ドキュメント データベース (DocDB) の使用法

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

ドキュメント データベース (DocDB) の使用法

InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

InterSystems IRIS ドキュメント データベース (DocDB) の概要

InterSystems IRIS® Data Platform の DocDB は、データベース・データの保存と取得のための機能です。従来の SQL のテーブルおよびフィールド (クラスおよびプロパティ) データの保存や取得との互換性はありますが、これとは別の機能です。また、Web ベースのデータ交換をサポートする JSON (JavaScript Object Notation) がベースになっています。InterSystems IRIS は、DocDB データの作成と照会のための SQL サポートを提供するだけでなく、REST や ObjectScript での DocDB データベースおよびアプリケーションの開発のためのサポートも提供します。

InterSystems IRIS ドキュメント データベースは、本質的に、スキーマレスなデータ構造です。つまり、各ドキュメントがそれぞれ独自の構造を持ちます。この構造は、同じデータベース内の他のドキュメントと異なる場合もあります。これには、事前定義されたデータ構造を必要とする SQL よりも優れたメリットがいくつか存在します。

ここでは、「ドキュメント」という用語が、業界全体で使用される特殊な専門用語として、動的なデータ保存構造の意味で使用されます。DocDB で使用される「ドキュメント」をテキスト・ドキュメントや文書と混同しないようにしてください。

1.1 機能およびメリット

InterSystems IRIS DocDB は、以下のような主要機能を備えています。

- アプリケーションの柔軟性 : ドキュメントは事前定義されたスキーマを必要としません。これにより、アプリケーションでデータ環境の設定を速やかに行い、データ構造の変化に容易に順応することができます。これにより、データの迅速な取得が可能になります。ドキュメント・データベースは、データの構造を定義することなく、即座にデータの取得を開始できます。これは、Web ベースおよびソーシャル・メディアのデータ・ソースでよく見られる予測不能なデータ・フィードに最適です。データの本文の取得時、そのデータ内の構造が役立つものであることが明らかになった場合、ドキュメントのデータ構造をさらに発展させることが可能です。既存の取得済みデータは、より高度に構造化されたこのデータ表現と共存できます。ドキュメントごとにデータ構造を決定し、これを適切に処理するかどうかはご使用のアプリケーションによります。これを行う方法の 1 つとして、ドキュメント構造のバージョンを表す key:value ペアの構築が挙げられます。したがって、ある JSON 構造から別の JSON 構造へのデータの変換は、データの取得やアクセスを妨げることなく段階的に行える場合もあれば、まったく行えない場合もあります。
- スパース・データにおける効率性 : ドキュメントは、スパース・データの保存における効率性が非常に優れています。これは、特定のキーを持つ属性がコレクション内の一部のドキュメントに出現し、他のドキュメントには出現しないようにすることができるためです。ドキュメントはそれぞれ一組の定義済みキーを持つことができるため、同じコレクション内の他のドキュメントがまったく異なる一組の定義済みキーを持つことも可能です。それに対して、SQL では、すべてのレコードにそれぞれのキーが含まれている必要があります。スパース・データでは、多くのレコードが NULL 値のキーを持ちます。例えば、SQL の患者医療記録では、さまざまな診断、症状、および検査に関するフィールドが

用意されていますが、大半の患者について、これらのフィールドのほとんどが NULL になります。使用されないこれらのフィールドには、すべてスペースが割り当てられます。DocDB の患者医療記録では、実際のデータが含まれるそれらのキーのみが表示されます。

- 階層データの保存：DocDB は、階層構造データの保存における効率性が非常に優れています。key:value ペアでは、データをそのデータ内で無制限に入れ子にすることができます。つまり、階層データを正規化されていない状態で保存できます。SQL リレーショナル・モデルでは、階層データは、複数のテーブルを使用して正規化された状態で保存されます。
- ダイナミック・データ型：キーは、定義されたデータ型を持ちません。キーに割り当てられた値は、関連付けられたデータ型を持ちます。したがって、各ドキュメントの key:value ペアはそれぞれ 1 つのデータ型を持つことができるため、他のドキュメント内の同じキーの key:value ペアが異なるデータ型を持つことも可能です。データ型が固定されていないため、異なるデータ型を持つ新しい値を割り当てることで、実行時にドキュメント内の key:value ペアのデータ型を変更できます。

DocDB のこれらの機能は、アプリケーションの開発において重要な意味を持ちます。従来の SQL 環境では、アプリケーションの開発で用いられるデータ構造は、データベースの設計によって確立されます。DocDB では、データ構造は、主にアプリケーション自体で提供されます。

1.2 DocDB のコンポーネント

DocDB のパッケージ名は **%DocDB** です。これには、以下のクラスが含まれています。

- **%DocDB.Database**: ドキュメントの管理に使用される ObjectScript 永続クラスです。データベースは、**%DocDB.Document** を拡張する永続クラスによって実装される一連のドキュメントです。このクラスのメソッドを使用することで、データベースの作成、既存データベースの取得、またはデータベースの削除を行ったり、データベース内でドキュメントの挿入、ドキュメントの取得、またはドキュメントの削除を行ったりすることができます。
- **%DocDB.Document**: ドキュメント・データの保存に使用される構造です。ドキュメント ID、最終変更日、およびドキュメントの内容で構成されます。ドキュメントの内容は、**%Doc** プロパティで保存されます。データは、JSON ダイナミック・オブジェクトまたは JSON 動的配列として保存されます。ドキュメントは、複数の key:value ペア (オブジェクト) または順序どおりに並べられた値のリスト (配列) で構成されます。
- **%DocDB.REST** は、ドキュメント・データベースにアクセスするための DocDB REST API を実装します。

