
# InterSystems 外部サーバの使用法

Version 2024.1
2024-06-03

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

InterSystems 外部サーバの使用法
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03




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

# 図一覧

# 1

# InterSystems 外部サーバの概要

このドキュメントで取り上げる内容の詳細なリストは、″目次″ を参照してください。

InterSystems 外部サーバは、InterSystems IRIS と外部言語プラットフォーム間に、瞬時に完全に統合された双方向の接続を提供します。外部サーバは、以下のような高速で、簡単で、強力な機能を提供します。

### 瞬時アクセス

$system.external インタフェースにより、Java、.NET、および Python オブジェクトのすべてが事実上 ObjectScript 言語の一部となります。コマンドを発行すると外部サーバは自動的に起動されるため、事前のセットアップなしで外部言語プラットフォームに瞬時にアクセスできます。

### 共有セッション

共有セッションにより、ObjectScript と外部アプリケーションが、同じ双方向の接続を共有して、同じコンテキストおよびトランザクション内で機能することが可能になります。

### リモート・オブジェクト制御

プロキシ・オブジェクトは、共有セッションの一方の側が、もう一方の側にあるターゲット・オブジェクトをリアルタイムで制御できるようにします。ObjectScript では Java、.NET、および Python オブジェクトを作成および制御でき、これらの言語では ObjectScript オブジェクトを作成および制御できます。

### 完全に再入可能な双方向の接続

通常の単方向のクライアント/サーバ・リレーションシップは、もはや障壁にはなりません。双方向の再入により、どちら側の新しいオブジェクトまたはメソッド呼び出しも、いつでも既存の共有セッションに入ることができます。これにより、共有セッションの一方の側のアプリケーションが、クライアントのクエリを開始し、もう一方からの要求を処理するといったように、サーバとクライアントの両方の役割を果たせるようになります。

### 再入可能なネイティブの SDK メソッド

ネイティブの SDK メソッドも完全に再入可能で、共有セッションに含めることができるため、外部アプリケーションは膨大な InterSystems IRIS リソースに直接アクセスできるようになります。

このドキュメントは、ObjectScript で InterSystems 外部サーバを操作する方法を説明しています。“外部言語の操作”のセクションでは、ObjectScript コードで外部サーバを使用するために必要なほとんどの情報を提供しています。ここで取り上げるすべてのメソッドについての詳細な情報は、“$system.external インタフェースのクイック・リファレンス”を参照してください。ソフトウェア要件とオプションの設定手順は、“InterSystems 外部サーバの要件”を参照してください。

外部サーバでは、以前のダイナミック・オブジェクト・ゲートウェイ・テクノロジの強化および簡素化された形態を使用しています。オブジェクト・ゲートウェイの機能はすべて利用できるため、コードのアップグレードは、特定のクラスとメソッド参照を置き換えるだけで済みます (詳細は “オブジェクト・ゲートウェイ・コードのアップグレード” を参照してください)。

このドキュメントの残りの部分では、外部サーバの制御と構成のためのオプションのメソッドについて説明します。管理ポータルで外部サーバのアクティビティを表示する方法、および ObjectScript で接続を制御する方法は、“外部サーバ接続の管理”を参照してください。“外部サーバの定義のカスタマイズ”は、特殊な目的の新しい外部サーバ構成を定義するための詳細オプションについて説明しています。

InterSystems Native SDK は、InterSystems 外部サーバ環境の重要な部分です。この SDK は、アプリケーションで外部サーバ接続を最大限に活用できるようにする Java、.NET、および Python のフレームワークを提供します。詳細は、以下のドキュメントを参照してください。

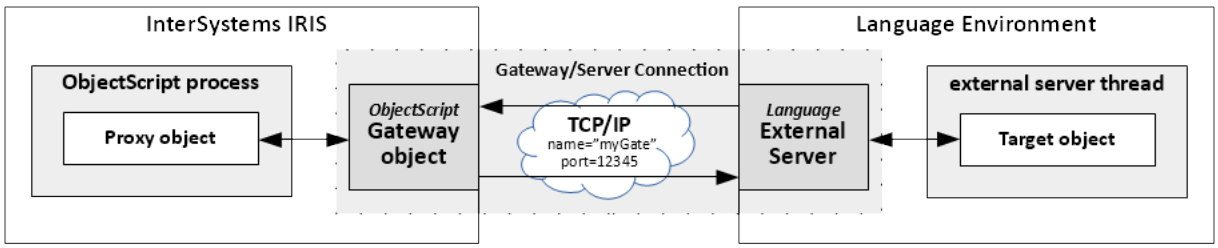
- Native SDK for Java の使用法
- Native SDK for .NET の使用法
- Native SDK for Python の使用法

# 2

# 外部言語の操作

$system.external インタフェースにより、対応する Java、.NET、または Python のターゲット・オブジェクトを制御する ObjectScript プロキシ・オブジェクトを生成できます。プロキシ・オブジェクトにはターゲットと同じメソッドとプロパティのセットがあり、プロキシの呼び出しがそれぞれターゲットによりエコーされます。プロキシとターゲット間の通信は、Java、.NET、または Python の External Server に接続された ObjectScript Gateway オブジェクトにより管理されます。

図 2-1: ゲートウェイ・オブジェクトと InterSystems 外部サーバ間の接続




- ゲートウェイ・プロセスは、InterSystems IRIS ネームスペース内で実行され、ObjectScript アプリケーションの接続を管理します。
- InterSystems 外部サーバ・プロセスは、外部言語環境 (Java、.NET、または Python) で実行され、外部言語共有ライブラリへのインタフェースを提供します。個々のサーバ・スレッドは、クラスのメソッドとプロパティへのアクセスを提供し、プロキシとターゲット・オブジェクト間の通信を管理します。
- 双方向のTCP/IP 接続により、ゲートウェイ・オブジェクトと外部サーバは、外部サーバ・インスタンスを一意に識別するポート番号を使用してメッセージを交換できるようになります。

以下のセクションでは、接続を確立し、ターゲットを定義し、プロキシ・オブジェクトを使用する方法を説明します。

- ゲートウェイの作成とプロキシ・オブジェクトの使用
- ターゲット・ソフトウェアへのパスの定義

# 2.1 ゲートウェイの作成とプロキシ・オブジェクトの使用

接続の開始、プロキシ・オブジェクトの作成、およびメソッドの呼び出しのプロセス全体を、1 つの文に圧縮できます。以下の例は Java 外部サーバを開始し、ターゲット・クラス java.util.Date のインスタンスのプロキシを作成し、日付を表示するメソッドを呼び出します。

```
write $system.external.getJavaGateway().new("java.util.Date").toString()

    Sun May 02 15:18:15 EDT 2021
```

この 1 行のコードは、プロキシ・オブジェクトの作成と使用のプロセス全体を示します。Java ゲートウェイ接続を確立し、クラス java.util.Date のインスタンスと対応する ObjectScript プロキシを作成し、メソッド toString() から返された日付を書きこみます。

Tipヒント InterSystems ターミナルでこの呼び出しを実行してみましょう。C# と Python での同等のコマンドを以下に示します。

```
write $system.external.getDotNetGateway.new("System.DateTime",0).Now
write $system.external.getPythonGateway.new("datetime.datetime",1,1,1).now().strftime(%c)
```

外部サーバは InterSystems IRIS のインストール時に自動的に設定され、それ以上注意を払わなくてもほぼ確実に動作します (そうでない場合、“外部サーバの定義のトラブルシューティング”を参照してください。希望の言語のプラットフォームを識別するパス設定を変更する必要がある場合があります)。

前の例では、すべてを 1 行に圧縮していましたが、その行は以下の 3 つの重要な動作を実行します。

- まず、ObjectScript と外部サーバ間の接続をカプセル化する ObjectScript Gateway オブジェクトを作成します。ゲートウェイを永続変数に割り当てることができます。

  ```
  set javaGate = $system.external.getJavaGateway()
  ```

- 次に、ゲートウェイ・オブジェクトの new() メソッドを呼び出します。これは java.util.Date のインスタンスを外部サーバ・スレッドに作成し、対応するプロキシ・オブジェクトを ObjectScript プロセスに作成します。プロキシ・オブジェクトを永続変数に割り当てることもできます。

  ```
  set dateJava = javaGate.new("java.util.Date")
  ```

- 最後に、プロキシのオブジェクト・メソッドを呼び出します。ターゲット・オブジェクトは呼び出しをエコーし、結果をプロキシに返します。

  ```
  write dateJava.toString()
  ```

以下の例は、3 つすべての言語でのこれらの手順を示しています。

### ゲートウェイ・オブジェクトの作成

Java、.NET、および Python 外部サーバそれぞれに、getJavaGateway()、getDotNetGateway()、getPythonGateway() の固有のゲートウェイ作成メソッドがあります。これらの各メソッドは既定の構成で外部サーバを開始し、Gateway オブジェクトを返して接続を管理します。

```
set javaGate = $system.external.getJavaGateway()
set netGate = $system.external.getDotNetGateway()
set pyGate = $system.external.getPythonGateway()
```

また、カスタマイズされた外部サーバ構成で使用するための汎用の getGateway() メソッドもありますが (詳細は“外部サーバの定義のカスタマイズ”を参照)、ほとんどの目的には既定のメソッドで十分なはずです。

### プロキシ・オブジェクトの作成

各ゲートウェイ・オブジェクトには、ターゲット・オブジェクトとプロキシ・オブジェクトを作成するための new() メソッドがあります。new() メソッドへの各呼び出しでは、クラス名と必要なすべての引数を指定します。呼び出しを実行すると、外部サーバはクラスのターゲット・インスタンスを作成し、ゲートウェイはターゲットと同じメソッドとプロパティのセットを持つ対応するプロキシ・オブジェクトを作成します。プロキシへの各呼び出しはターゲットによってエコーされ、結果がプロキシに返されます。

この例では、3 つの異なる外部サーバ (各言語に独自のサーバが必要であるため) を介して接続される 3 つのプロキシ・オブジェクトを作成します。

```
set dateJava = javaGate.new("java.util.Date")
set dateNet = newGate.new("System.DateTime",0)
set datePy = pyGate.new("datetime.datetime",1,1,1).now()
```

各プロキシは、その他の ObjectScript オブジェクトと同様に扱うことができます。3 つすべてを、同じ ObjectScript アプリケーションで使用できます。

### プロキシ・オブジェクトのメソッドとプロパティの呼び出し

同じ文で、3 つすべてのプロキシ・オブジェクトからのメソッド呼び出しとプロパティ呼び出しを使用できます。

```
  write !,"  Java: "_dateJava.toString(),!,"  .NET: "_dateNet.Now,!, "Python:
"_datePy.strftime("%c")

       Java: Sun May 02 15:18:15 EDT 2021
       .NET: 2021-05-02 16:38:36.9512565
    Python: Sun May 02 15:23:55 2021
```

Java および Python の例はメソッドの呼び出しで、.NET の例ではプロパティを使用しています。

これまでの例では、システム・クラスのみを使用していました。これらはいつでも利用可能なので、説明しやすいためです。その他の場合は、使用する前にクラスの場所を指定する必要があります。詳細は、後述する“ターゲット・ソフトウェアへのパスの定義”を参照してください。

# 2.2 ターゲット・ソフトウェアへのパスの定義

すべての Gateway オブジェクトには、特定言語のソフトウェア・ライブラリへのパスのリストを格納できます。Jave ゲートウェイは .jar ファイルを受け入れ、.NET ゲートウェイは .dll アセンブリを受け入れ、Python ゲートウェイは .py (モジュールまたはクラス) ファイルを受け入れます。

addToPath() メソッドでは、リストへの新しいパスを追加できます。path 引数は 1 つのパスを含む単純な文字列にするか、複数のパスを含む動的配列にすることができます。以下に例を示します。

