Magic uniPaaS V1Plus
サーバ構成

初版　　2010 年 8 月 25 日

マジックソフトウェア・ジャパン株式会社

目次

1 はじめに

本書の目的

本書は、Magic uniPaaS サーバ製品を使ってサーバシステムを構成するための理解を深めることが目的です。

Magic uniPaaS サーバを構成する各種ソフトウェアモジュールの設定方法については、製品のリファレンスマニュアルなどに技術情報がありますが、断片的で総花的なので、全体像がわかりにくい、やりたいと思うことを実現するにはどうしたら良いのかがよくわからない、という声を聞きます。

このため、本書では、次のような話題について、QA 形式でまとめて、「やりたいこと」から「どのようにする？」がわかるように工夫してみました。

- Magic uniPaaS 製品を使ってできることとできないこと
- サーバ構成のバリエーション
- それぞれの構成の設定の方法

本書はリファレンスマニュアルに代わるものではありません。情報の詳細さではリファレンスマニュアルが一番ですので、本書を道案内としながら、詳細な情報を調べたい場合にはリファレンスマニュアルをあたってみる、という使い方がベストと思います。

Magic uniPaaS 製品を使ってサーバシステムを開発・保守しようとする人にはぜひ一読していただきたい内容です。

本書で扱わないことがら

本書では、サーバシステムの構築と運用の方法に重点をおいていますので、次のような話題は割愛してあります。

- Magic uniPaaS アプリケーションのプログラミングテクニック: Magic アプリケーションはすでに開発されていることを前提としています。サーバの安定稼働や、機能説明のためにプログラミングに言及することもありますが、プログラミングテクニックそのものについては、深入りしません。他の弊社発行の技術文書、トレーニングコースなどで習得してください。
- Magic uniPaaS 以外のサードパーティソフトウェアに関する説明: サーバシステムでは、Magic uniPaaS 製品以外に、Magic uniPaaS 稼働のプラットフォームである Windows Server オペレーティングシステム、Microsoft 社の Web サーバである Internet Information Service、Magic uniPaaS がアクセスする各種 DBMS（Pervasive, MS-SQL Server、Oracle、DB/400 等）、その他のソフトウェアモジュールが多数存在しますが、他社製品の機能や設定についての説明は基本的に省略しています。それぞれのソフトウェアに関する技術情報や図書は多く入手可能ですので、そちらの方を参照してください。

  ただし、Web サーバなど、uniPaaS サーバの設定と密接な関わりのあるものについては、簡単に説明をしているところもあります。
- デバッグ・トラブルシューティング・パフォーマンスチューニングの方法: 本書の内容を理解することにより、デバッグ・トラブルシューティング・パフォーマンスチューニングがより効率的・的確に行えるようになると思いますが、デバッグ・トラブルシューティング・パフォーマンスチューニングそのものについての

話題は割愛いたしました。

本書の構成

本書は、次のような構成となっています。

- 第 2 章「基本構成」では、全体の概論として、Magic uniPaaS 製品を使って作ることのできるシステムの種類を分類してみました。
- 第 3 章「Web アプリケーションサーバ」では、Magic uniPaaS を使ったサーバシステムのうち、一番基本となる Web アプリケーションサーバシステムについて解説しました。ここで説明した内容は、Web アプリケーションに限らず、他の形態のサーバシステムにも共通する内容がありますので、他の形態のサーバシステムを構築する場合にも、目を通してください。
- 第 4 章「パーティショニングサーバ」は、負荷分散・あるいは効率よい大量バッチ処理のためのパーティショニング機能を実現するための方法について説明します。大量印刷や集計などの処理のために、パーティショニング機能は他の形態のサーバシステムと組み合わせて使われることもよくあるので、大規模なシステムには必須となる機能でしょう。
- 第 5 章「リッチクライアントサーバ」は、最近注目を浴びてきているリッチクライアントシステム（Rich Client, あるいは RIA, Rich Internet Application などとも呼ばれる）を構築するための方法を説明します。
- 第 6 章「共通事項」は、すべての形態のサーバシステムに共通すると思われる話題について集めました。
- 第 7 章「Web サーバの設定の詳細」では、Web サーバ（Internet Information Service, IIS）の設定方法の詳細について解説しました。