関連クラスは、JSON 構造を含めるのに使用され、JSON 配列と JSON key:value オブジェクトのサブクラスが含まれる **%Library.DynamicAbstractObject** です。

1.2.1 データベースの作成

データベースは、抽象クラス **%DocDB.Document** を拡張する ObjectScript 永続クラスです。DocDB で使用するネームスペースごとにデータベースをインスタンス化する必要があります。ネームスペースごとに必要になるデータベースは 1 つのみです。通常、ネームスペース名と同じ名前が割り当てられます。

次の例は、クラス定義を使用してデータベースを作成する方法を示しています。

```
Class MyDBs.People Extends %DocDB.Document [ DdlAllowed ]
```

次の例は、パッケージ名の指定による、%CreateDatabase() メソッドを使用したデータベースの作成方法を示しています。

ObjectScript

```
SET personDB = ##class(%DocDB.Database).%CreateDatabase("MyDBs.People")
```

次の例は、ISC.DM のデフォルトのパッケージ名の取得による、%CreateDatabase() メソッドを使用したデータベースの作成方法を示しています。

ObjectScript

```
SET personDB = ##class(%DocDB.Database).%CreateDatabase("People")
```

%SYSTEM.DocDB クラスは、ドキュメント・データベースを管理するためのインタフェースを提供します。

ドキュメント・データベースの作成または取得、データベースへのドキュメントの入力、およびそれらのドキュメントからのデータの取得に使用される API メソッドの詳細については、“ドキュメントの管理”の章を参照してください。

1.3 JSON 構造

InterSystems IRIS ドキュメント データベースは、JSON ダイナミック・オブジェクトと JSON 動的配列をサポートします。これらの JSON 構造は、SET コマンドを使用して作成できます。

次の例は、JSON を使用して階層データを保存する方法を示しています。最初の SET は、入れ子になった JSON 構造の key:value ペアおよび配列を含む動的な抽象オブジェクトを作成します。その後、この例では、動的な抽象オブジェクトが JSON 文字列に変換され、その JSON 文字列がドキュメントとして既存のドキュメント・データベース内に挿入されます。

ObjectScript

```
SET dynAbObj = {
 "FullName":"John Smith",
 "FirstName":"John",
 "Address":{
            "street":"101 Main Street",
            "city":"Mapleville",
            "state":"NY",
            "postal code":10234
           },
 "PhoneNumber":
            [
             {"type":"home","number":"212-456-9876"},
             {"type":"cell","number":"401-123-4567"},
             {"type":"work","number":"212-444-5000"}
            ]
}
SET jstring = dynAbObj.%ToJSON() // dynamic abstract object to JSON string
DO personDB.%FromJSON(jstring)   // JSON string inserted into document database
```

この例では、FullName は単純な key:value ペアとして保存されます。Address は、key:value ペアで構成されるオブジェクトして保存されるサブ構造を持ちます。PhoneNumber は、配列として保存されるサブ構造を持ちます。

詳細は、″JSON の使用″の“ダイナミック・エンティティの作成と変更”を参照してください。

1.3.1 ドキュメント

ドキュメントは、作成したデータベース・クラスのインスタンスの %Doc プロパティで保存されます。これは、次の例で示されています。この例では、%Doc プロパティで JSON 配列が保存されます。

ObjectScript

```
SET jarry = ["Anne","Bradford","Charles","Deborah"]
SET myoref = ##class(MyDBs.DB1).%New()
SET myoref.%Doc = jarry
SET docoref = myoref.%Doc
WRITE "%Doc property oref: ",docoref,!
WRITE "%Doc Property value: ",docoref.%ToJSON()
```

既定では、**%Doc** のデータ型は **%Library.DynamicAbstractObject** です。これは、JSON オブジェクトまたは JSON 配列の保存に使用されるデータ型です。%CreateDatabase() メソッドで異なるデータ型を指定できます。

その他のデータベース・プロパティ：

- %DocumentId は、ドキュメントを識別する一意の整数を含む IDENTITY プロパティです。%DocumentId は 1 からカウントされます。ほとんどの場合、%DocumentId 値は、システムによって割り当てられます。%DocumentId は一意である必要があります。%DocumentId は、連続的に割り当てるとは限らず、割り当て順序内で欠番が生じる場合もあります。ドキュメント・データベースは、%DocumentId 値の IdKey インデックスも自動的に生成します。
- %LastModified は、ドキュメント・インスタンスの定義時の UTC タイムスタンプを記録します。

1.3.2 正規化されていないデータ構造

以下に示すのは、従来の SQL の正規化されたリレーショナル・データ構造の JSON の例です。これは、2 つのドキュメントで構成されます。これらのドキュメントは、2 つの異なるコレクションに含まれる場合もあります。

```
{
   "id":123,
   "Name":"John Smith",
   "DOB":"1990-11-23",
   "Address":555
}
{
   "id":555,
   "street":"101 Main Street",
   "city":"Mapleville",
   "state":"NY",
   "postal code":10234
 }
```