### 単一パスの追加

```
do javaGate.addToPath("/home/myhome/someclasses.jar")
do netGate.addToPath("C:\Dev\myApp\somedll.dll")
do pyGate.addToPath("/rootpath/person.py")
```

Java ゲートウェイと .NET ゲートウェイは、1 つ以上の共有ライブラリを含むフォルダへのパスも受け入れます。高度な Python オプションの詳細は、“Python のターゲットの指定”を参照してください。

### 動的配列へのパスの追加

%DynamicArray クラスは、JSON 配列構造を簡単に作成できるようにする ObjectScript ラッパです。動的配列の %Push() メソッドを使用して、配列にパスの文字列を追加します。以下の例は、配列 pathlist に 2 つのパスを追加し、それを Java Gateway オブジェクトの addToPath() メソッドに渡します。

```
set pathlist = []
do pathlist.%Push("/home/myhome/firstpath.jar")
do pathlist.%Push("/home/myhome/another.jar")
do javaGate.addToPath(pathlist)
```

動的配列の詳細は、"JSON の使用" の “動的配列での %Push と %Pop の使用” を参照してください。

注釈　path 引数は、複数のパス文字列を含む %Library.ListOfDataTypes のインスタンスとしても指定できますが、使いやすさを考慮すると動的配列をお勧めします。

以下の例は、各言語にクラスを指定する方法を示しています。

### Java クラスへのパスの定義

Java の場合、パスにはフォルダまたは jar を指定できます。以下の例は、someclasses.jar へのパスを追加し、次に someclasses.classname のインスタンスのプロキシを作成します。

```
set javaGate = getJavaGateway()
do javaGate.addToPath("/home/myhome/someclasses.jar")
set someProxy = javaGate.new("someclasses.classname")
```

### .NET クラスへのパスの定義

.NET の場合、パスにはフォルダまたはアセンブリを指定できます。以下の例は、someassembly.dll へのパスを追加し、次に someassembly.classname のインスタンスのプロキシを作成します。

```
set netGate = getDotNetGateway()
do netGate.addToPath("C:\Dev\myApp\somedll.dll")
set someProxy = netGate.new("someassembly.classname")
```

### Python クラスへのパスの定義

Python の場合、パスにはモジュールまたはパッケージを指定できます。モジュールがパッケージの一部である場合、モジュールのパスはパッケージの最上位のディレクトリから開始するドット表記で指定する必要があります。例えば、モジュール Foo.py へのパスは、以下の 2 つの方法のどちらかで指定できます。

- モジュールとして処理される場合の標準のパス表記 : `C:\Dev\demo\Foo.py`
- パッケージ demo の一部として処理される場合のドット表記 : `C:\Dev\demo.Foo.py`

以下の例は、動的配列を使用して、フォルダ C:¥Dev¥demo¥ に含まれる 2 つの異なるファイルへのパスを追加します。ファイル Foo.py には、パッケージ化されていないクラス personOne が含まれ、ファイル Bar.py には、パッケージ demo の一部であるクラス personTwo が含まれます。new() を呼び出すと、以下のように両方のクラスのプロキシが作成されます。

```
set pyPaths = []
do pyPaths.%Push("C:\Dev\demo\Foo.py")
do pyPaths.%Push("C:\Dev\demo.Bar.py")

set pyGate = getPythonGateway()
do pyGate.addToPath(pyPaths)
set fooProxy = pyGate.new("Foo.personOne")
set barProxy = pyGate.new("demo.Bar.personTwo")
```

動的配列の詳細は、"JSON の使用" の “動的配列での %Push と %Pop の使用” を参照してください。

クラスのないモジュールまたはパッケージのターゲットを割り当てることも可能です (以降のセクションを参照)。

## 2.2.1 Python のターゲットの指定

Python のターゲットを指定する構文は、ターゲットがパッケージまたはクラスのいずれに含まれるか、その両方か、またはどちらでもないかによって異なります。以下の例は、パスが addToPath() で指定され、クラス名が new() で指定される場合を示しています。

クラスの例では、person がメイン・クラスで、company はそれがインポートするクラスです。インポートされるクラスは、それが別個のファイル内にあるとしても、指定する必要はありません。

クラスのないモジュールまたはパッケージのターゲットも割り当てられますが、それらは実質的に静的なメソッドおよびプロパティに制限されます (クラスなしでクラス・インスタンス・メソッドを持つことはできないため)。

### パッケージなし、クラスなし

パッケージもクラスもないモジュールは、ファイル名を使用します。new() の引数は、ファイル名の後にピリオドが続き、クラスがないことを示していることに注意してください。

```
do pyGate.addToPath("/rootpath/noclass.py")
set proxy = pyGate.new("noclass."))
```

### パッケージなし、1 つのファイル内に 2 つのクラス

メイン・クラス person およびインポートされるクラス company は両方ともファイル /rootpath/onefile.py 内にあります。

```
do pyGate.addToPath(/rootpath/onefile.py")
set proxy = pyGate.new("onefile.person")
```

### パッケージなし、別個のファイル内に 2 つのクラス

メイン・クラス person およびインポートされるクラス company は、/rootpath 内の別個のファイルに含まれます。

```
do pyGate.addToPath(/rootpath/person.py")
set proxy = pyGate.new("person.person")
```

### パッケージ、クラスなし

クラスなしのパッケージでは、パッケージ名とファイル名を使用します。この例でのファイルの実際のパスは /rootpath/demo/noclass.py になります。new() の引数はピリオドで終了し、クラスがないことを示しています。

```
do pyGate.addToPath(/rootpath/demo.noclass.py")
set proxy = pyGate.new("demo.noclass.")
```

### パッケージ、1 つのファイル内に 2 つのクラス

メイン・クラス person およびインポートされるクラス company は両方ともファイル /rootpath/demo/onefile.py 内にあります:

```
do pyGate.addToPath(/rootpath/demo.onefile.py")
set proxy = pyGate.new("demo.onefile.person")
```

### パッケージ、別個のファイル内に 2 つのクラス

メイン・クラス person およびインポートされるクラス company は、/rootpath/demo 内の別個のファイルに含まれます。

```
do pyGate.addToPath(/rootpath/demo.person.py")
set proxy = pyGate.new("demo.person.person")
```

# 3

# 外部サーバ接続の管理

このセクションでは、外部サーバの起動および停止方法と、現在のサーバ・アクティビティについての情報の取得方法について説明します。これには、管理ポータルを介してインタラクティブに実行する方法と、ObjectScript **$system.external** インタフェースを使用してプログラムによって実行する方法の 2 つがあります。

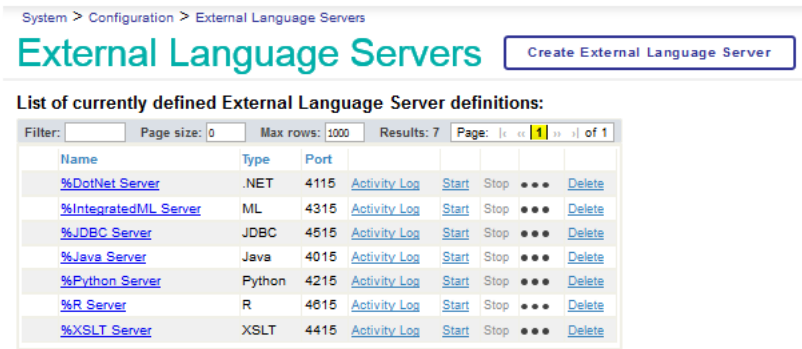
## 3.1 管理ポータルを使用した接続の制御

InterSystems 外部サーバの定義とは、構成設定の名前付きコレクションです。これらの設定を使用して、外部サーバ・プロセスを開始するコマンドが構築されます。例えば、定義には TCP/IP アドレス、ポート番号、およびクラスまたは実行可能ファイルへのパスなどの情報が含まれます。

管理ポータルには、定義済みのすべての外部サーバの現在のステータスを表示するページがあります。これにより、外部サーバを起動または停止したり、ゲートウェイ接続を制御および監視することが可能になります。以下にリストするすべての定義には、インストール時に構成された既定の設定が含まれています。これらの定義の変更または新しい定義の作成については、“外部サーバの定義のカスタマイズ” を参照してください。

注釈　既定では、すべてのサーバにゲートウェイ・リソースが必要です。これにより、サーバはパスフレーズでセキュリティ保護された接続で使用できるようになります。セキュリティが無効になっている場合を除き、ゲートウェイを開始および停止したり、ゲートウェイに接続するには、適切なリソースに対する USE 特権が必要です。Java、.NET、Python サーバには、**%Gateway_Object** リソースが必要です。詳細は、″承認ガイド″ を参照してください。

図 3-1: [External Servers] ページ ([システム管理]→[構成]→[接続性])




[InterSystems External Servers] ページを開くには、**システム管理 > 構成 > 接続性 > External Servers** に移動します。この例で、すべての名前は、既定の外部サーバ構成であることを示す **%** で始まります (InterSystems IRIS インストール時に自動的に定義されます)。**%Java Server**、**%Python Server**、および **%DotNet Server** の定義は、このドキュ

メントで取り上げる言語の既定の外部サーバ構成です。その他のリストされている定義は、その他の InterSystems インタフェースによって内部で使用される特殊な構成です (関連するドキュメントで説明しています)。

注釈　`%Java Server` を `%JDBC Server` と混同しないでください。これは、その他の InterSystems インタフェースによって内部で使用される特殊な構成です。

表示されている各行は 1 つの一意の構成を識別するもので、それを使用する外部サーバを監視および制御することを可能にします。例えば、この行は既定の .NET 外部サーバ構成をリストします。


この行には以下のフィールドが含まれます。

- [Name] はこの構成設定のコレクションの一意の識別子です。それによって表される構成を使用できるのは、一度に 1 つの外部サーバ・インスタンスのみです。[Name] フィールドをクリックすると、構成のほとんどの値を変更できるエディタが表示されます (詳細は “外部サーバの定義のカスタマイズ” を参照してください)。
- [Type] は、外部サーバが実行される言語を示しています。
- [Port] は、この構成によって使用されるポート番号を表示します。2 つの異なる定義で同じポート番号を指定できますが、同時には使用できません。2 つのゲートウェイ接続で同じポートを同時に使用することはできません。
- [Activity Log] は、InterSystems IRIS が開始されてから起動または停止されたこのサーバ定義のすべてのインスタンスについての詳細な情報を表示するページへのリンクです。$system.external インタフェースを使用して同じ情報を表示する方法は、次のセクションの “Activity Log の表示” を参照してください。
- Start/Stop では、外部サーバ・プロセスが現在実行中であるかどうかを制御できます。[Start] は、外部サーバのインスタンスを作成し、そのプロセスについての詳細な起動情報を表示します。外部サーバが既に実行中である場合は、[Stop] コマンドが代わりに表示されます。$system.external インタフェースを使用してこれを実行する方法は、次のセクションの “外部サーバの起動および停止” を参照してください。
- [Delete] では、外部サーバ定義を削除できます。$system.external インタフェースを使用して定義を作成および削除する方法は、“外部サーバ定義の作成と変更” を参照してください。

ほとんどの目的には既定の構成で十分ですが、追加の構成を定義すると便利な場合もあります。既存の定義を変更する方法または新しい定義を作成する方法は、“外部サーバの定義のカスタマイズ” を参照してください。

