Nextra
REST サーバ ユーザガイド

Version 6.5

目　次

第 1 章 はじめに

本書について

本書は、REST API を利用して、ネットワークプロセス位置情報管理や Nextra サーバの利用を考えている、開発者・管理者向けの技術マニュアルです。

対象読者

Nextra REST サーバは、二通りのサービスを提供します。

1. ネットワークプロセス位置情報管理
2. Nextra サーバの REST サービス化

本書は、一方、又は両方のサービスを利用する開発者・管理者を対象にしています。

前提知識

ネットワークプロセス位置情報管理を利用する場合には、前提知識は不要です。

但し、Nextra サーバの REST サービス化に関しては、Nextra を利用したサーバ開発の知識が必要となります。Nextra サーバに関しては、『サーバ開発者ガイド』をご参照下さい。

用語集

用語	定義
Broker（ブローカ）	Nextra が提供する Naming Server（ネーミングサーバ）です。ネットワークプロセスの位置情報（サービス名、ホスト名、ポート番号）をメモリに保持します。
定義ファイル	Nextra/RPC にてクライアント・サーバ間の通信を行う際に必要となる、インタフェース名、関数名、そして関数内で使用されるパラメータとそれぞれのデータ型を定義するテキストファイルです。IDL（Interface Definition Language）とも呼ばれ、ファイル拡張子に

	は.def や.idl と命名することをお勧めします。
RPC (リモート・プロシージャ・コール)	異なるプログラムでプロシージャを起動し、OS やハードウェアの違いによらず、データを交換する関数の呼出しを行う。呼出しシンタックスは、ローカル関数呼び出しと同じです。

表記規則

文中の表記規則

本書で使用する規則を理解しておくと、ユーティリティの使用方法などを容易に理解できます。

形式	説明	例
sub-text	ユーザが指定する必要がある値を示します。	*text*.def
bold	本文中では Nextra ユーティリティを示します。サンプル中では、強調される部分を示します。	**broker**
[brackets]	| がない場合は、オプションテキストを示します。| がある場合は、いずれか 1 つを選択することを示します。	[NONE | ERROR | WARN | DEBUG]

本書で使用するシンボル

本書では、次のようなシンボルを使用しています。

警告メッセージ

このシンボルに続くメッセージに、特別な注意を払う必要があることを示しています。このメッセージには重要な情報が含まれており、この情報を正しく理解してから先に進んでください。

ヒントメッセージ

このシンボルに続く本文は、必須ではありませんが状況に応じて役立つ手順であることを示しています。

オプションメッセージ

このシンボルに続く本文はオプションであることを示しています。内容は、追加機能または代替手法の概要、ある概念を理解するために役立つプロセスステップの詳細などです。

デバッグのヒント

このシンボルに続く本文は、プロジェクトの現在のステップをデバッグする手順が含まれていることを示しています。この方法はあくまで参考であり、別の有効なデバッグ方法の使用を妨げるものではありません。

第 2 章 インストール

Nextra 製品入手には、Github よりスクリプトを入手し Docker イメージ内に作成する方法、又は弊社より製品パッケージを直接入手する方法のいずれかを選択して下さい。

Githubより入手するには、https://github.com/inspire-international/nextra-install-centos7にアクセスし、記述に従い作業を行います。事前に、Docker がインストールされている必要があります。

製品パッケージを入手した場合には、製品同梱インストールノート(Install.txt)を参照に、インストールを行います。JDK1.8 のインストール並びに javac への PATH 設定が必要となります。

起動

Dockerイメージ版

初回は **run.sh** にて起動、2 回目以降は **start.sh** にて起動します。停止の際は、いずれの場合も **stop.sh** を利用します。

設定ファイル

デフォルトでは、コンテナ内にて **broker**(ブローカ)をポート番号 39001 にて、サンプルプログラム(testInterface、testInterface2)2 セットを任意のポートにて起動します。

これらの指定は、以下のファイルに記述されています。

- run.sh/start.sh
- Docker コンテナ内、$ODEDIR/bin/nextra-rest-server.sh。login.sh にて Docker コンテナにログイン出来ます。

必要に応じて上記ファイルを編集して下さい。

パッケージ版

Nextra ネーミングサーバである **broker**(ブローカ)、Nextra サーバを起動します。その後、$ODEDIR/bin(%ODEDIR%\bin)配下の nextra-rest-server.sh(bat)を必要に応じて編集し起動します。