以下に示すのは、入れ子になったデータ構造を含むコレクション内の単一ドキュメントとして指定された、正規化されていない同一のデータです。

```
{
   "id":123,
   "Name":"John Smith",
   "DOB":"1990-11-23",
   "Address":{
              "street":"101 Main Street",
              "city":"Mapleville",
              "state":"NY",
              "postal code":10234
             }
 }
```

SQL では、最初のデータ構造から 2 つ目のデータ構造に変換する場合、テーブル・データの定義を変更してからデータを移行する必要があります。

DocDB では、固定スキーマが存在しないため、これらの 2 つのデータ構造は、同じデータの異なる表現として共存できます。アクセスするデータ構造をアプリケーション・コードで指定する必要があります。データは、新しいデータ構造に移行することもでき、古いデータ構造形式のままにしておくこともできます。後者の場合、新しいデータ構造を使用してデータにアクセスする際に毎回データが移行されます。

JSON データ構造の詳細は、このマニュアルの“柔軟なデータ構造”の章を参照してください。

1.3.3 データ型とデータ値

DocDB では、キーはデータ型を持ちません。しかしながら、DocDB にインポートされたデータ値は、関連付けられたデータ型を持つことができます。データ型は特定の値に関連付けられるため、値を他の値に置換した場合、そのレコードの key:value ペアのデータ型が変更される場合があります。

InterSystems IRIS DocDB には、予約語や特別な名前付け規約が一切存在しません。key:value ペアでは、任意の文字列をキーとして使用でき、任意の文字列または数値を値として使用できます。キー名は、"name":"name" のように値と同一にすることができます。キー名は、インデックス名と同一にすることができます。

以下のテーブルのように、InterSystems IRIS DocDB は、データ値を JSON 値として表します。

データ値	表現
文字列	文字列
数値	数値は、"キャノニック形式" で表されます。ただし、例外として、1 から -1 までの JSON 小数は先頭に整数ゼロを付けて表されます (例 : 0.007)。対応する InterSystems IRIS の数値は、先頭に整数ゼロを付けずに表されます (例 : .007)。
$DOUBLE の数値	"IEEE 倍精度 (64 ビット) 浮動小数点数" として表されます。
非表示文字	JSON では、以下の非表示文字のエスケープ・コード表現が用意されています。 $CHAR(8): "¥b" $CHAR(9): "¥t" $CHAR(10): "¥n" $CHAR(12): "¥f" $CHAR(13): "¥r" その他の非表示文字はすべて、エスケープされた 16 進数で表されます。例えば、$CHAR(11) は "¥u000b" として表されます。エスケープされた 16 進数 (Unicode) 表記を使用して表示可能文字を表すこともできます。例えば、ギリシャ文字の小文字のアルファは、"¥u03b1" として表すことができます。
その他のエスケープ文字	JSON は、2 つの表示可能文字 (二重引用符文字とバックスラッシュ文字 (円記号)) をエスケープします。 $CHAR(34): "¥"" $CHAR(92): "¥¥"

1.3.4 特殊な JSON 値

特殊な JSON 値は、JSON オブジェクトと JSON 配列内でのみ使用できます。これらの値は、対応する特殊な ObjectScript 値とは異なります。特殊な JSON 値は、引用符なしで指定されます (同じ値を引用符で囲んで指定すると、通常のデータ値として扱われます)。これらの値は大文字と小文字の任意の組み合わせで指定でき、すべて小文字として保存されます。

- JSON では、特殊な null 値を使用して、値が存在しないことを表します。実際の値が存在しない場合は、通常、ドキュメント・データベースに key:value ペアは含まれないため、null は特別な条件でのみ使用されます (予想値のプレースホルダなど)。この null の使用法は、以下の例を参照してください。

ObjectScript

```
SET jsonobj = {"name":"Fred","spouse":null}
WRITE jsonobj.%ToJSON()
```

- JSON では、特殊な `true` 値および `false` 値を使用してブーリアン値を表します。このブーリアン値の使用法は、以下の例を参照してください。

ObjectScript

```
SET jsonobj = {"name":"Fred","married":false}
WRITE jsonobj.%ToJSON()
```

ObjectScript では、0 と 1 を使用してブーリアン値を指定します (実際は、"true" は 1 に限らず、ゼロ以外の任意の数値で表すことができます)。これらの値は、JSON ドキュメント内でブーリアン値としてサポートされません。

いくつかの特殊な場合において、JSON では、構文をわかりやすくするために括弧を使用します。

- null、true、または false という名前でローカル変数を定義する場合、これを特殊な JSON 値ではなく、ローカル変数として扱うために、JSON 内で括弧を使用する必要があります。詳細は、以下の例を参照してください。

ObjectScript

```
SET true=1
SET jsonobj = {"bool":true,"notbool":(true)}
WRITE jsonobj.%ToJSON()
```

- 式内で ObjectScript の "後続関係演算子" (]) を使用する場合、これを JSON 配列の終端文字ではなく後続関係演算子として扱うために、JSON 内で括弧を使用する必要があります。次の例では、照合順で b が a の後に続くかどうかを式 `b]a` がテストし、ObjectScript のブーリアン値を返します。後続関係式は括弧で囲む必要があります。

ObjectScript

```
SET a="a",b="b"
SET jsonarray=[(b]a)]
WRITE jsonarray.%ToJSON()
```

2
ドキュメントの管理

InterSystems IRIS® Data Platform の DocDB は、ObjectScript からの DocDB の操作を可能にするクラス・メソッドを提供します。ObjectScript からの JSON メソッドの呼び出しの詳細は、ドキュメント ″JSON の使用″ を参照してください。