# 3.2 $system.external インタフェースを使用した接続の制御

このセクションでは、$system.external インタフェースを使用したゲートウェイ・プロセスの開始および停止、ゲートウェイ・サーバへの接続の管理、ゲートウェイのステータスに関する情報の取得の方法を説明します。

- 外部サーバの起動および停止
- Activity Log の表示

## 3.2.1 外部サーバの起動および停止

$system.external startServer() および stopServer() メソッドを使用すると、指定した外部サーバのインスタンスを開始または停止できます。isServerRunning() は、指定した外部サーバが現在実行中であるかどうかを示すブーリアン値を返します。メソッド・シグニチャは以下のとおりです。

```
startServer(serverName As %String) as %Boolean
```

```
stopServer(serverName As %String, verbose As %Boolean = 0) as %Boolean
```

```
isServerRunning(serverName As %String) as %Boolean
```

InterSystems 外部サーバは自動的に動作するよう設計されています。外部サーバは接続要求が受信されると起動し（既に実行中でない場合）、そのサーバを起動した InterSystems IRIS インスタンスがシャットダウンされると停止します。

注釈　既定では、すべてのサーバにゲートウェイ・リソースが必要です。これにより、サーバはパスフレーズでセキュリティ保護された接続で使用できるようになります。セキュリティが無効になっている場合を除き、ゲートウェイを開始および停止したり、ゲートウェイに接続するには、適切なリソースに対する USE 特権が必要です。Java、.NET、Python サーバには、**%Gateway_Object** リソースが必要です。詳細は、"承認ガイド" を参照してください。

通常、サーバはゲートウェイ・オブジェクトのインスタンスが作成されると起動されます。しかし、startServer() および stopServer() メソッドを使用すれば、ゲートウェイ・オブジェクトが存在しない場合でも外部サーバを制御できます（これは、以前のセクションの "管理ポータルを使用した接続の制御" で説明したように、管理ポータルの [　] / [　] リンクと同じように動作します）。

以下の例は、%Java Server のインスタンスが実行中であるかどうかをテストします。実行中でない場合は、外部サーバが起動され、返り値がチェックされて、起動が成功したことが確認されます。最後に、外部サーバが停止され、関連するアクティビティ・ログ・メッセージが表示されます。

### サーバの起動および停止

この例は、既定の Java 構成である %Java Server を使用する外部サーバ・インスタンスを起動および停止します。isServerRunning() メソッドは、開始前にサーバが期待される状態であることを確認するために使用されます。

```
set serverName = "%Java Server"
set isRunning = $system.external.isServerRunning(serverName)
write isRunning

   0
```

startServer() メソッドは、外部サーバの起動が成功した場合に (1) を返します。

```
if '(isRunning) set isRunning = $system.external.startServer(serverName)
write isRunning

   1
```

stopServer() メソッドも、成功または失敗を示すブーリアン値を返します。verbose 引数が true に設定されている場合、コンソールには Activity Log メッセージも表示されます。

```
set isRunning = $system.external.stopServer(serverName,1)

   2020-06-24 13:09:39  Stopping Java Gateway Server '%Java Server'
   2020-06-24 13:09:39  Stopping process '89381' that is monitoring the Gateway
   Server '%Java Server' on port '53180'
   2020-06-24 13:09:39  Shutting down the Gateway Server
   2020-06-24 13:09:39  Invoking %Connect with Server='127.0.0.1', Port='53180',
   Namespace='USER', Timeout='5'
   2020-06-24 13:09:39  Shutting down Java Gateway Server '%Java Server'
   2020-06-24 13:09:39  Return from %Shutdown: OK
   2020-06-24 13:09:39  Gateway Server stopped

write isRunning

   0
```

Activity Log のエントリは、次のセクションで説明するように後で取得することができます。

## 3.2.2 Activity Log の表示

getActivity() メソッドは、指定した InterSystems 外部サーバに対する一連の Activity Log エントリを返します。メソッド・シグニチャは以下のとおりです。

```
getActivity(serverName As %String, entryCount As %Integer = 10, verbose As
%Boolean = 0) as %Library.DynamicArray
```

管理ポータルの Activity Log インタフェースについては、“外部サーバの定義のトラブルシューティング” を参照してください。

各サーバは、サーバの状態についてのメッセージをログに記録します。getActivity() は、このログから指定された entryCount のメッセージ数を動的配列として返します。verbose が true である場合、現在のデバイスについてのメッセージも表示されます。

ダイナミック・オブジェクト・メソッド %ToJSON() を使用して、一度に設定のリスト全体を表示することができます。

注釈　%DynamicArray クラスは、JSON 文字列およびデータ構造を直接操作できるようにする、いくつかの関連する ObjectScript 機能の 1 つです。詳細は、″JSON の使用″ を参照してください。

### JSON での getActivity()

以下の例では、最初の 3 つの Activity Log のエントリを取得し、%ToJSON() を使用してそれらを表示します (わかりやすくするために改行を追加しています)。

```
set activity = $system.external.getActivity("%Java Server",3)
write activity.%ToJSON()

   [{"DateTime":"2021-05-08 17:19:28","Job":"15037","Port":4013,"Server":"%Java Server",
   "Text":"Starting Java Gateway Server '%Java Server'","Type":"Info"},
   {"DateTime":"2021-05-08 17:19:28","Job":"15037","Port":4013,"Server":"%Java Server",
   "Text":"Return from RunStartCmd: ERROR #5001: Java executable not found in the given
   directory: /nethome/bad/java/path/bin/","Type":"Error"},
   {"DateTime":"2021-05-08 17:19:28","Job":"15037","Port":4013,"Server":"%Java Server",
   "Text":"An error occurred while trying to start the Gateway Server","Type":"Info"}]
```

オプションで、verbose を true (1) に設定することで、ログの項目を現在のデバイス上に表示できます。

### 形式設定された表示での getActivity()

以下に、同じセットのログ・エントリを画面表示として示します。

```
set activity = $system.external.getActivity("%Java Server",3,1)
```

```
+--------------+-------+---------------------+--------+-------+-----------------------------+
| Server       | Port  | DateTime            | Type   | Job   | Text                        |
+--------------+-------+---------------------+--------+-------+-----------------------------+
| %Java Server | 4013  | 2021-05-08 17:19:28 | Info   | 15037 | Starting Java Gateway Serve |
|              |       |                     |        |       | r '%Java Server'            |
| %Java Server | 4013  | 2021-05-08 17:19:28 | Error  | 15037 | Return from RunStartCmd: ER |
|              |       |                     |        |       | ROR #5001: Java executable  |
|              |       |                     |        |       | not found in the given dire |
|              |       |                     |        |       | ctory: /nethome/bad/java/pa |
|              |       |                     |        |       | th/bin/                     |
| %Java Server | 4013  | 2021-05-08 17:19:28 | Info   | 15037 | An error occurred while try |
|              |       |                     |        |       | ing to start the Gateway Se |
|              |       |                     |        |       | rver                        |
+--------------+-------+---------------------+--------+-------+-----------------------------+
```

# 4

# 外部サーバの定義のカスタマイズ

外部サーバの定義とは、構成設定の名前付きコレクションです。これらの設定を使用して、外部サーバ・プロセスを開始するコマンドが構築されます。例えば、定義にはTCP/IPアドレス、ポート番号、およびクラスまたは実行可能ファイルへのパスなどの情報が含まれます。

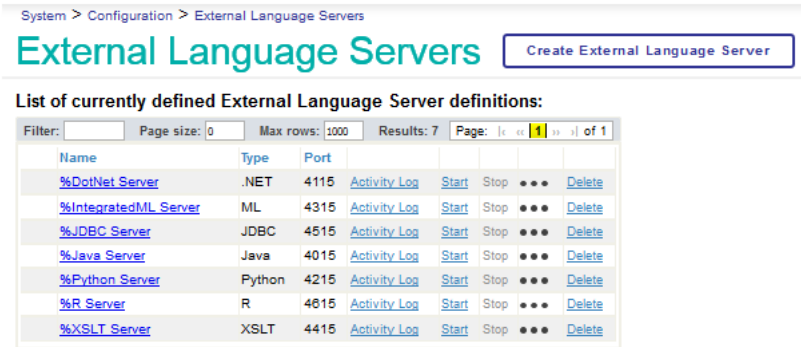
ほとんどの目的には、事前定義された外部サーバ設定により必要なすべての機能が提供されますが (“外部サーバ接続の管理” を参照)、必要に応じて既定の設定を簡単に変更できます。または、カスタム設定を含むまったく新しい外部サーバの定義を作成することもできます。

このセクションでは、外部サーバの構成設定の名前付きコレクションを管理する方法と、新しい外部サーバの定義を作成する方法を説明します。これには、管理ポータルを介してインタラクティブに実行する方法と、ObjectScript $system.external インタフェースを使用してプログラムによって実行する方法の 2 つがあります。

## 4.1 管理ポータルでの定義のカスタマイズ

外部サーバの定義は、InterSystems IRIS データベースに格納された構成設定の名前付きコレクションです。この情報を使用して、外部サーバ・プロセスを開始するコマンド行が構築されます。外部サーバの定義にはそれぞれ一意の名前があり、一度に 1 つの外部サーバによってのみ使用できる接続を定義します。定義には、管理ポータルの [External Servers] ページで名前をクリックしてアクセスできます。

図 4-1: [External Servers] ページ ([システム管理]→[構成]→[接続性])




注釈　このセクションの説明に従ってサーバ定義を作成および変更するには、**%Admin_ExternalLanguageServerEdit:USE** 特権 (既定では、**%Manager** ロールが保持) が必要です。

以下のセクションでは、既存の定義にアクセスし、新しい定義を作成するために使用される管理ポータルのページについて説明します。

- 新しい外部サーバ・フォーム
- Java 用の外部サーバ構成の定義
- .NET 用の外部サーバ構成の定義
- Python 用の外部サーバ構成の定義
- 詳細設定の定義

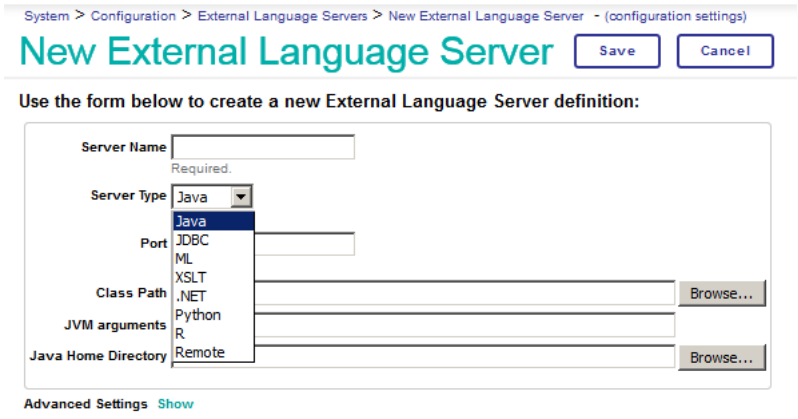
## 4.1.1 新しい外部サーバ・フォーム

サーバ定義を作成および変更するには、**%Admin_ExternalLanguageServerEdit:USE** 特権 が必要です。既定のサーバの設定を変更する代わりに、新しいサーバ定義を作成する方が便利な場合があります。システムの既定値とは異なる言語環境、バージョン、およびシステム構成を指定できます。ほとんどの構成設定はオプションで、その他の値を指定しなければ、既定値が使用されます。

各外部サーバ定義には、サーバ名、タイプ (現時点では Java、.Net、および Python)、およびポートを含める必要があります。追加のプロパティはタイプによって異なり、言語プラットフォームのパス、ユーザ・コードのパス、およびその他言語固有のオプションを定義できます。ログ、接続パラメータ、および監視の構成可能な項目もあります。