	**定義ファイルはオプションです。** ネットワークプロセス位置情報管理のみを利用する場合には、起動時に定義ファイルを指定する必要がありません。 起動スクリプトである、nextra-rest-server.bat(sh)をエディタで開き、適宜に修正してください。出荷時は、サンプルとして提供される定義ファイルを指定しています。

動作確認

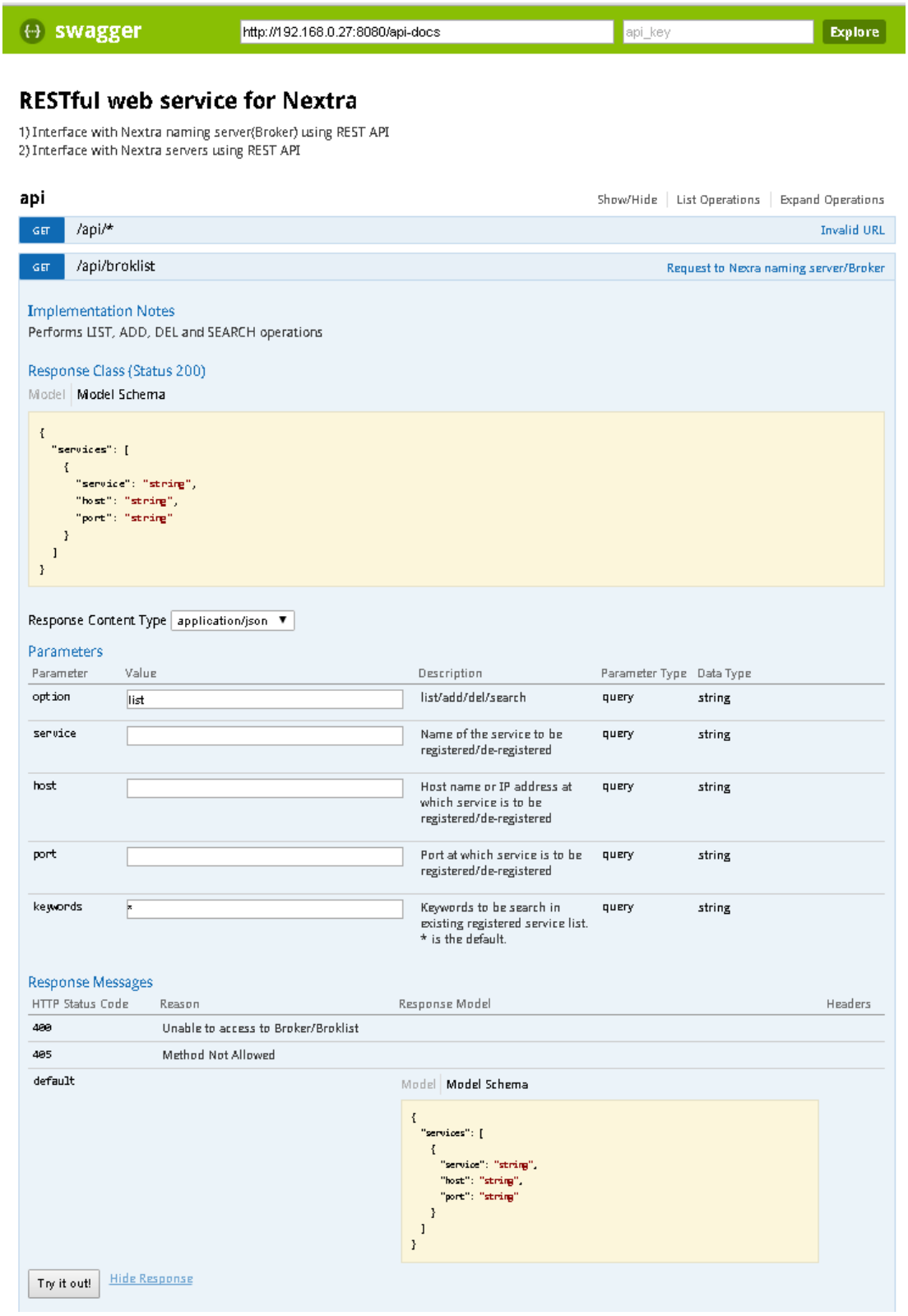
ブラウザにて、`http://x.x.x.x:8080`[1]にアクセスします。 `Swagger`が提供するマニュアル、`REST API`テストインタフェースが表示されます。

	**Swaggerとは？** The goal of Swagger™ is to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined via Swagger, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interfaces have done for lower-level programming, Swagger removes the guesswork in calling the service. 詳細は、http://swagger.io/getting-startedにて。