2.1 ドキュメント・データベースの作成または取得

現在のネームスペース内の新規のドキュメント・データベースを作成するには、%CreateDatabase() メソッドを呼び出します。

現在のネームスペース内の既存のドキュメント・データベースを取得するには、%GetDatabase() メソッドを呼び出します。

ドキュメント・データベースに現在のネームスペース内で一意の名前を割り当てます。名前は修飾 `"packagename.docdbname"`、未修飾のどちらでもかまいません。データベース名が未修飾の場合は、既定値として `ISC.DM` パッケージが設定されます。

次の例では、その名前のデータベースが現在のネームスペース内に存在する場合にデータベースを取得します。それ以外の場合は、データベースを作成します。

ObjectScript

```
IF $SYSTEM.DocDB.Exists("People")
  { SET db = ##class(%DocDB.Database).%GetDatabase("People")}
ELSE {SET db = ##class(%DocDB.Database).%CreateDatabase("People") }
```

次の例では、データベースを作成または取得し、その後、%GetDatabaseDefinition() を使用してデータベース定義情報を表示します。これは JSON 動的抽象オブジェクトとして格納されています。

ObjectScript

```
IF $SYSTEM.DocDB.Exists("People")
  { SET db = ##class(%DocDB.Database).%GetDatabase("People")}
ELSE {SET db = ##class(%DocDB.Database).%CreateDatabase("People") }
SET defn = db.%GetDatabaseDefinition()
WRITE defn.%ToJSON()
```

%SYSTEM.DocDB.GetAllDatabases() を使用すると、このネームスペースで定義されているすべてのデータベースの名前を含む JSON 配列を返すことができます。

2.2 プロパティの定義 : %CreateProperty()

key:value ペアでドキュメントを取得するには、%CreateProperty() を使用して、そのキーのプロパティを定義する必要があります。プロパティを定義すると、ドキュメントが挿入、変更、および削除されたときに、InterSystems IRIS が保持するそのキーのインデックスが自動的に作成されます。プロパティでは、キーのデータ型を指定する必要があります。プロパティは、一意の値 (1) のみの受け入れとして、または非一意 (0) として指定することができます。デフォルトは非一意です。

次の例では、データベースに 2 つのプロパティを割り当てます。その後、データベース定義情報を表示します。

ObjectScript

```
  IF $SYSTEM.DocDB.Exists("People")
    { SET db = ##class(%DocDB.Database).%GetDatabase("People")}
  ELSE {SET db = ##class(%DocDB.Database).%CreateDatabase("People") }
  DO db.%CreateProperty("firstName","%String","$.firstName",0) // creates non-unique property
  DO db.%CreateProperty("lastName","%String","$.lastName",1)   // create a unique property; an index
to support uniqueness is automatically created
  WRITE db.%GetDatabaseDefinition().%ToJSON()
```

2.3 ドキュメントの挿入または置換 : %SaveDocument()

ドキュメント ID またはデータ選択条件のいずれかを使用して、データベース内のドキュメントを挿入または置換することができます。

%SaveDocument() および %SaveDocumentByKey() メソッドは、新規ドキュメントを挿入するか既存のドキュメントを置換することで、ドキュメントを保存します。%SaveDocument() はドキュメント ID でドキュメントを指定し、%SaveDocumentByKey() はキー名とキー値でドキュメントを指定します。

ドキュメント ID を指定しない場合、%SaveDocument() は新しいドキュメントを挿入し、新しいドキュメント ID を生成します。ドキュメント ID を指定した場合は、既存のドキュメントをそのドキュメント ID に置き換えます。ドキュメント ID を指定して、かつそのドキュメントが存在しない場合は、ERROR＃5809 例外が生成されます。

ドキュメント・データは、1 つ以上の key:value ペアで構成されます。一意として定義されているキー・プロパティの重複値を指定すると、ERROR＃5808 例外が生成されます。

%SaveDocument() メソッドと %SaveDocumentByKey() メソッドは、データベース・ドキュメント・クラスのインスタンスの参照を返します。これは常に、%DocDB.Document のサブクラスです。このメソッドの戻り値のデータ型は %DocDB.Document です。

次の例では、3 つの新しいドキュメントを挿入し、それらにドキュメント ID を割り当てます。その後、ドキュメント ID 2 で識別されるドキュメントの内容全体を、指定された内容に置き換えます。

ObjectScript

```
  IF $SYSTEM.DocDB.Exists("People")
    { SET db = ##class(%DocDB.Database).%GetDatabase("People")}
  ELSE {SET db = ##class(%DocDB.Database).%CreateDatabase("People") }
    WRITE db.%Size(),!
  DO db.%CreateProperty("firstName","%String","$.firstName",0)
  SET val = db.%SaveDocument({"firstName":"Serena","lastName":"Williams"})
  SET val = db.%SaveDocument({"firstName":"Bill","lastName":"Faulkner"})
  SET val = db.%SaveDocument({"firstName":"Fred","lastName":"Astare"})
  WRITE "Contains ",db.%Size()," documents: ",db.%ToJSON()
  SET val = db.%SaveDocument({"firstName":"William","lastName":"Faulkner"},2)
  WRITE !,"Contains ",db.%Size()," documents: ",db.%ToJSON()
```