[External Servers] ページ (**システム管理 > 構成 > 接続性 > External Servers**) には、**[Create External Server]** ボタンがあります。


ボタンをクリックすると、以下に示すフォームが表示されます。このフォームを使用して、このドキュメントで取り上げるすべての外部サーバ (外部 Java サーバ、外部 .NET サーバ、および外部 Python サーバ) と、ここでは取り上げないその他の特殊なタイプのサーバの構成を定義できます。




この画像は、選択したタイプが Java である場合の、サーバ・タイプのリストを示しています。必須の [サーバ名]、[サーバタイプ]、および [ポート] のオプションに加えて、フォームには 3 つの Java 固有のフィールドが表示されています。任意の選択されたタイプについて、同様のタイプ固有のフィールドが表示されます。

以下のフィールドは、すべての言語で同一です。

- ー 必須。

このサーバ定義に割り当てる一意の名前です。この名前は、格納される定義のデータベース識別子として使用され、定義の保存後は変更できません。

- ー 必須。

  選択した言語によってのみ使用されるフィールドを表示するには、[Java]、[.NET]、または [Python] を選択します。

- ー 必須。

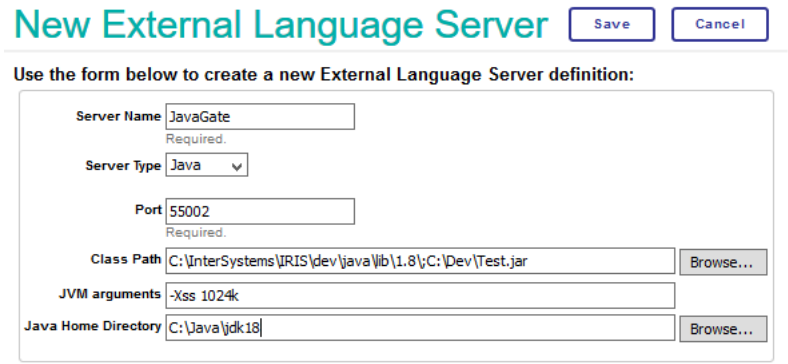
  外部サーバと InterSystems IRIS 間の通信に使用する一意の TCP ポート番号です。2 つの外部サーバ接続で同じポートを同時に使用することはできません。

以下のセクションでは、Java、.NET、および Python のタイプ固有のフィールドについて説明します。

# 4.2 Java 用の外部サーバ構成の定義

すべての言語に共通のフィールドの詳細は、“新しい外部サーバ・フォーム” を参照してください。[ ]、[JVM ]、および [Java ] フィールドは、Java に固有です。

図 4-2: 外部 Java サーバ定義の作成




- (必須) ー この外部サーバ構成の一意の名前です。
- (必須) ー [Java] を選択すると、Java 外部サーバでのみ使用されるフィールドが表示されます ([JDBC] を選択しないでください。これは以前の InterSystems インタフェースによってのみ使用される特殊な構成です)。
- (必須) ー この外部サーバ構成によって使用される一意の TCP ポート番号です。
-

  このフィールドはクラス・パスに追加されます。これには、プロキシを生成するために必要なターゲット・クラスを含むすべての jar ファイルに対するパスが含まれている必要があります。InterSystems 外部サーバの .jar ファイルを含める必要はありません。スペースまたは複数のファイルを含むファイル・パスを指定する場合は、クラスパスを引用符で囲って、プラットフォームに適切なセパレータを付ける必要があります。

- JVM

  外部サーバを開始するコマンドで JVM に渡されるオプションの引数。例えば、必要に応じてシステム・プロパティ (Dsystemvar=value) や最大ヒープ・サイズ (Xmx256m) などの設定を指定できます。

- Java

  外部サーバで必要となる Java バージョンのパス。指定しない場合、これは既定で管理サーバの [Javaホームディレクトリ] の設定になります。

・ ー 詳細は“詳細設定の定義”を参照してください。

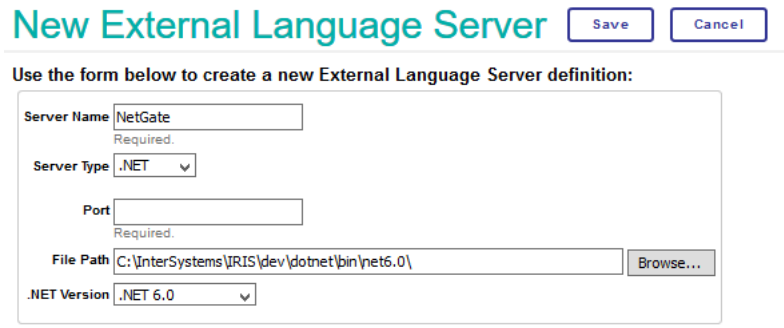
[保存] をクリックして、データベースにサーバ定義を保存します。[ ] は、この定義で一意の識別子で、定義の保存後は変更できません。

既存のサーバ定義は、メインの [External Servers] ページでその定義の [編集] リンクをクリックすることで編集できます。[編集] ページはここで示すページと同じですが、[ ] フィールドを変更できない点のみが異なります。

# 4.3 .NET 用の外部サーバ構成の定義

すべての言語に共通のフィールドの詳細は、“新しい外部サーバ・フォーム”を参照してください。[ ] と [.NET ] フィールドは、.NET に固有です。

図 4-3: 外部 .NET サーバ定義の作成




・ (必須) ー この外部サーバ構成の一意の名前です。

・ (必須) ー [.NET] を選択すると、.NET 外部サーバでのみ使用されるフィールドが表示されます。

・ (必須) ー この外部サーバ構成によって使用される一意の TCP ポート番号です。

・

.NET に必要なサーバの実行可能ファイルを含むディレクトリのフル・パスを指定します (“.NET 外部サーバのセットアップ”を参照)。これは、外部サーバを開始するコマンドによって使用されます。このフィールドが指定されていない場合、起動コマンドは選択した .NET の既定の実行可能ファイルを使用します。

.NET を使用する場合、システムによっては <install-dir>¥dev¥dotnet (すべての .NET 外部サーバ・アセンブリの最上位ディレクトリ) を指定する必要がある場合があります。

・ .NET

外部サーバの実行可能ファイルで必要とされる .NET バージョンを指定します。.NET オプションは、すべてのオペレーティング・システムで利用可能です。Windows では、.NET Framework のバージョンもサポートされています。

・ ー 詳細は、“詳細設定の定義”および“.NET の詳細設定”を参照してください。

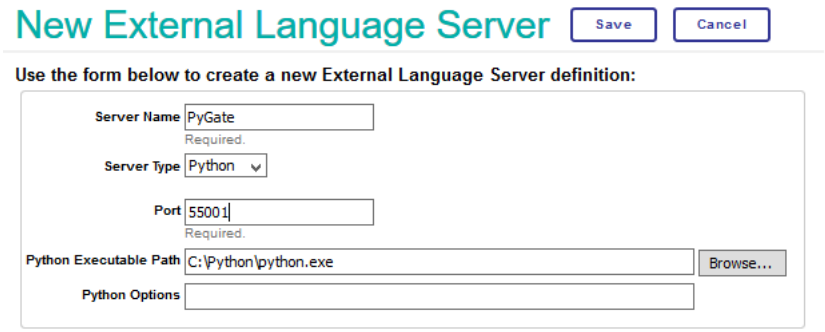
[保存] をクリックして、データベースにサーバ定義を保存します。[ ] は、この定義で一意の識別子で、定義の保存後は変更できません。

既存のサーバ定義は、メインの [External Servers] ページでその定義の [編集] リンクをクリックすることで編集できます。[編集] ページはここで示すページと同じですが、[ ] フィールドを変更できない点のみが異なります。

# 4.4 Python 用の外部サーバ構成の定義

すべての言語に共通のフィールドの詳細は、“新しい外部サーバ・フォーム”を参照してください。[Python Executable Path] と [Python Options] フィールドは、Python に固有です。

図 4-4: 外部 Python サーバ定義の作成




- （必須）– この外部サーバ構成の一意の名前です。
- （必須）– [Python] を選択すると、Python 外部サーバでのみ使用されるフィールドが表示されます。
- （必須）– この外部サーバ構成によって使用される一意の TCP ポート番号です。
- Python Executable Path

  外部サーバを実行するために使用される Python 実行可能ファイルの完全修飾ファイル名。場所を指定せずに使用できる既定の Python インタプリタがある場合、この設定はオプションです。
- Python Options

  外部サーバを開始する Python コマンドを構築する際に含める追加のコマンド行設定を定義するオプションのプロパティです。
- – 詳細は、“詳細設定の定義”および“Python の詳細設定”を参照してください。

[保存] をクリックして、データベースにサーバ定義を保存します。[ ] は、この定義で一意の識別子で、定義の保存後は変更できません。

既存のサーバ定義は、メインの [External Servers] ページでその定義の [編集] リンクをクリックすることで編集できます。[編集] ページはここで示すページと同じですが、[ ] フィールドを変更できない点のみが異なります。

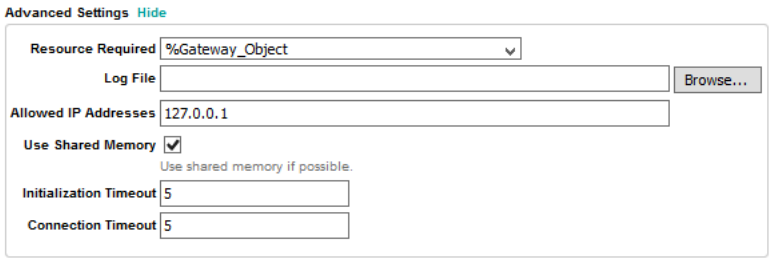
# 4.5 詳細設定の定義

の既定値は、ほとんどの場合、変更する必要はありません。

## 4.5.1 すべての外部サーバの詳細オプション

次のオプションは、Java、.NET、および Python で使用できます。

図 4-5: すべての外部サーバで使用される詳細設定




·

既定では、すべてのサーバにゲートウェイ・リソースが必要です。これにより、サーバはパスフレーズでセキュリティ保護された接続で使用できるようになります。Java、.NET、Python サーバには、**USE** 特権を持つ **%Gateway_Object** リソースが必要です。システム管理者はカスタマイズされたリソースを作成し、それを特定のゲートウェイを使用するために必要なリソースにすることができます。このフィールドを空の文字列に設定して、ゲートウェイをパブリックにすることもできます。ただし、すべてのゲートウェイを適切なリソースで保護することを強くお勧めします。詳細は、"承認ガイド" を参照してください。

·

ホスト・マシン上のログ・ファイルの完全パス名です。サーバへの接続が開かれたことや閉じられたことなどのイベントや、ターゲット・クラスをプロキシ・クラスにマッピングするときに生じたエラー状態についての外部サーバ・メッセージが記録されます。パフォーマンスを考慮して、このオプションのプロパティは、デバッグ時にのみ使用してください。

· `IP`

外部サーバが実行されるホスト・マシンの IP アドレスまたは名前です。既定は `"127.0.0.1"` (ローカル・マシン) です。マシンに対してローカルなすべての IP アドレス (127.0.0.1、VPN アドレスなど) でリッスンするには、`"0.0.0.0"` を指定します。