[1] `Docker` イメージ版の場合、`x.x.x.x` は `Docker` が稼働するホスト `OS` の `IP` アドレス、又はホスト名となります。

第 3 章 ネットワーク プロセス位置情報管理

http://x.x.x.x:8080 にアクセスすると、以下のインタフェースが表示されます



Broker（ブローカ）に対して、list、add、del、search を行えます。記述に従い操作し、JSON 形式にて値が Response Body に表示されることを確認して下さい。

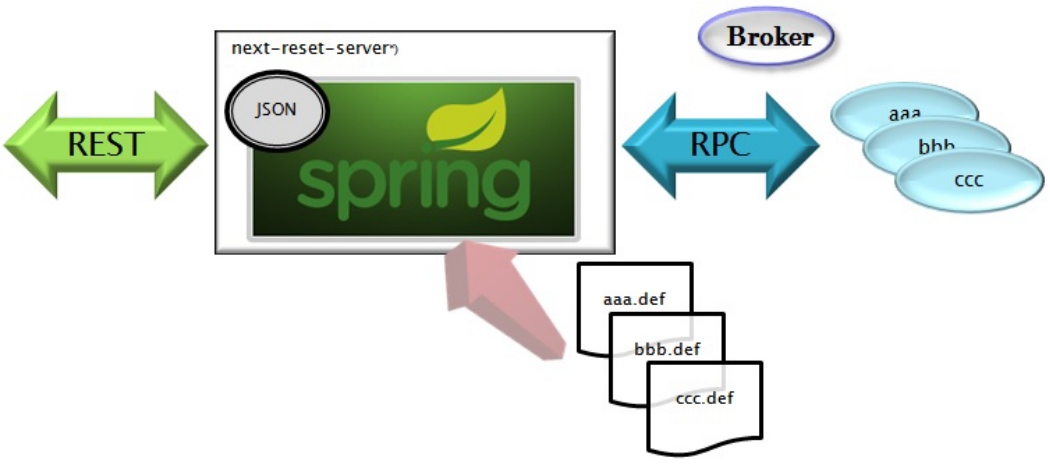
同名サービスの序列について

list、又は search のリクエストを行った際、同名サービスの序列は、毎回ランダムに変更されます。これは、特定のサービスへのリクエストが集中しない、つまりリクエストが分散されることを目的とし、Broker 内部に組込まれたアルゴリズムによるものです。

第 4 章 ネクストラサーバのRESTサービス提供

Nextra(ネクストラ)REST サーバ(nextra-rest-server)は、Nextra サーバに対するプロキシとして動作し、JSON オブジェクトと Nextra/RPC のマッピングを行います。