次の例では、%Id() メソッドを各 %SaveDocument() に連鎖し、挿入または置換された各ドキュメントのドキュメント ID を返します。

ObjectScript

```
IF $SYSTEM.DocDB.Exists("People")
  { SET db = ##class(%DocDB.Database).%GetDatabase("People")}
ELSE {SET db = ##class(%DocDB.Database).%CreateDatabase("People") }
DO db.%CreateProperty("firstName","%String","$.firstName",0)
WRITE db.%SaveDocument({"firstName":"Serena","lastName":"Williams"}).%Id(),!
WRITE db.%SaveDocument({"firstName":"Bill","lastName":"Faulkner"}).%Id(),!
WRITE db.%SaveDocument({"firstName":"Fred","lastName":"Astare"}).%Id(),!
WRITE "Contains ",db.%Size()," documents: ",db.%ToJSON()
WRITE db.%SaveDocument({"firstName":"William","lastName":"Faulkner"},2).%Id(),!
WRITE !,"Contains ",db.%Size()," documents: ",db.%ToJSON()
```

2.4 データベース内のドキュメント数の計算 : %Size()

データベース内のドキュメント数を計算するには、%Size() メソッドを呼び出します。

ObjectScript

```
SET doccount = db.%Size()
WRITE doccount
```

2.5 データベース内のドキュメントの取得 : %GetDocument()

%DocumentId によってデータベースから単一のドキュメントを取得するには、次の例に示すように %GetDocument() メソッドを呼び出します。

ObjectScript

```
DO db.%GetDocument(2).%ToJSON()
```

このメソッドは、%Doc プロパティの内容のみを返します。例 :

```
{"firstName":"Bill","lastName":"Faulkner"}
```

このメソッドの戻り値の型は %Library.DynamicAbstractObject です。

指定された %DocumentId が存在しない場合、%GetDocument() は、ERROR #5809 例外 : “ロードするオブジェクトがみつかりません” を生成します。

%GetDocumentByKey() を使用して、キー値によってデータベースから単一のドキュメントを取得できます。

%FindDocuments() メソッドを使用して、%DocumentId によってデータベースから単一のドキュメントを返すこともできます。例 :

ObjectScript

```
DO db.%FindDocuments(["%DocumentId",2,"="]).%ToJSON()
```

このメソッドは、ラッパを含む完全な JSON ドキュメントを返します。

```
{"sqlcode":100,"message":null,"content":[{"%Doc":"{\"firstName\":\"Bill\",\"lastName\":\"Faulkner\"}",
"%DocumentId":2,"%LastModified":"2018-03-06 18:59:02.559"}]}
```

2.6 データベース内のドキュメントの検索 : %FindDocuments()

データベース内の 1 つ以上のドキュメントを検索し、そのドキュメントを JSON として返すには、%FindDocuments() メソッドを呼び出します。このメソッドは、3 つのオプションの位置述語 (制約配列、プロジェクション配列、および制限 key:value ペア) の任意の組み合わせを取ります。

位置述語のない次の例は両方とも、データベース内のすべてのドキュメントのすべてのデータを返します。

ObjectScript

```
WRITE db.%FindDocuments().%ToJSON()
```

ObjectScript

```
WRITE db.%FindDocuments(,,).%ToJSON()
```

2.6.1 制約述語配列

制約述語構文 `["property","value","operator"]` は、一致するドキュメントの内容全体を返します。検索条件には、プロパティ、値、および演算子を配列として指定します。演算子を指定しない場合、既定値として `"="` が設定されます。暗黙の AND ロジックを持つ制約述語の配列として、複数の制約を指定できます : `[["property","value","operator"],["property2","value2","operator2"]]`。制約述語はオプションです。

次の例では、ドキュメント ID が 2 より大きいすべてのドキュメントを返します。

ObjectScript

```
SET result = db.%FindDocuments(["%DocumentId",2,">"])
WRITE result.%ToJSON()
```

または、複数のメソッドを連鎖させることもできます。

ObjectScript

```
WRITE db.%FindDocuments(["%DocumentId",2,">"]).%ToJSON()
```

ドキュメントの内容が検索条件と一致する場合は、次のような結果が返されます。

```
{"sqlcode":100,"message":null,"content":[{"%Doc":"{\"firstName\":\"Fred\",\"lastName\":\"Astare\"}","%DocumentId":"3","%LastModified":"2018-03-05
18:15:30.39"},{"%Doc":"{\"firstName\":\"Ginger\",\"lastName\":\"Rogers\"}","%DocumentId":"4","%LastModified":"2018-03-05
 18:15:30.39"}]}
```

検索条件に一致するドキュメントがない場合は、次の結果が返されます。

```
{"sqlcode":100,"message":null,"content":[]}
```

key:value ペアでドキュメントを検索するには、そのキーのドキュメントプロパティを定義しておく必要があります。

ObjectScript

```
IF $SYSTEM.DocDB.Exists("People")
  { SET db = ##class(%DocDB.Database).%GetDatabase("People")}
ELSE {SET db = ##class(%DocDB.Database).%CreateDatabase("People") }
  WRITE db.%Size(),!
DO db.%CreateProperty("firstName","%String","$.firstName",0)
SET val = db.%SaveDocument({"firstName":"Fred","lastName":"Rogers"})
SET val = db.%SaveDocument({"firstName":"Serena","lastName":"Williams"})
SET val = db.%SaveDocument({"firstName":"Bill","lastName":"Faulkner"})
SET val = db.%SaveDocument({"firstName":"Barak","lastName":"Obama"})
SET val = db.%SaveDocument({"firstName":"Fred","lastName":"Astare"})
SET val = db.%SaveDocument({"lastName":"Madonna"})
SET result = db.%FindDocuments(["firstName","Fred","="])
WRITE result.%ToJSON()
```