· `Use Shared Memory`

使用可能な場合、共有メモリを使用するかどうかを示します (既定値は true)。

·

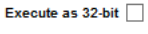
外部サーバの初期化が失敗したと見なされるまで待機する秒数です。

·

外部サーバへの接続試行が失敗したと見なされるまで待機する秒数です。

## 4.5.2 .NET の詳細設定

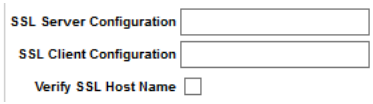
“すべての外部サーバの詳細オプション” で説明した設定に加えて、.NET の詳細設定には、外部サーバを 32 ビットとして実行するオプションが含まれます。




· `32-bit` — 64 ビット・プラットフォーム上の外部サーバにのみ適用されます。このプロパティにチェックが付けられている場合、外部サーバは 32 ビットとして実行されます。既定は 64 ビットの実行となります。

### 4.5.3 Python の詳細設定

“すべての外部サーバの詳細オプション”で説明した設定に加えて、Python の詳細設定には、TSL/SSL 構成のオプションが含まれます (“TLS 構成の作成または編集”を参照)。




- SSL — サーバ TLS/SSL に使用する SSL/TLS 構成の名前。
- SSL — クライアント TLS/SSL に使用する SSL/TLS 構成の名前。
- SSL — このプロパティにチェックが付けられている場合、TLS/SSL クライアントはホスト名検証を実行します。

## 4.6 $system.external インタフェースによる定義のカスタマイズ

管理ポータルには、外部サーバの構成を定義および変更するための便利なインタラクティブ・ツールが用意されていますが (前のセクションで説明したとおり)、同じことを $system.external インタフェースを使用して ObjectScript で実行することもできます。

注釈 このセクションの説明に従ってサーバ定義を作成および変更するには、**%Admin_ExternalLanguageServerEdit:USE** 特権 (既定では、**%Manager** ロールが保持) が必要です。

管理ポータルに入力する情報とまったく同じ情報を指定して、数行の ObjectScript コードで外部サーバの定義を作成および使用できます。既存のすべての外部サーバ定義についての詳細情報にもアクセスできます。

- ″getServer() を使用した構成設定の取得″ では、個々の定義について完全な情報を取得する方法を説明しています。
- ″既存の定義についてのその他の情報の取得″ では、既存のすべての定義についての基本的な情報を取得する方法を説明しています。
- ″外部サーバの定義の作成と変更″ では、すべての言語の外部サーバ構成を定義および変更する方法を説明しています。

注釈 %DynamicObject と %DynamicArray

このセクションで説明する多くのメソッドは、%DynamicObject および %DynamicArray のインスタンスを受け入れるか、返します。これらのクラスにより、ObjectScript で JSON 文字列とデータ構造を処理できるようになります。詳細および例は、″JSON の使用″ を参照してください。

## 4.6.1 getServer() を使用した構成設定の取得

$system.external インタフェースの getServer() メソッドは、指定された外部サーバ定義のすべての構成設定を含む %DynamicObject を返します。メソッド・シグニチャは以下のとおりです。

```
getServer(serverName As %RawString) as %Library.DynamicObject
```

例えば、以下のコードは既定の Java 構成である、%Java Server の設定を返します。

```
set srv = $system.external.getServer("%Java Server")
write srv

   2@%Library.DynamicObject
```

各フィールドはダイナミック・オブジェクトのプロパティであるため、以下のように標準のプロパティの表記を使用して、希望のフィールドを表示できます。

```
write "Name:"_srv.Name_", Type:"_srv.Type_", Port:"_srv.Port,!

   Name:%Java Server, Type:Java, Port:4015
```

%DynamicObject メソッド、%GetIterator() および %GetNext() を使用してダイナミック・オブジェクトのプロパティを反復処理できます (詳細は、"JSON の使用" の "%GetNext() を使用したダイナミック・エンティティの反復処理" を参照してください)。

または、ダイナミック・オブジェクト・メソッド %ToJSON() を使用して、一度に設定のリスト全体を表示することもできます。以下の例は、getServer() を使用して 3 つの既定の外部サーバ構成 (%Java Server、%Python Server、または %DotNet Server) のすべての設定を取得し、%ToJSON() を使用してその情報を表示します。

### 既存の Java 外部サーバ定義の表示

この例では、既定の Java サーバ構成のすべてのフィールドを含む %DynamicObject を返して表示します (わかりやすくするために改行を追加しています)。

```
write $system.external.getServer("%Java Server").%ToJSON()

   {"Name":"%Java Server",
   "FullName":"user3:IRIS_DS2:%Java Server","Type":"Java",
   "Port":4015,"LogFile":"","AllowedIPAddresses":"127.0.0.1",
   "ConnectionTimeout":5,"HeartbeatFailureAction":"R","HeartbeatFailureRetry":300,
   "HeartbeatFailureTimeout":30,"HeartbeatInterval":10,"InitializationTimeout":5,
   "UsePassphrase":0,"UseSharedMemory":1,"passphraseList":"",
   "SSLConfigurationServer":"","SSLConfigurationClient":"",
   "JVMArgs":"","JavaHome":"","ClassPath":""}
```

この例の最後の行は、Java 外部サーバによってのみ使用されるフィールドを表示します (各フィールドの詳細は "Java 用の外部サーバ構成の定義" を参照してください)。

### 既存の .NET 外部サーバ定義の表示

この例では、既定の .NET サーバ構成のすべてのフィールドを含む %DynamicObject を返して表示します (わかりやすくするために改行を追加しています)。

```
write $system.external.getServer("%DotNet Server").%ToJSON()

   {"Name":"%DotNet Server",
   "FullName":"user3:IRIS_DS2:%DotNet Server","Type":".NET",
   "Port":4115,"LogFile":"","AllowedIPAddresses":"127.0.0.1",
   "ConnectionTimeout":5,"HeartbeatFailureAction":"R","HeartbeatFailureRetry":300,
   "HeartbeatFailureTimeout":30,"HeartbeatInterval":10,"InitializationTimeout":5,
   "UsePassphrase":0,"UseSharedMemory":1,"passphraseList":"",
   "SSLConfigurationServer":"","SSLConfigurationClient":"",
   "DotNetVersion":"C2.1","Exec32":0,"FilePath":""}
```

この例の最後の行は、.NET 外部サーバによってのみ使用されるフィールドを表示します (各フィールドの詳細は “.NET 用の外部サーバ構成の定義” を参照してください)。

#### 既存の Python 外部サーバ定義の表示

この例では、既定の Python サーバ構成のすべてのフィールドを含む %DynamicObject を返して表示します (わかりやすくするために改行を追加しています)。

```
write $system.external.getServer("%Python Server").%ToJSON()

   {"Name":"%Python Server",
   "FullName":"user3:IRIS_DS2:%Python Server","Type":"Python",
   "Port":4215,"LogFile":"","AllowedIPAddresses":"127.0.0.1",
   "ConnectionTimeout":5,"HeartbeatFailureAction":"R","HeartbeatFailureRetry":300,
   "HeartbeatFailureTimeout":30,"HeartbeatInterval":10,"InitializationTimeout":5,
   "UsePassphrase":0,"UseSharedMemory":1,"passphraseList":"",
   "SSLConfigurationServer":"","SSLConfigurationClient":"",
   "PythonOptions":"","PythonPath":""}
```

この例の最後の行は、Python 外部サーバによってのみ使用されるフィールドを表示します (各フィールドの詳細は “Python 用の外部サーバ構成の定義” を参照してください)。

## 4.6.2 既存の定義についてのその他の情報の取得

$system.external インタフェースの serverExists()、getServers()、および getServerLanguageVersion() メソッドは、データベースに格納された外部サーバ定義についての情報を返します。これらはローカル・データベース・クエリであるため、これらのメソッドのいずれも外部サーバ接続を必要としません。

### serverExists()

serverExists() メソッドは、外部サーバの定義が存在する場合に true (1) を返し、そうでない場合に false (0) を返します。メソッド・シグニチャは以下のとおりです。

```
serverExists(serverName As %RawString) as %Boolean
```

以下に例を示します。

```
write $system.external.serverExists("foo")

   0

write $system.external.serverExists("%Java Server")

   1
```

### getServers()

getServers() メソッドは、すべての外部サーバ定義の名前を含む %DynamicArray を返します。メソッド・シグニチャは以下のとおりです。

```
getServers() as %Library.DynamicArray
```

以下に例を示します。

```
set srv = $system.external.getServers()
write srv.%ToJSON()

   ["%DotNet Server","%IntegratedML Server","%JDBC Server","%Java Server",
   "%Python Server","%R Server","%XSLT Server","JavaGate","NetGate2","PyGate2"]
```

この例では、最後の 2 つの名前はユーザにより作成された定義です (慣例により、事前定義された既定の構成のみが "%" で始まる名前を持ちます)。

### getServerLanguageVersion()

getServerLanguageVersion() メソッドは、指定した外部サーバ定義の外部言語バージョンの文字列を返します。メソッド・シグニチャは以下のとおりです。

```
getServerLanguageVersion(serverName As %RawString) as %String
```

以下に例を示します。

```
write $system.external.getServerLanguageVersion("%Java Server")

   8
```

getServer() メソッドを呼び出して関連する情報を含むダイナミック・オブジェクトを取得することで、画面表示をより便利なものにできます（“getServer() を使用した構成設定の取得”を参照してください）。

```
set javaDef = $system.external.getServer("%Java Server")
set LanguageVersion = $system.external.getServerLanguageVersion(javaDef.Name)

write javaDef.Name_" language version is: "_javaDef.Type_" "_LanguageVersion

   %Java Server version is: Java 8
```

## 4.6.3 外部サーバ定義の作成と変更

$system.external インタフェースにより、createServer()、modifyServer()、および deleteServer() メソッドを使用して外部サーバ定義を操作できます。これらのメソッドはデータベースに格納された定義を直接操作するため、これら 3 つのすべてにおいてデータベース・アクセスのために %Admin_Manage リソースが必要です。対応する外部サーバの実行中は、定義を変更できません。

### createServer()

createServer() メソッドは、%DynamicObject 内の設定のリストを受け入れ、新しいサーバ定義（既定値を含む）を %DynamicObject として返します。メソッド・シグニチャは以下のとおりです。

```
createServer(serverDef As %DynamicObject) as %DynamicObject
```

入力リストで定義可能なフィールド・セットは、getServer() により返されるセットと同じです（既定の設定の完全なリストは、“getServer() を使用した構成設定の取得”を参照してください）。

入力定義では、すべての必須フィールド（Name、Type、および Port）の値と、カスタマイズするその他のフィールドの値を指定する必要があります。すべての入力値は、createServer() の呼び出し時に検証されます。以下のように、ダイナミック・オブジェクト・プロパティとして入力設定を指定できます。

```
set def = {}
set def.Name = "AnotherServer"
set def.Type = "Java"
set def.Port = 5309
set def.ClassPath = "C:/dev/pathname/"
```

または JSON 文字列として指定できます。

```
set def = {"Name":"MyNewServer","Type":"Java","Port":5309, "ClassPath": "C:/dev/pathname/"}
```

この例の名前が、% 文字で始まらないことに注意してください。慣例により、事前定義された既定の構成のみが、"%" で始まる名前を持ちます。

Name の値は一意である必要があります。そのため、この例では createServer() を呼び出す前に serverExists() を呼び出してテストします。

```
set badName = $system.external.serverExists(def.Name)
if ('badName) set srv = $system.external.createServer(def)
write srv.%ToJSON()
```

```
   {"Name":"MyNewServer","FullName":"user3:IRIS_DS2:MyNewServer","Type":"Java",
   "Port":5309,"LogFile":"","AllowedIPAddresses":"127.0.0.1",
   "ConnectionTimeout":5,"HeartbeatFailureAction":"R","HeartbeatFailureRetry":300,
   "HeartbeatFailureTimeout":30,"HeartbeatInterval":10,"InitializationTimeout":5,
   "UsePassphrase":0,"UseSharedMemory":1,"passphraseList":"",
   "SSLConfigurationServer":"","SSLConfigurationClient":"",
   "JVMArgs":"","JavaHome":"","ClassPath":"C:/dev/pathname/"}
```