全体として、「やりたい、知りたい」から入っていけるように、QA 形式を全面的に採用しました。各 QA は、それぞれで完結するようにしてあります。

話の流れについては、できるだけ最初から自然に読んでいけるように工夫しましたが、本書の性質上、後の章で詳細に解説してある事柄を、前の章で参照していることがままあります。そのような場合には、クロスリファレンスを入れて、すぐに調べられるようにしました。

また、話題に直接関係ない関連事項で深入りする必要はないと思われることがらについては、リファレンスヘルプの項目を参照するようにしました。

2 基本構成

2.1 uniPaaS サーバでどのようなシステムを構築できますか？

次のようなシステムを構築できます。
- Web アプリケーションサーバ
- パーティショニングサーバ
- Web サービス（SOAP）サーバ
- リッチクライアントサーバ

uniPaaS サーバ製品には、エンタープライズサーバ と リッチクライアントサーバ とがあり、実現できるシステムと、ライセンスの考え方が異なります。

リッチクライアントシステムを構築するには、uniPaaS リッチクライアントサーバ 製品を使います。その他のシステムを構築するには、 uniPaaS エンタープライズサーバ製品を使います。

以上をまとめると、以下のようになります。

製品総称	uniPaaS サーバ			
製品名	uniPaaS エンタープライズサーバ			uniPaaS リッチクライアントサーバ
用途	Web アプリケーションサーバ	パーティショニングサーバ	Web サービス（SOAP）サーバ	リッチクライアントサーバ
本書での説明の章	第 3 章	第 4 章	（なし）	第 5 章

これらのシステム形態は、次のような観点で特徴づけることができます。

項目	Web アプリケーションサーバ	パーティショニングサーバ	Web サービスサーバ	リッチクライアントサーバ
クライアントソフト	Web ブラウザ	uniPaaS 実行エンジン	Web サービス(コンシューマ側)をサポートする任意のプログラム	専用クライアントモジュール
クライアントとサーバ間の通信プロトコル	HTTP	独自	SOAP（HTTP 上に実現）	HTTP
リクエストの形式	URL とパラメータ	独自	Web サービス形式（RPC およびドキュメント形式）	独自（XML による）
レスポンスの形式	（一般に）HTML	独自	Web サービス形式	独自（XML による）

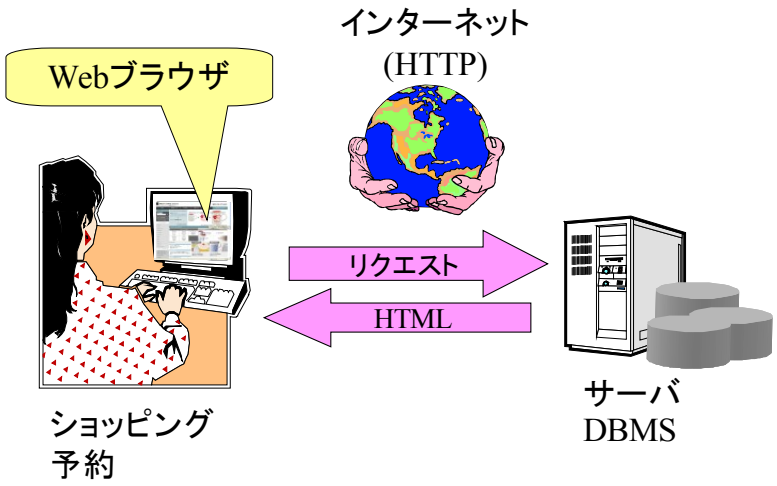
2.2 Web アプリケーションサーバでどんなことができますか？

インターネットでの EC サイトや予約システムのような Web システムを構築できます。

インターネットでの EC サイトや予約システムはすでに一般的に広く使われるようになってきましたが、uniPaaS エンタープライズサーバを使えば、このような Web アプリケーションシステムを構築することができます。

この形態では、次のような構成となります。

- クライアントは Web ブラウザを使います。
- クライアントは、インターネット（TCP/IP）上を、HTTP プロトコルを使って、サーバにアクセスします。
- Web アプリケーションサーバは、クライアントの Web ブラウザからのリクエストを受け、データベースに照会・変更を行い、結果を HTML の形式でクライアントに送り返します。