鳥瞰図

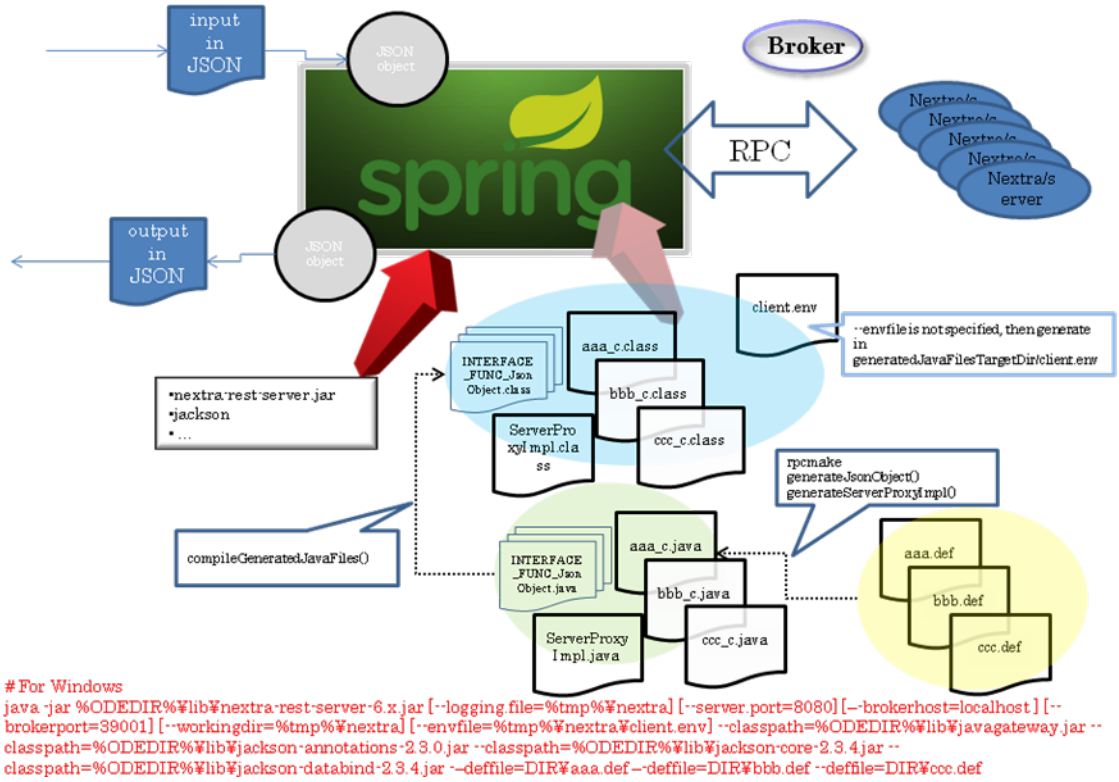


*) java -jar $ODEDIR/lib/nextra-rest-server.jar --deffile=abc.def --deffile=bbb.def --deffile=ccc.def

機能

- Nextra サーバへの一切の変更無しに、REST API による接続を提供します。
- 既存プログラムの再コンパイル不要。Nextra 定義ファイル(.def)ファイルのみで起動します。
- REST API ドキュメントには Swagger を採用。nextra-rest-server 起動後、ドキュメントを参照できます。更に、Nextra サーバに対するテストも同時に行うことができます。

詳細図



内部動作解説

起動時

1. Nextra サーバ毎の定義ファイル(.def)ファイルを入力オプション(--deffile)として指定し起動する。

2. nextra-rest-server は、定義ファイルよりクライアントスタブクラス、Json Bean クラス、プロキシクラスを自動生成。

3. 生成されたクラスをコンパイル。

リクエスト処理時

1. 以下の POST リクエストを受けつける。
/api/server/interface=*INTERFACE_NAME*&function=*FUNCTION_NAME*

2. 対応するクラス群を動的にロード。

3. ネーミングサーバ Broker(ブローカ)にアクセスし、対応する Nextra サーバの位置情報を取得。

4. JSON データを、Nextra/RPC に変換し、Nextra サーバに処理要求。

5. 返り値を JSON データに変換し、クライアントに返却。

サポートされるデータタイプ

インタフェースを定義する定義ファイル(.def/.idl)に記述される以下のデータタイプがサポートされます。

返り値データ型

データ型	定義ファイル内宣言
int	int FUNC_NAME(... ...);
long	long FUNC_NAME(... ...);
void	void FUNC_NAME(... ...);

パラメータ

名称	データ型	定義ファイル内宣言
integer	int	RETURN_DATA_TYPE FUNC_NAME([in] int PARAM_NAME, [out] int PARAME_NAME);
long	long	RETURN_DATA_TYPE FUNC_NAME([in] long PARAM_NAME, [out] long PARAME_NAME);
1 次元固定長配列	char [c] *)c は固定長	RETURN_DATA_TYPE FUNC_NAME([in] char PARAM_NAME[10],

		[out] long PARAME_NAME[10]);
	int[c] *)c は固定長	RETURN_DATA_TYPE FUNC_NAME([in] int PARAM_NAME[10], [out] intPARAME_NAME[10]);
1 次元ヌル終端文字配列	char []	RETURN_DATA_TYPE FUNC_NAME([in] char PARAM_NAME[], [out] char PARAME_NAME[]);
2 次元固定長配列	char [c][c] *)c は固定長	RETURN_DATA_TYPE FUNC_NAME([in] char PARAM_NAME[10][10], [out] char PARAME_NAME[10][10]);
2 次元ヌル終端文字配列	char [][]	RETURN_DATA_TYPE FUNC_NAME([in] char PARAM_NAME[][], [out] char PARAME_NAME[][]);