この例では 4 つの必須フィールドと ClassPath のみが指定されていますが、返されるダイナミック・オブジェクトにはすべてのフィールドが含まれ、必要に応じて既定値を割り当てています。

### modifyServer()

modifyServer() メソッドは、%DynamicObject に含まれる設定のリストを受け入れ、変更された定義を含む %DynamicObject を返します。メソッド・シグニチャは以下のとおりです。

```
modifyServer(serverDef As %DynamicObject) as %DynamicObject
```

入力ダイナミック・オブジェクトは、JSON 文字列として、または (この例のように) 一連のプロパティ定義として指定できます。

```
set mod = {}
set mod.Name = "%Java Server"
set mod.JavaHome = "/Library/Java/Contents/Home/"
```

[ ] および [ ] 以外のすべてのフィールドで新しい値を指定できます。このメソッドは既存の定義に対して機能するため、これらのプロパティは変更できません。

modifyServer() は、対応する外部サーバの実行中に呼び出しを試行すると、エラーをスローします。isServerRunning() を使用してサーバ上でチェックを実施し、stopServer() を使用して必要に応じてサーバを停止します (詳細は “$system.external インタフェースを使用した接続の制御” を参照してください)。

```
set isRunning = $system.external.isServerRunning(mod.Name)
if (isRunning) $system.external.stopServer(mod.Name)
set modifiedserver = $system.external.modifyServer(mod)
```

このメソッドは、変更された定義のすべてのフィールドを含むダイナミック・オブジェクトを返します。

```
write modifiedserver.%ToJSON()
```

```
   {"Name":"%Java Server","FullName":"user3:IRIS_DS2:%Java Server","Type":"Java",
   "Port":4015,"LogFile":"","AllowedIPAddresses":"127.0.0.1",
   "ConnectionTimeout":5,"HeartbeatFailureAction":"R","HeartbeatFailureRetry":300,
   "HeartbeatFailureTimeout":30,"HeartbeatInterval":10,"InitializationTimeout":5,
   "UsePassphrase":0,"UseSharedMemory":1,"passphraseList":"",
   "SSLConfigurationServer":"","SSLConfigurationClient":"",
   "JVMArgs":"","JavaHome":"/Library/Java/somename/Contents/Home/","ClassPath":""}
```

### deleteServer()

deleteServer() メソッドは、既存の外部サーバ定義を削除し、削除された設定を含むダイナミック・オブジェクトを返します。メソッド・シグニチャは以下のとおりです。

```
deleteServer(serverName As %String) as %DynamicObject
```

deleteServer() は、対応する外部サーバの実行中に呼び出しを試行すると、エラーをスローします。isServerRunning() を使用してサーバ上でチェックを実施し、stopServer() を使用して必要に応じてサーバを停止します（詳細は“$system.external インタフェースを使用した接続の制御”を参照してください）。

```
set oldServer = "MyOldServer"
set isRunning = $system.external.isServerRunning(oldServer)
if (isRunning) $system.external.stopServer(oldServer)

set deletedDef = $system.external.deleteServer(oldServer)
write $system.external.serverExists(oldServer)

   0
```

このメソッドは、削除された定義のすべてのフィールドを含むダイナミック・オブジェクトを返します。

```
write deletedDef.%ToJSON()

   {"Name":"MyOldServer","FullName":"user3:IRIS_DS2:MyOldServer","Type":"Java",
   "Port":5309,"LogFile":"","AllowedIPAddresses":"127.0.0.1",
   "ConnectionTimeout":5,"HeartbeatFailureAction":"R","HeartbeatFailureRetry":300,
   "HeartbeatFailureTimeout":30,"HeartbeatInterval":10,"InitializationTimeout":5,
   "UsePassphrase":0,"UseSharedMemory":1,"passphraseList":"",
   "SSLConfigurationServer":"","SSLConfigurationClient":"",
   "JVMArgs":"","JavaHome":"","ClassPath":""}
```

# 5

# InterSystems 外部サーバの要件

すべての InterSystems 外部サーバは、インストール時に事前定義された既定の構成に自動的に設定されます。言語プラットフォームが既定の場所にインストールされている場合、追加の構成は必要ないはずです。事前定義された設定で外部サーバが起動に失敗する場合、ほとんどの場合サポート対象の言語プラットフォームへの正しいパスを設定することで問題を解決できます (詳細は、このセクションの最後の “外部サーバの定義のトラブルシューティング” を参照してください)。

このセクションでは、Java、.NET、および Python のすべての外部サーバの既定の場所とサポートされるバージョンについての情報を提供します。

- InterSystems IRIS の要件
- Java 外部サーバのセットアップ
- .NET 外部サーバのセットアップ
- Python 外部サーバのセットアップ
- 外部サーバの定義のトラブルシューティング
- オブジェクト・ゲートウェイ・コードのアップグレード

## 5.1 InterSystems IRIS の要件

InterSystems 外部サーバと通信するには、InterSystems IRIS のインスタンスがバージョン 2020.4 以降である必要があります。Java、.NET、および Python の外部サーバは、これらの言語プラットフォームの既定の設定が使用できる場合に自動的に機能するよう設計されています。(以下のセクションの説明のとおり) サーバ構成に対する小さな変更が必要になる場合もあります。

Tipヒント **<install-dir> の特定**

インストール・ディレクトリへのパスは、以下の ObjectScript コマンドを入力することでターミナルに表示できます。

```
write $SYSTEM.Util.InstallDirectory()
```

# 5.2 Java 外部サーバのセットアップ

Java 外部サーバの実行可能ファイルのバージョンは、Java のサポートされる各バージョンに対して提供されます。通常は、インストール時に、既定の Java サーバ構成 (`%Java Server`) に対して適切なバージョンが自動的に指定されます。

各サーバのバージョンは、install-dir¥dev¥java¥bin の異なるサブディレクトリに配置されています。この install-dir に InterSystems IRIS の希望のインスタンスが含まれます。

以下のバージョンの Java 外部サーバは、標準インストールに含まれています。

- Java 8: install-dir¥dev¥java¥lib¥JDK18¥intersystems-gateway-3.2.0.jar
- Java 11: install-dir¥dev¥java¥lib¥JDK11¥intersystems-gateway-3.2.0.jar

`JAVA_HOME` 設定で指定された以外の JVM を使用する場合、必要な言語プラットフォームへのパスを設定するための追加の構成が必要になります (“Java 用の外部サーバ構成の定義” を参照してください)。

注釈　これらのコンポーネントを使用するためには、適切なバージョンの Java をシステムにインストールする必要があります。InterSystems IRIS のインストール手順では、いかなるバージョンの Java もインストールやアップグレードは行われません。

# 5.3 .NET 外部サーバのセットアップ

.NET 外部サーバのアセンブリのバージョンは、.NET のサポートされる各バージョンに対して提供されます。通常は、インストール時に、既定の .NET 外部サーバ構成 (`%DotNet Server`) に対して適切な外部サーバのバージョンが自動的に指定されます。

すべてのアセンブリが Windows でサポートされ、.NET Core 2.1 も Linux および MacOS でサポートされます。各バージョンは、install-dir¥dev¥dotnet¥bin の異なるサブディレクトリに配置されています。この install-dir に InterSystems IRIS の希望のインスタンスが含まれます。以下のアセンブリを使用できます。

.NET バージョン 2.0 :

- install-dir¥dev¥dotnet¥bin¥v2.0.50727¥InterSystems.Data.Gateway.exe
- install-dir¥dev¥dotnet¥bin¥v2.0.50727¥InterSystems.Data.Gateway64.exe

.NET バージョン 4.0 :

- install-dir¥dev¥dotnet¥bin¥v4.0.30319¥InterSystems.Data.Gateway.exe
- install-dir¥dev¥dotnet¥bin¥v4.0.30319¥InterSystems.Data.Gateway64.exe

.NET バージョン 4.5 :

- install-dir¥dev¥dotnet¥bin¥v4.5¥InterSystems.Data.Gateway.exe
- install-dir¥dev¥dotnet¥bin¥v4.5¥InterSystems.Data.Gateway64.exe

.NET Core 2.1 :

- install-dir¥dev¥dotnet¥bin¥Core21¥IRISGatewayCore21.dll

Core 2.1 のインストール手順は、"InterSystems ソフトウェアでの .NET の使用法" の ".NET Core 2 のインストールと構成" を参照してください。

一部のアプリケーションでは、.NET Framework アセンブリを使用して管理されていないコード・ライブラリがロードされることがあります。サポートされる各バージョンに対して 32 ビットと 64 ビットの両方のアセンブリが提供されています。これにより、32 ビット・ライブラリをロードできる 64 ビット Windows 向けゲートウェイ・アプリケーションを作成することが可能になります。

システムの既定以外のバージョンを使用する場合、必要な言語プラットフォームへのパスを設定するための追加の構成が必要になります (".Net 用の外部サーバ構成の定義" を参照してください)。

注釈　これらのアセンブリを使用するためには、サポートされるバージョンの .NET Framework をシステムにインストールする必要があります。InterSystems IRIS のインストール手順では、いかなるバージョンの .NET Framework もインストールやアップグレードは行われません。

# 5.4 Python 外部サーバのセットアップ

Python 外部サーバでは、Python 3.6.6 以降が必要です。Python のサポートされるバージョンがシステムで使用可能な場合、インストール時に、既定の Python 外部サーバ構成 (`Python Server`) に対して、通常は Python 実行可能ファイルへの正しいパスが自動的に指定されます。

Python 3 のサポートされるバージョンがシステムの既定ではない場合、必要なインスタンスへのパスを設定するために追加の構成が必要になります ("Python 用の外部サーバ構成の定義" を参照してください)。

インストール・ファイルは **intersystems_irispython-3.2.0-py3-none-any.whl** で、**<install-dir>¥dev¥python¥** にあります (ここで、<install-dir> は InterSystems IRIS インスタンスのルート・ディレクトリです)。このディレクトリには、その他の **.whl** ファイルも含まれる場合があります。これらは外部サーバに関係がないため、無視してもかまいません。(最新バージョンのインストレーション・ファイルは、"InterSystems IRIS Driver Packages" ページからもダウンロードできます。)

以下のコマンドを使用して Python 外部サーバのパッケージをインストールします。

```
python -m pip install --upgrade <path>\intersystems_irispython-3.2.0-py3-none-any.whl
```

`--user` オプションを使用しないでください。

注釈　このパッケージを使用するためには、サポートされるバージョンの Python 3 をシステムにインストールする必要があります。InterSystems IRIS のインストール手順では、いかなるバージョンの Python もインストールやアップグレードは行われません。

# 5.5 外部サーバの定義のトラブルシューティング

サポートされるすべての言語で、言語プラットフォームが適切にインストールされ、InterSystems IRIS で使用できる状態である必要があります。外部サーバが起動しない場合、Activity Log のエラー・メッセージに原因が示されていることがよくあります。最も一般的な構成の問題は、言語プラットフォームへの無効なパスです。

