName ());
            admin . deleteTabl  e ( table . getName ());

            table . close ();
            admin . close ();
            connection . close ();
       }  catch  ( IOExceptio  n   e ) {
            System . err . println (" Exception   while   running
HelloTable  store : " +  e . toString ());
            System . exit ( 1 );
       }
    }
```

```
}
```