使用例

`samples/nextra-rest-server/java/standard` を使用した場合。

定義ファイル(test.def)

```
[uuid(00.00.00.00.00.00.00.04) version(1.1)]
interface testInterface {
    int fInt(
        [in]   int    nIn,
        [out]  int    nOut
    );
    long fLong(
        [in]   long   lIn,
        [out]  long   lOut
    );
    void fChaFStr(
        [in]   char   sIn[10],
        [out]  char   sOut[10]
    );
    void fIntFArr(
        [in]   int    nArrIn[10],
        [out]  int    nArrOut[10]
    );
    void fChaNStr(
        [in]   char   sIn[],
        [out]  char   sOut[]
    );
    void fChaFArr(
        [in]   char   sArrIn[10][10],
        [out]  char   sArrOut[10][10]
```

```
    );
    void fChaNArr(
        [in]   char   sArrIn[][],
        [out]  char   sArrOut[][]
    );
};
```

リクエスト並びにレスポンスの例

関数名	Method	Request URL	Request Body	Response Body

fInt	Post	/api/server?interface=testInterface&function=fInt	{
			"nIn" : 12345
			}
				{
				"returnValue": 12345,
				"nIn": 0,
				"nOut": 12345
				}

fLong	Post	/api/server?interface=testInterface&function=fLong	{
			"lIn" : 12345
			}
				{
				"returnValue": 12345,
				"lIn": 0,
				"lOut": 12345
				}

fChaFStr	Post	/api/server?interface=testInterface&function=fChaFStr	{
			"sIn":"①～～"
			}
				{
				"sIn": null,
				"sOut": "①～～"
				}

fIntFArr	Post	/api/server?interface=testInterface&function=fIntFArr	{
			"nArrIn" : [1,2,3,4,5,6,7,8,9,10]
			}
				{
				"nArrIn": [1,2,3,4,5,6,7,8,9,10],
				"nArrOut": [1,2,3,4,5,6,7,8,9,10]
				}

fChaNStr	Post	/api/server?interface=testInterface&function=fChaNStr	{ "sIn":"①～インスパイア～①" 	{ "sIn": null, "sOut": "①～インスパイア

			}	～①" }

fChaFArr	Post	/api/server?interface=testInterface&function=fChaFArr	{
			
			"sArrIn" : [
			
			"①～～",
			
			"②～～",
			
			"Ⅲ～～",
			
			"④～～",
			
			"Ⅴ～～",
			
			"Ⅵ～～",
			
			"Ⅷ～～",
			
			"⑧～～",
			
			"⑨～～",
			
			"Ⅹ～～"]
			
			}
				{
				
				"sArrIn": null,
				
				"sArrOut": [
				
				"①～～",
				
				"②～～",
				
				"Ⅲ～～",
				
				"④～～",
				
				"Ⅴ～～",
				
				"Ⅵ～～",
				
				"Ⅷ～～",
				
				"⑧～～",
				
				"⑨～～",
				
				"Ⅹ～～"]
				
				}

fChaNArr	Post	/api/server?interface=testInterface&function=fChaNArr	{
			
			"sArrIn" : [
			
			"①～～",
			
			"②～～",
			
			"Ⅲ～～"]
			
			}
				{
				
				"sArrIn": null,
				
				"sArrOut": [
				
				"①～～",
				
				"②～～",
				
				"Ⅲ～～"]
				
				}

Curl を利用した場合のリクエスト例

関数名	Curl
fInt	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{ \"nIn\" : 12345 }" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fInt"
fLong	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{ \"lIn\" : 12345 }" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fLong"
fChaFStr	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{ \"sIn\":\"①～～\" }" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fChaFStr"
fIntFArr	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{ \"nArrIn\" : [1,2,3,4,5,6,7,8,9,10] }" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fIntFArr

fChaNStr	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{
	
	\"sIn\":\"①～インスパイア～①\"
	
	}
	
	" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fChaNStr"

fChaFArr	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{ \"sArrIn\" : [\"①～～\",\"②～～\",\"Ⅲ～～\",\"④～～\",\"Ⅴ～～\",\"Ⅵ～～\",\"Ⅶ～～\",\"⑧～～\",\"⑨～～\",\"Ⅹ～～\"] }" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fChaFArr"
fChaNArr	curl -X POST --header "Content-Type: application/json" --header "Accept: application/json;charset=utf-8" -d "{ \"sArrIn\" : [\"①～～\",\"②～～\",\"Ⅲ～～\"] }" "http://192.168.0.27:8080/api/server?interface=testInterface&function=fChaNArr"

ご注意

商標権に関する注意

Nextra 製品は、全て Inspire International の商標または登録商標です。その他記載のブランドおよび製品名は、該当する会社の商標または登録商標です。

著作権に関する注意

Nextra
REST サーバ 利用者ガイド

2018 年 03 月 05 日　マイナー修正
2015 年 11 月 08 日　初版発行

著者　Inspire International Inc.