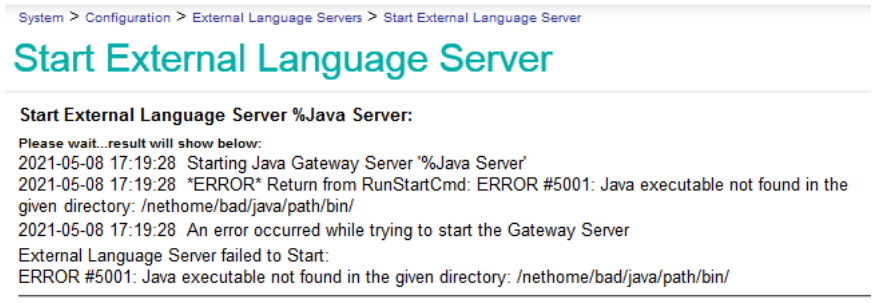
[External Servers] ページ (**システム管理 > 構成 > 接続性 > External Servers**) には、既定の各外部サーバ構成のリストが含まれます。

図 5-1: InterSystems 外部サーバ ([システム管理]→[構成]→[接続性])


このドキュメントで説明する既定の外部サーバは、`%DotNet Server`、`%Java Server`、および `%Python Server` です。`[Activity Log]` リンクをクリックすることで、外部サーバの以前の開始と停止に関するすべてのメッセージを表示できます。

[ ] をクリックして、外部サーバをテストできます。これにより、起動コマンドが表示され、外部サーバが起動に失敗した場合はエラー・メッセージがリストされます。例えば、以下のメッセージは Java 実行可能ファイルへのパスが間違っていることを示しています。

図 5-2: 起動時のエラー・メッセージの例




この例を生成するために、Java Home の `%Java Server` の設定は、意図的に ¥nethome¥java¥bad¥path に変更されています。

注釈　Java 言語の既定の構成は、`%Java Server` で定義されます。これを `%JDBC Server` と混同しないでください。これは、特定の古い InterSystems インタフェースによってのみ使用される特殊な構成です。

各言語にはオプションの構成設定があり、それを使用して言語のパスを指定できます (詳細は、"管理ポータルでの定義のカスタマイズ" の Java、.NET、および Python のそれぞれの言語固有の管理ポータルのページを参照してください)。

# 5.6 オブジェクト・ゲートウェイ・コードのアップグレード

外部サーバでは、以前のダイナミック・オブジェクト・ゲートウェイ・テクノロジの強化および簡素化された形態を使用しています。オブジェクト・ゲートウェイの機能はすべて利用できるため、コードのアップグレードは、特定のクラスとメソッド参照を置き換えるだけで済みます。以下のコードは、いくつかの一般的なアクティビティを実行するための旧式の方法と新しい方法を示しています。

### サーバの起動とゲートウェイ・オブジェクトの取得

アクティブなサーバ接続により Gateway オブジェクトを取得するプロセスには、以下のように 2 つの異なる ObjectScript クラスのメソッドに対する複数の呼び出しが含まれます。

```
  set status = ##class(%Net.Remote.Service).OpenGateway("JavaGate",.GatewayInfo)
  set name = GatewayInfo.Name
  set port = GatewayInfo.Port
  set server = GatewayInfo.Server
  if ('##class(%Net.Remote.Service).IsGatewayRunning(server,port,,.status)) {
    set status = ##class(%Net.Remote.Service).StartGateway(name)
  }
set gateway = ##class(%Net.Remote.Gateway).%New()
set status = gateway.%Connect(server, port, "USER")
```

外部サーバでは、これはやや単純になります。

```
set gateway = $system.external.getJavaGateway()
```

1 つの呼び出しで、すべてが適切に設定された Gateway オブジェクトを作成し、外部サーバへの接続を自動的に開始します。外部サーバに手動でアクセスすることもできますが (“外部サーバの起動および停止” を参照)、ほとんどのアプリケーションでこれは不要です。

### クラス・パスの指定とプロキシ・オブジェクトの作成

オブジェクト・ゲートウェイ・コードでは、プロキシを作成する前に、いくつかの異なる ObjectScript クラスへの参照と、非常に多くのサーバ固有の情報を必要としていました。

```
set path = ##class(%ListOfDataTypes).%New()
do path.Insert("C:\Dev\SomeClasses.jar")
do ##class(%Net.Remote.Service).OpenGateway("JavaGate",.GatewayInfo)
set gateway = ##class(%Net.Remote.Gateway).%New()
do gateway.%Connect(GatewayInfo.Server, GatewayInfo.Port, "USER",,path)
set proxy = ##class(%Net.Remote.Object).%New(gateway,"SomeClasses.ClassOne")
```

ここでも再び、外部サーバのコードは大幅に簡素化されています。

```
set gateway = getJavaGateway()
do gateway.addToPath("C:\Dev\SomeClasses.jar")
set proxy = gateway.new("SomeClasses.ClassOne")
```

新しい addToPath() メソッドも、複数のクラス・パスを追加するためのより単純な方法を提供します。

### クラス・メソッドの呼び出し

以前の %ClassMethod() の代わりに、外部サーバでは新しい Gateway invoke() メソッドを使用します。

```
// Object Gateway
set num =
##class(%Net.Remote.Object).%ClassMethod(gateway,"Demo.ReverseGateway","factorial",num-1)

// external server
set num = gateway.invoke("Demo.ReverseGateway","factorial",num-1)
```

# 6

# ObjectScript $system.external インタフェースのクイック・リファレンス

これは、ObjectScript **$system.external** インタフェースのクイック・リファレンスです。このインタフェースは、ObjectScript にすべての InterSystems 外部サーバへのプログラムによるアクセスを提供します。

注釈　この章は、このドキュメントの読者の利便性を目的としたものであり、最終的なリファレンスではありません。最も包括的な最新情報については、**%system.external** のオンライン・クラス・ライブラリ・ドキュメントを参照してください。

**$system.external.**Help() メソッドを呼び出すことで、コマンド行で同じ情報を取得できます。すべてのメソッドのリストを取得するには、以下を呼び出します。

```
do $system.external.Help()
```

また、特定のメソッド methodName についての詳細な情報を取得するには、以下を呼び出します。

```
do $system.external.Help("methodName")
```

## 6.1 用途別のすべてのメソッド

**ゲートウェイとプロキシ・オブジェクト**

“外部言語の操作”を参照してください。

ゲートウェイ・オブジェクトの作成

- getDotNetGateway() は、既定の .NET 外部サーバへの接続を取得します。
- getJavaGateway() は、既定の Java 外部サーバへの接続を取得します。
- getPythonGateway() は、既定の Python 外部サーバへの接続を取得します。

プロキシ・オブジェクトの作成

- new() は、外部クラスのインスタンスに結合された新しいプロキシ・オブジェクトを返します。
- addToPath() は、実行可能ファイルへのパスを、現在の言語のゲートウェイ・パスに追加します。

外部クラスの静的メソッドおよびプロパティへのアクセス

- invoke() は外部オブジェクト・メソッドを呼び出し、返り値を取得します。
- getProperty() は、外部オブジェクトからの静的プロパティの値を取得します。
- setProperty() は、外部オブジェクト内の静的プロパティの値を設定します。

### 外部サーバの管理

“外部サーバ接続の管理”を参照してください。

- startServer() はサーバを起動します。
- stopServer() はサーバを停止します。
- isServerRunning() は要求されたサーバが実行中である場合に true を返します。
- getActivity() は指定したサーバの ActivityLog エントリを取得します。

### 外部サーバの構成のカスタマイズ

“外部サーバの定義のカスタマイズ”を参照してください。

外部サーバ定義に関する情報の取得

- getServer() は、サーバ定義を含むダイナミック・オブジェクトを取得します。
- getServers() は、既存のすべての外部サーバ定義の名前を取得します。
- getServerLanguageVersion() は、サーバ定義の構成された外部言語バージョンの文字列を取得します。
- serverExists() はサーバ定義が存在する場合に true を返します。

外部サーバ定義の管理

- createServer() は新しいサーバ定義を作成します。
- deleteServer() は既存のサーバ定義を削除します。
- modifyServer() は既存のサーバ定義を変更します。

# 6.2 $system.external メソッドの詳細

### addToPath()

**$system.external**.addToPath() は、現在の言語ゲートウェイ・パスに path を追加します。ここで、path 文字列には 1 つ以上の実行可能ファイルへのパスが含まれます。

```
addToPath(path:%RawString)
```

返り値 :なし

path 引数には、単一のパスを含む単純な文字列を指定できます (Java の場合、これはフォルダまたは jar URL にできます)。複数のパスを追加するには、追加するパスを含む動的配列または **%Library.ListOfDataTypes** のインスタンスを渡します。

エラーが発生した場合、この関数は例外をスローします。

パラメータ :

- path — 単一のパスを含む文字列、または複数のパスを含む %DynamicArray または %ListOfDataTypes

### createServer()

$system.external.createServer() は新しい外部サーバ定義を作成します。この関数では、%Admin_Manage リソースが必要です。

```
createServer(serverDef As %DynamicObject) as %DynamicObject
```

パラメータ：

- serverDef — 新しい外部サーバ設定を含む %DynamicObject

関連項目：“外部サーバ定義の作成と変更”

### deleteServer()

$system.external.deleteServer() は、既存の外部サーバ定義を削除します。この関数では、%Admin_Manage リソースが必要です。

```
deleteServer(serverName As %String) as %DynamicObject
```

パラメータ：

- serverName — 既存の外部サーバ定義の名前

関連項目：“外部サーバ定義の作成と変更”

### getActivity()

$system.external.getActivity() は、指定された外部サーバの指定された数の ActivityLog エントリを含む %DynamicArray を返します。

```
getActivity(serverName As %String, entryCount As %Integer = 10,
verbose As %Boolean = 0) as %Library.DynamicArray
```

パラメータ：

- serverName — 既存の外部サーバ定義の名前
- entryCount — 返される ActivityLog エントリの数
- verbose — true の場合、現在のデバイスのエントリを表示

versbose が true の場合、これらの行は現在のデバイス上にも表示されます。

関連項目：“Activity Log の表示”

### getExternalLanguage()

$system.external.getExternalLanguage() は、外部サーバからの外部言語を返します。

```
getExternalLanguage(externalServerName As %String) as %String
```

パラメータ：

- externalServerName — 現在接続されているサーバの名前

関連項目：″Gateway.″getExternalLanguage()

### getExternalLanguageVersion()

$system.external.getExternalLanguageVersion() は、外部サーバからの外部言語のバージョンを返します。

```
getExternalLanguageVersion(externalServerName As %String) as %String
```

パラメータ：

- externalServerName — 現在接続されているサーバの名前

関連項目：″Gateway.″getExternalLanguageVersion()

### getGateway()

$system.external.getGateway() は、外部サーバへの新しいゲートウェイ接続を返します。

```
getGateway(externalServer As %RawString) as %External.Gateway
```

パラメータ：

- externalServer — 接続されるサーバの名前

このメソッドは、キャッシュされた既存のゲートウェイ接続を取得しません。常に、要求された外部サーバへの新しいゲートウェイ接続を取得します。

### getDotNetGateway()

$system.external.getDotNetGateway() は、既定の .NET 外部サーバへの新しいゲートウェイ接続を返します。getGateway("%DotNet Server") と同等（″getGateway()″ を参照）。

```
getDotNetGateway() As %External.Gateway
```

このメソッドは、キャッシュされた既存のゲートウェイ接続を取得しません。常に、既定の .NET 外部サーバへの新しいゲートウェイ接続を取得します。

関連項目：“ゲートウェイの作成とプロキシ・オブジェクトの使用”

### getJavaGateway()

$system.external.getJavaGateway() は、既定の Java 外部サーバへの新しいゲートウェイ接続を返します。getGateway("%Java Server") と同等（″getGateway()″ を参照）。

```
getJavaGateway() As %External.Gateway
```