Web アプリケーションサーバのより詳しい説明は、第 3 章「Web アプリケーションサーバ」にあります。

2.3 パーティショニングサーバでどんなことができますか？

uniPaaS のバッチタスクの処理をサーバ側に代行委託して実行させることができます。

uniPaaS で作成するシステムでは、日次処理・月次処理のように

- 長時間かかる集計・印刷処理
- サーバ側のリソース(高速なプリンタなど)を使う必要がある処理
- データを大量にアクセスする処理 (DBMS に近いところで行う方がよい)

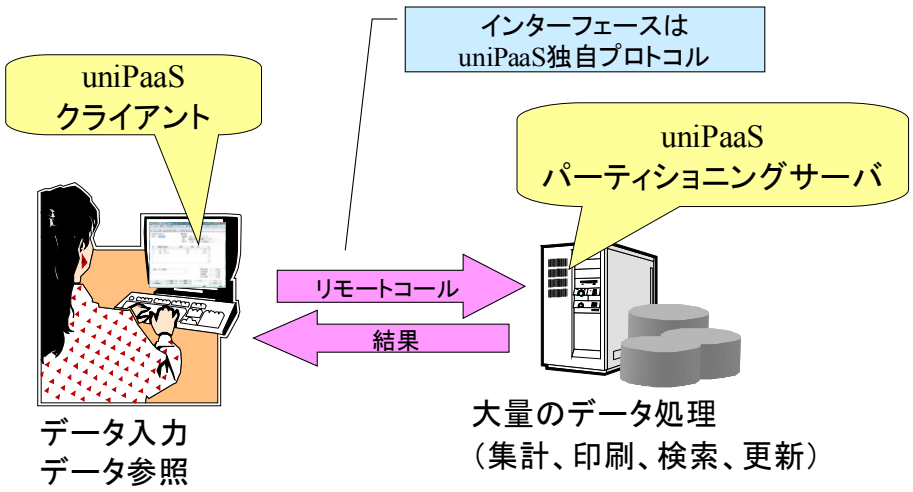
が多くあります。uniPaaS の パーティショニングサーバとしての機能を使えば、このようなバッチタスクをクライアントとは独立して、サーバ側で実行させることができます。

この形態では、次のような構成となります。

- uniPaaS クライアントから、リモートコール コマンドによりリクエストが発行されます。
- パーティショニングサーバは、リクエストを受けて処理を行います。
- 結果をクライアントに返します。

ここで発行されるリクエストは、uniPaaS の内部独自形式によるもので、このため、クライアントは uniPaaS 製品に限られます。Web ブラウザのような汎用的なプログラムから呼び出すことはできません。

クライアント側の uniPaaS 製品としては、uniPaaS Client (クライアントサーバ)の他に、uniPaaS エンタープライズサーバ あるいは uniPaaS リッチクライアントサーバ がリモートコールコマンドを実行して、別の uniPaaS エンタープライズサーバ を呼び出すこともできます。



パーティショニングサーバは、独自のポートとプロトコルを使ってインターフェースを取るので、一般には外部インターネットに公開することはせず、社内 LAN 内で使います。もし、インターネットを通して外部にサービスを公開したい場合には、2.4 節で説明する「Web サービス(SOAP) サーバ」として構成するのが適切です。

パーティショニングサーバには、実行のタイプとして同期と非同期の 2 種類があります。

- 同期処理では、クライアントがサーバ側でのバッチ処理の終了を待ちます。バッチの処理結果が必要な場合に、このタイプを使います。
- 非同期 処理では、クライアントがサーバ側でのバッチ処理の終了を待たずに、次の処理に進みます。クライアントとサーバとで、処理が同時並行して進むので、長時間かかるバッチ処理であっても、クライアント側は終了を待たずに次の処理を続けることができます。バッチ処理の結果は、クライアントには返って来ません。