次の例に示すように、%STARTSWITH、IN、NULL、NOT NULL を含め、さまざまな述語演算子を使用できます。

ObjectScript

```
SET result = db.%FindDocuments(["firstName","B","%STARTSWITH"])
WRITE result.%ToJSON()
```

ObjectScript

```
SET result = db.%FindDocuments(["firstName","Bill,Fred","IN"])
WRITE result.%ToJSON()
```

ObjectScript

```
SET result = db.%FindDocuments(["firstName","NULL","NULL"])
WRITE result.%ToJSON()
```

ObjectScript

```
SET result = db.%FindDocuments(["firstName","NULL","NOT NULL"])
WRITE result.%ToJSON()
```

2.6.2 プロジェクション述語配列

返されるドキュメントの値の一部のみを返すには、プロジェクションを使用します。オプションのプロジェクション述語 `["prop1","prop2",...]` は、対応する値を返すキーを示す配列です。プロジェクション配列でユーザ定義のキーを指定する場合、そのキーのドキュメントプロパティを定義しておく必要があります。

構文 `["property","value","operator"],["prop1","prop2",...]` は、一致するドキュメントから指定されたプロパティを返します。

ObjectScript

```
SET result = db.%FindDocuments(["firstName","Bill","="],["%DocumentId","firstName"])
WRITE result.%ToJSON()
```

制約述語の有無にかかわらず、プロジェクションを指定することができます。したがって、以下の両方とも有効な構文になります。

- `db.%FindDocuments(["property","value","operator"],[prop1,prop2,...])` 制約およびプロジェクション。
- `db.%FindDocuments(,[prop1,prop2,...])` 制約なし、プロジェクション。

2.6.3 制限述語

制限 key:value 述部 `{"limit":int}` を指定すると、最大で、指定した数の一致するドキュメントのみを返すことができます。

構文 `["property","value","operator"],["prop1","prop2",...],{"limit":int}` は、指定された制限数のドキュメントから指定されたプロパティを返します。

ObjectScript

```
SET result = db.%FindDocuments(["firstName","Bill","="],["%DocumentID","firstName"],{"limit":5})
WRITE result.%ToJSON()
```

この例では、最大 5 つのドキュメントからデータが返されます。

制約述語またはプロジェクション述語の有無にかかわらず、制限を指定することができます。したがって、以下のすべてが有効な構文になります。

- `db.%FindDocuments(["property","value","operator"],,{"limit":int})` 制約、プロジェクションなし、制限。
- `db.%FindDocuments(,[prop1,prop2,...],{"limit":int})` 制約なし、プロジェクション、制限。
- `db.%FindDocuments(,,{"limit":int})` 制約なし、プロジェクションなし、制限。

次の例では、制約なし、プロジェクション、制限を指定します。

ObjectScript

```
WRITE db.%FindDocuments(,["%DocumentId","lastName"],{"limit":3}).%ToJSON()
```

2.7 データベース内のドキュメントのクエリ：%ExecuteQuery()

%ExecuteQuery() メソッドを使用して、データベースからドキュメント・データを結果セットとして返すことができます。標準 SQL クエリ SELECT を指定して、FROM 節にデータベース名を指定します。パッケージ名 (スキーマ) なしでデータベース名を作成した場合は、デフォルトの `ISC_DM` スキーマを指定します。

次の例では、%Doc コンテンツ・データをデータベース内のすべてのドキュメントから結果セットとして取得します。

ObjectScript

```
SET rval=db.%ExecuteQuery("SELECT %Doc FROM ISC_DM.People")
DO rval.%Display()
```

次の例では、WHERE 節の条件を使用して、ドキュメント ID 値によって取得するドキュメントを制限します。

ObjectScript

```
SET rval=db.%ExecuteQuery("SELECT %DocumentId,%Doc FROM ISC_DM.People WHERE %DocumentId > 2")
DO rval.%Display()
```

次の例では、WHERE 節の条件を満たすすべてのドキュメントから、%DocumentId および lastName プロパティを取得します。ユーザ定義キーの値を取得するには、そのキーのドキュメント・プロパティを定義しておく必要があります。

ObjectScript

```
 SET rval=db.%ExecuteQuery("SELECT %DocumentId,lastName FROM ISC_DM.People WHERE lastName %STARTSWITH
'S'")
 DO rval.%Display()
```

クエリ結果セットの処理の詳細については、"完全な結果セットの返送" または "結果セットからの特定値の返送" を参照してください。

2.8 ドキュメントの削除 : %DeleteDocument()

ドキュメント ID またはデータ選択条件のいずれかを使用して、データベースからドキュメントを削除できます。

- %DeleteDocument() メソッドは、ドキュメント ID で指定された 1 つのドキュメントを削除します。

  ObjectScript

  ```
  SET val = db.%DeleteDocument(2)
  WRITE "deleted document = ",val.%ToJSON()
  ```

  正常に完了した場合、削除されたドキュメントの JSON 値が返されます。失敗した場合、StatusException エラーが返されます。

- %DeleteDocumentByKey() メソッドは、ドキュメントの内容で指定されたドキュメントを削除します。key:value ペアを指定します。

- %Clear() メソッドは、データベース内のすべてのドキュメントを削除し、データベースの oref (オブジェクト参照) を返します (次の例を参照してください)。