このメソッドは、キャッシュされた既存のゲートウェイ接続を取得しません。常に、既定の Java 外部サーバへの新しいゲートウェイ接続を取得します。

関連項目：“ゲートウェイの作成とプロキシ・オブジェクトの使用”

### getPythonGateway()

$system.external.getPythonGateway() は、既定の Python 外部サーバへの新しいゲートウェイ接続を返します。getGateway("%Python Server") と同等（″getGateway()″ を参照）。

```
getPythonGateway() As %External.Gateway
```

このメソッドは、キャッシュされた既存のゲートウェイ接続を取得しません。常に、既定の Python 外部サーバへの新しいゲートウェイ接続を取得します。

関連項目：“ゲートウェイの作成とプロキシ・オブジェクトの使用”

### getProperty()

$system.external.getProperty() は、外部クラスからの静的プロパティの値を返します。

```
getProperty(externalServerName As %String, externalClass As %String, propertyName As %String)
as %ObjectHandle
```

パラメータ：

- `externalServerName` ― 現在接続されているサーバの名前
- `externalClass` ― 外部クラスの名前
- `propertyName` ― 外部プロパティの名前

関連項目：″Gateway.″getProperty()

### getServer()

$system.external.getServer() は、指定された外部サーバ定義からの構成設定を含む %DynamicObject を返します。

```
getServer(serverName As %RawString) as %Library.DynamicObject
```

パラメータ：

- `serverName` ― 既存の外部サーバ定義の名前

注：類似した名前の getServers() メソッドは、定義済みのすべての外部サーバ構成の名前を返します。

関連項目：“getServer() を使用した構成設定の取得”

### getServerLanguageVersion()

$system.external.getServerLanguageVersion() は、外部サーバ構成の外部言語バージョンの文字列を返します。

```
getServerLanguageVersion(serverName As %RawString) as %String
```

パラメータ：

- `serverName` ― 既存の外部サーバ定義の名前

この関数は、ほとんどの場合外部サーバへの接続を確立しません。構成から言語バージョンを返すだけです。この関数は外部コマンドを実行する場合がありますが、外部サーバを起動することはありません。

関連項目：“既存の定義についてのその他の情報の取得”

### getServers()

$system.external.getServers() は、定義済みのすべての外部サーバ構成の名前を含む %DynamicArray を返します。

```
getServers() as %Library.DynamicArray
```

注：類似した名前の getServer() メソッドは、1 つの外部サーバの詳細な構成設定を返します。

関連項目：“既存の定義についてのその他の情報の取得”

### Help()

$system.external.Help() は、$system.external メソッドのリストを含む文字列を返します。オプションの method 引数が指定された場合は、そのメソッドについての詳細な情報が返されます。

```
Help(method As %String = "") as %String
```

パラメータ：

- methodName — 詳細な説明が返されるメソッドの名前を含むオプションの文字列

関連項目：″Gateway.″Help()

### invoke()

$system.external.invoke() は、外部コードを呼び出します。このメソッドが式として呼び出されると、これは外部コードによって返される任意の値を返します。

```
invoke(externalServerName As %String,externalClass As %String, externalMethod As %String,
args... As %RawString) as %String
```

パラメータ：

- externalServerName — 現在接続されているサーバの名前
- externalClass — 外部クラスの名前
- externalMethod — 外部メソッドの名前
- args...— 指定されたクラス・コンストラクタに対するゼロ個以上の引数

外部コードによって値が返されない場合は、<COMMAND> 例外がスローされます。externalClass は、外部コードのコンテナの名前です (Java または .NET クラス名、Python クラスまたはモジュール名)。externalMethod は、externalClass から呼び出す外部コード単位 (関数、メソッド、またはその他の言語固有の単位) の名前です。返り値は、外部コードによって返される値です。外部メソッドが値を返さない場合、invoke() は値を返しません。

関連項目：″Gateway.″invoke()

### isServerRunning()

$system.external.isServerRunning() は、要求されたサーバが実行中である場合に true を返します。

```
isServerRunning(arg As %RawString) as %Boolean
```

パラメータ：

- arg — 外部サーバ名。(getServer() により返される) サーバ定義を含む文字列またはダイナミック・オブジェクトのいずれかになります。

関連項目：″Gateway.isServerRunning()″、“外部サーバの起動および停止”

### modifyServer()

$system.external.modifyServer() は、既存の外部サーバ定義内の設定を変更します。この関数では、%Admin_Manage リソースが必要です。

```
modifyServer(serverDef As %DynamicObject) as %DynamicObject
```

パラメータ：

- serverDef — 変更された外部サーバ設定を含む %DynamicObject

関連項目：“外部サーバ定義の作成と変更”

### new()

$system.external.new() は、外部クラスのインスタンスに結合された新しいプロキシ・オブジェクトを返します。externalClass および必要に応じて追加のコンストラクタの引数を渡します。

```
new(externalServerName As %String, externalClass As %String, args... As %RawString) as
%Net.Remote.Object
```

パラメータ：

- `externalServerName` ― 現在接続されているサーバの名前
- `externalClass` ― 外部クラスの名前
- `args...` ― 指定されたクラス・コンストラクタに対するゼロ個以上の引数

関連項目：″Gateway.″new()

### serverExists()

$system.external.serverExists() は、serverName が既存の外部サーバ定義である場合に `true` を返します。

```
serverExists(serverName As %RawString) as %Boolean
```

パラメータ：

- `serverName` ― 既存の外部サーバ定義の名前

関連項目：“既存の定義についてのその他の情報の取得”

### setProperty()

$system.external.setProperty() は、外部クラス内の静的プロパティの値を設定します。

```
setProperty(externalServerName As %String, externalClass As %String, propertyName As %String,
value As %RawString)
```

パラメータ：

- `externalServerName` ― 現在接続されているサーバの名前
- `externalClass` ― 外部クラスの名前
- `propertyName` ― 外部プロパティの名前
- `value` ― 指定されたプロパティの新しい値

関連項目：″Gateway.″setProperty()

### startServer()

$system.external.startServer() は外部サーバを起動します。

```
startServer(serverName As %String) as %Boolean
```

パラメータ：

- `serverName` ― 既存の外部サーバ定義の名前

関連項目：“外部サーバの起動および停止”

stopServer()

$system.external.stopServer() は外部サーバを停止します。

```
stopServer(serverName As %String, verbose As %Boolean = 0) as %Boolean
```

パラメータ：

- serverName ― 既存の外部サーバ定義の名前
- verbose ― true の場合、現在のデバイスのエントリを表示

関連項目：“外部サーバの起動および停止”

# 6.3 ゲートウェイ・メソッド

Gateway オブジェクトには、同じ名前の $system.external メソッドとまったく同じように動作する一連のメソッドがありますが、最初の引数としてサーバ名を取らない点のみが異なります。例えば、以下の文は、"%Java_Server" を最初の引数として $system.external.new() を呼び出すことで、Java Date() メソッドを呼び出します。

```
set date = $system.external.new("%Java_Server","java.util.Date")
```

以下のほぼ同じ文は、実際には $system.external.getGateway() を使用して Java Gateway オブジェクトを作成し、次に new() の Gateway バージョンの呼び出しを続けます。

```
set date = $system.external.getGateway("%Java Server").new("java.util.Date")
```

この例は、getJavaGateway() で Gateway オブジェクトを取得することで簡素化できます。これには、サーバ名の引数は必要ありません。

```
set date = $system.external.getJavaGateway().new("java.util.Date")
```

これら 3 つの例はすべて機能的に同一です。

## 6.3.1 ゲートウェイ・メソッドの詳細

addToPath()

Gateway..addToPath() は、現在の言語のゲートウェイ・パスに path を追加します。ここで、path 文字列には 1 つ以上の実行可能ファイルへのパスが含まれます。

```
addToPath(path As %RawString)
```

パラメータ：

- path ― 単一のパスを含む文字列、または複数のパスを含む %DynamicArray または %ListOfDataTypes

path 引数には、単一のパスを含む単純な文字列を指定できます (Java の場合、これはフォルダまたは jar URL にできます)。複数のパスを追加するには、追加するパスを含む動的配列または %Library.ListOfDataTypes のインスタンスを渡します。

エラーが発生した場合、この関数は例外をスローします。

関連項目："$system.external."addToPath()

### disconnect()

Gateway.disconnect() は、外部サーバへの接続がある場合にそれを切断します。

```
disconnect()
```

関連項目 : "$system.external."disconnect()

### getExternalLanguage()

Gateway..getExternalLanguage() は、外部サーバからの外部言語を返します。

```
getExternalLanguage() as %String
```

関連項目 : "$system.external."getExternalLanguage()

### getExternalLanguageVersion()

Gateway..getExternalLanguageVersion() は、外部サーバからの外部言語のバージョンを返します。

```
getExternalLanguageVersion() as %String
```

関連項目 : "$system.external."getExternalLanguageVersion()

### getProperty()

Gateway..getProperty() は、外部クラスからの静的プロパティの値を返します。

```
getProperty(externalClass As %String, propertyName As %String) as %ObjectHandle
```

パラメータ :

- `externalClass` ― 外部クラスの名前
- `propertyName` ― 外部プロパティの名前

関連項目 : "$system.external."getProperty()

### Help()

Gateway..Help() は、Gateway メソッドのリストを含む文字列を返します。オプションの method 引数が指定された場合は、そのメソッドについての詳細な情報が返されます。

```
Help(method As %String = "") as %String
```

パラメータ :

- `methodName` ― 詳細な説明が返されるメソッドの名前を含むオプションの文字列

関連項目 : "$system.external."Help()

### invoke()

Gateway..invoke() は、外部コードを呼び出します。このメソッドが式として呼び出されると、これは外部コードによって返される任意の値を返します。

```
invoke(externalClass As %String, externalMethod As %String,     args... As %RawString) as %String
```

パラメータ :

- `externalClass` ― 外部クラスの名前

- externalMethod — 外部メソッドの名前
- args...— 指定されたクラス・コンストラクタに対するゼロ個以上の引数

外部コードによって値が返されない場合は、<COMMAND> 例外がスローされます。externalClass は、外部コードのコンテナ名として渡されることが期待されます。Java または .NET の場合、これはクラス名です。Python の場合、これはクラスまたはモジュールの名前です。externalMethod は、externalClass コンテナ内で呼び出す外部コード単位（関数、メソッド、またはその他の言語固有の単位）の名前です。返り値は、外部コードによって返される値です。外部メソッドが値を返さない場合、invoke() は値を返しません。

関連項目：″$system.external.″invoke()

### isServerRunning()

Gateway.isServerRunning() は、このゲートウェイのサーバが実行中である場合に true を返します。

```
isServerRunning() as %Boolean
```

関連項目：″$system.external.isServerRunning()″、“外部サーバの起動および停止”

### new()

Gateway..new() は、外部クラスのインスタンスに結合される %Net.Remote.Object の新しいインスタンスを返します。externalClass および必要に応じて追加のコンストラクタの引数を渡します。

```
new(externalClass As %String, args... As %RawString) as %Net.Remote.Object
```

パラメータ：

- externalClass — 外部クラスの名前
- args...— 指定されたクラス・コンストラクタに対するゼロ個以上の引数

関連項目：″$system.external.new()″、“ゲートウェイの作成とプロキシ・オブジェクトの使用”

### setProperty()

Gateway..setProperty() は、外部クラス内の静的プロパティの値を設定します。

```
setProperty(externalClass As %String, propertyName As %String, value As %RawString)
```

パラメータ：

- externalClass — 外部クラスの名前
- propertyName — 外部プロパティの名前
- value — 指定されたプロパティの新しい値

関連項目：″$system.external.″setProperty()