長時間バッチの場合には、非同期を使うのが便利です。

パーティショニングサーバについての説明は、第 4 章「パーティショニングサーバ」にあります。

2.4 Web サービス（SOAP）サーバでどんなことができますか？

uniPaaS のバッチタスクの処理をサーバ側に代行委託して実行させることができます。

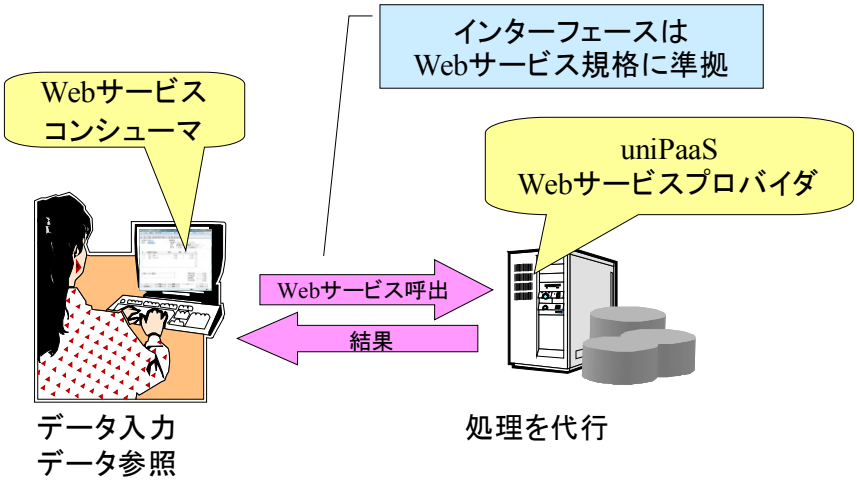
Web サービス（SOAP）は、分散システムにおける相互接続性を実現するためのオープンな技術です。

Web サービスには、サービスを提供するサーバ(プロバイダと呼びます)と、サービスを利用するクライアント(コンシューマと呼びます)とがあり、uniPaaS はそのいずれにも対応しています。

uniPaaS エンタープライズサーバの持つ Web サービスサーバ機能を用いれば、パーティショニングサーバと同様に、uniPaaS のバッチタスクをサーバ側で実行させることができます。

パーティショニングサーバと Web サービスサーバとが異なる点は、パーティショニングサーバが uniPaaS 同士を接続する独自プロトコルを使う技術であるのに対し、Web サービスはクライアントを選びません。

uniPaaS を Web サービスプロバイダとして構成した場合、コンシューマは uniPaaS に限りません。Web サービスの規格に準拠したインターフェースを利用する限り、どんなシステムでも uniPaaS の Web サービスサーバを呼び出すことができます。



Web サービスについては多くの話題があり、本書ではカバーしきれない内容ですので、本書では Web サービスサーバの構成については割愛いたしますが、サーバの構成方法についての基本的な考え方は、Web アプリケーションの場合とほぼ同じに考えることができます。

2.5 リッチクライアントサーバでどんなことができますか？

クライアントサーバのような操作性と、サーバ集中管理による TCO 軽減の良いとこ取りをしたシステムを構成できます。

リッチクライアントという言葉は、SaaS やクラウドが注目を浴びる今日では必須の技術として多くの実現方法がありますが、uniPaaS でもリッチクライアントを実現することができます。

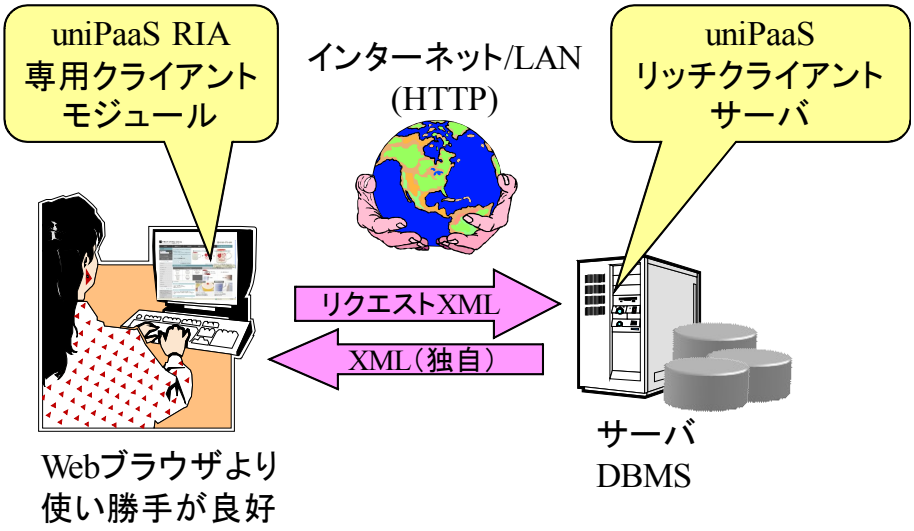
リッチクライアントの特徴を簡単に言えば、クライアントサーバシステムのような操作性の高さと、Web システムのようなサーバ集中管理による利便性の向上や TCO の削減などを併せ持つ、良いとこ取りのシステムであると言えます。すなわち、次のような特徴を持ちます。