  ObjectScript

  ```
  SET dboref = db.%Clear()
  WRITE "database oref: ",dboref,!
  WRITE "number of documents:",db.%Size()
  ```

  これにより、次の例に示すように、メソッドを連鎖させることができます。

  ObjectScript

  ```
  WRITE db.%Clear().%SaveDocument({"firstName":"Venus","lastName":"Williams"}).%Id(),!
  ```

3

REST クライアント・メソッド

InterSystems IRIS® Data Platform の DocDB REST クライアント API は、REST からの DocDB の操作を可能にするメソッドを提供します。REST API は他の DocDB API とは異なります。他の API はオブジェクトに作用しますが、REST はリソースに作用するためです。このリソース指向は、REST の特性の基本となります。

この章の curl の例では、ポート番号 57774 を指定しています。これは、一例に過ぎません。実際は、ご使用の InterSystems IRIS インスタンスに適したポート番号を使用する必要があります。

curl では GET コマンドが既定なので、-X GET を省略した curl コマンドは、既定で -X GET になります。

DocDB の場合、唯一有効な Content-Type は application/json です。予期しない Content-Type が要求された場合、HTTP 応答コード = 406 が発行されます。

REST API の詳細は、″インターシステムズ・クラス・リファレンス″ の **%Api.DocDB.v1** を参照してください。REST の詳細は、″REST サービスの作成″ を参照してください。

3.1 データベースの管理

- **GetAllDatabases**: ネームスペース内のすべてのデータベースの名前が含まれる JSON 配列を返します。

```
curl -i -X GET -H "Content-Type: application/json" http://localhost:57774/api/docdb/v1/namespaceName
```

- **DropAllDatabases**: 現在のユーザが書き込み権限を持つネームスペース内のすべてのデータベースを削除します。

```
curl -i -X DELETE -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName
```

- **CreateDatabase**: ネームスペース内でデータベースを作成します。

```
curl -i -X POST -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/db/databaseName ?type= documentType& resource=
databaseResource
```

- **GetDatabase**: ネームスペース内のデータベースのデータベース定義を返します。

```
curl -i -X GET -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/db/databaseName
```

- **DropDatabase**: ネームスペースからデータベースを 1 つ削除します。

```
curl -i -X DELETE -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/db/databaseName
```

3.2 プロパティの管理

- CreateProperty: 指定されたデータベースで新規プロパティを作成したり、既存プロパティを置換したりします。プロパティは URL パラメータによって定義されます。また、プロパティはコンテンツではありません。パラメータはすべてオプションです。

  ```
  curl -i -X POST -H "Content-Type: application/json"
  http://localhost:57774/api/docdb/v1/namespaceName/prop/databaseName/
  propertyName?type= propertyType& path= propertyPath& unique=propertyUnique
  ```

  以下の例では、Wx データベースで City プロパティを作成しています。

  ```
  http://localhost:57774/api/docdb/v1/mysamples/prop/wx/city?type=%String&path=query.results.channel.location.city&0
  ```

- GetProperty: 指定されたデータベースからプロパティ定義を返します。

  ```
  curl -i -X GET -H "Content-Type: application/json"
  http://localhost:57774/api/docdb/v1/namespaceName/prop/databaseName/propertyName
  ```

  返されるプロパティ定義は、以下のような JSON 構造になります。

  ```
  {"content":{"Name":"city","Type":"%Library.String"}}
  ```

- DropProperty: 指定されたデータベースからプロパティ定義を削除します。

  ```
  curl -i -X DELETE -H "Content-Type: application/json"
  http://localhost:57774/api/docdb/v1/namespaceName/prop/databaseName/propertyName
  ```

以下で示されているのは、JSON プロパティ定義です。

```
{"content":{"Name":"mydocdb","Class":"DB.MyDocDb","properties":[Array[5]
   0: {"Name":%%OTD,"Type":"%RawString"},
   1: {"Name":%Concurrency,"Type":"%RawString"},
   2: {"Name":%Doc,"Type":"%Library.DynamicAbstractObject"},
   3: {"Name":%DocumentId,"Type":"%Library.Integer"},
   4: {"Name":%LastModified,"Type":"%Library.UTC"}
   ]}}
```

3.3 ドキュメントの挿入および更新

- SaveDocument: 指定されたデータベースに新規ドキュメントを挿入します。

  ```
  curl -i -X POST -H "Content-Type: application/json"
  http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/
  ```

  この URL の最後にスラッシュがある点に注意してください。

  1 つ以上のドキュメントを挿入するには、POST を実行します。要求の本文は、JSONドキュメント・オブジェクトまたはドキュメント・オブジェクトの JSON 配列のいずれかになります。ドキュメント・オブジェクトは、コンテンツのみのラップされていない形式にすることができる点に注意してください。このラップされていない形式では、必ず新しい %DocumentId で挿入されます。 データベース内で %DocumentId が存在しないラップされたドキュメント・オブジェクトを指定した場合、その %DocumentId で挿入されます。それ以外の場合、%DocumentId プロパティは無視され、新しい %DocumentId で挿入されます。要求が成功した場合、1 つの JSON ドキュメント・ヘッダ・オブジェクトまたは JSON ドキュメント・ヘッダ・オブジェクトの配列が返されます。要求が失敗した場合、JSON ヘッダ・オブジェクトがエラー・オブジェクトに置換されます。

- SaveDocument: 指定されたデータベース内の既存ドキュメントを置換します。

```
curl -i -X PUT -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/id
```

  特定の ID 位置で JSON ドキュメント・オブジェクトを 1 つ挿入します。指定された %DocumentId が既に存在する場合、既存ドキュメントが新規ドキュメントに置換されます。

- SaveDocumentByKey: 指定されたデータベース内の既存ドキュメントを置換します。

```
curl -i -X PUT -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/keyPropertyName/keyValue
```

3.4 ドキュメントの削除

- DeleteDocument: 指定されたデータベースからドキュメントを削除します。

```
curl -i -X DELETE -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/id
```

  %DocumentId で指定されたドキュメントを削除します。要求が成功した場合、指定されたドキュメントが削除され、ドキュメント・ラッパー・メタデータ `{"%DocumentId":<IDnum>,"%LastModified":<timestamp>}` が返されて、ステータス 200 (OK) になります。

- DeleteDocumentByKey: 指定されたデータベースからドキュメントを削除します。

```
curl -i -X DELETE -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/keyPropertyName/keyValue
```

3.5 ドキュメントの取得

- GetDocument: データベースから指定ドキュメントを返します。

```
curl -i -X GET -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/id? wrapped=true|false
```

- GetDocumentByKey: データベースから一意のキーとして定義されたプロパティでドキュメントを返します。

```
curl -i -X POST -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/doc/databaseName/keyPropertyName/keyValue
```

  FindDocuments: データベースからクエリ仕様と一致するすべてのドキュメントを返します。

```
curl -i -X POST -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/namespaceName/find/databaseName? wrapped=true|false
```

以下の curl スクリプトの例では、完全なユーザの資格情報およびヘッダ情報を指定しています。MySamples ネームスペースにある Continents ドキュメント・データベースのすべてのドキュメントが返されます。

```
curl --user _SYSTEM:SYS -w "\n\n%{http_code}\n" -X POST
-H "Accept: application/json" -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/mysamples/find/continents
```

ドキュメント・データベースから以下のような JSON データが返されます。

```
{"content":{"sqlcode":100,"message":null,"content":[
  {"%Doc":"{\"code\":\"NA\",\"name\":\"North America\"}","%DocumentId":"1","%LastModified":"2018-02-15
 21:33:03.64"},
  {"%Doc":"{\"code\":\"SA\",\"name\":\"South America\"}","%DocumentId":"2","%LastModified":"2018-02-15
 21:33:03.64"},
  {"%Doc":"{\"code\":\"AF\",\"name\":\"Africa\"}","%DocumentId":"3","%LastModified":"2018-02-15
21:33:03.64"},
  {"%Doc":"{\"code\":\"AS\",\"name\":\"Asia\"}","%DocumentId":"4","%LastModified":"2018-02-15
21:33:03.64"},
  {"%Doc":"{\"code\":\"EU\",\"name\":\"Europe\"}","%DocumentId":"5","%LastModified":"2018-02-15
21:33:03.64"},
  {"%Doc":"{\"code\":\"OC\",\"name\":\"Oceana\"}","%DocumentId":"6","%LastModified":"2018-02-15
21:33:03.64"},
  {"%Doc":"{\"code\":\"AN\",\"name\":\"Antarctica\"}","%DocumentId":"7","%LastModified":"2018-02-15
21:33:03.64"}]}}
```

制限：以下の curl スクリプトの例では、制限オブジェクトを指定して、Continents ドキュメント・データベースから返されるドキュメントを制限しています。この制限例では、返されるドキュメントの名前が A の文字から始まるドキュメントに制限されます。

```
curl --user _SYSTEM:SYS -w "\n\n%{http_code}\n" -X POST
-H "Accept: application/json" -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/mysamples/find/continents -d
'{"restriction":["Name","A","%STARTSWITH"]}'
```

ドキュメント・データベースから以下の JSON ドキュメントが返されます。

```
{"content":{"sqlcode":100,"message":null,"content":[
  {"%Doc":"{\"code\":\"AF\",\"name\":\"Africa\"}","%DocumentId":"3","%LastModified":"2018-02-15
21:33:03.64"},
  {"%Doc":"{\"code\":\"AS\",\"name\":\"Asia\"}","%DocumentId":"4","%LastModified":"2018-02-15
21:33:03.64"},
  {"%Doc":"{\"code\":\"AN\",\"name\":\"Antarctica\"}","%DocumentId":"7","%LastModified":"2018-02-15
21:33:03.64"}]}}
```

プロジェクション：以下の curl スクリプトの例では、Continents ドキュメント・データベースの各ドキュメントから返す JSON プロパティを投影しています。この例では、前の例と同じ制限が使用されています。

```
curl --user _SYSTEM:SYS -w "\n\n%{http_code}\n" -X POST
-H "Accept: application/json" -H "Content-Type: application/json"
http://localhost:57774/api/docdb/v1/mysamples/find/continents -d
'{"restriction":["Name","A","%STARTSWITH"],"projection":["%DocumentId","name"]}'
```

ドキュメント・データベースから以下のような JSON データが返されます。

```
{"content":{"sqlcode":100,"message":null,"content":[
  {"%Doc":"{"%DocumentId":"3","name":"Africa"}},
  {"%Doc":"{"%DocumentId":"4","name":"Asia"}},
  {"%Doc":"{"%DocumentId":"7","name":"Antarctica"}}]}}
```