- 操作性：豊富なコントロール種類、軽快な動作、細かなカーソルコントロール、IME コントロールなど。
- サーバ集中管理の利便性：ゼロからの自動インストール。アップデート時に自動更新。

一般にリッチクライアントシステムを開発するのは、クライアントサーバシステムの開発よりも倍以上の工数がかかり、躊躇してしまうことも多いのですが、uniPaaS のリッチクライアント機能を使えば、クライアントサーバ並の生産性と保守性を実現することができます。

uniPaaS のリッチクライアントシステムにおいては、次のような構成となります。

- クライアント側では小さな uniPaaS リッチクライアント専用のクライアントモジュールを実行します。
- クライアントは TCP/IP (HTTP)を介して、サーバと接続します。
- サーバ側では uniPaaS リッチクライアントサーバ を実行します。
- この両者が連携しながら処理を進めて行きます。



uniPaaS のリッチクライアント技術については、別の弊社技術資料などで詳しく紹介されていますので、本書ではリッチクライアントそのものについての詳細は省略し、第 5 章「リッチクライアントサーバ」で、サーバシステムの構成にフォーカスして説明いたします。

3 Web アプリケーションサーバ

本章では、サーバシステムのうちで一番単純であり、基本形となる Web アプリケーションサーバの構成について説明します。

本章で説明する内容の多くは、パーティショニングサーバやリッチクライアントサーバシステムでもそのまま応用できる内容です。

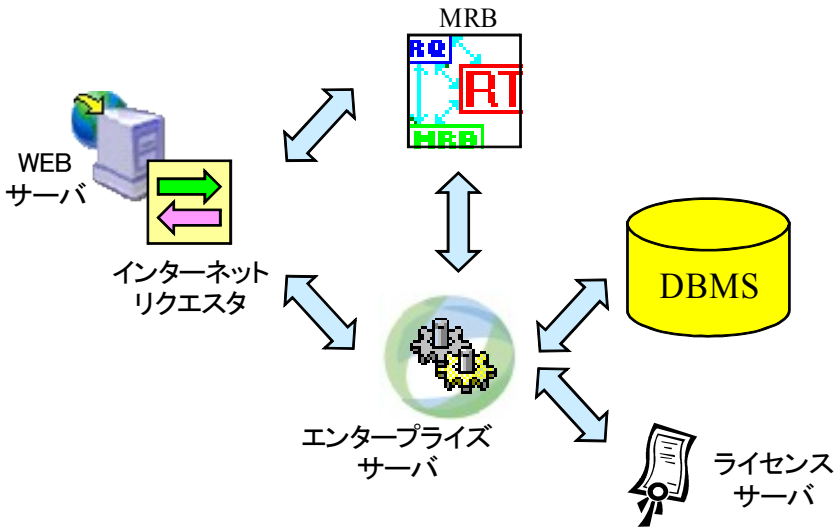
パーティショニングサーバやリッチクライアントサーバシステムに固有な話題については、次章以降に、本章に補足する形で説明していきます。

3.1 Web システムにおけるソフトウェアモジュールとそれぞれの機能は？

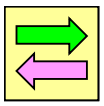
Web システムは、以下のものからなります。

- Web サーバ
- インターネットリクエスタ
- エンタープライズサーバ
- リクエストブローカ
- ライセンスサーバ

これらのモジュールの関連を図示すると、下図のようになります。



それぞれの機能は以下のようになります。

図	名称	機能
	**Web サーバ**	クライアントからのリクエストを受け取る窓口となるモジュールです。Windows プラットフォーム上では通常 Internet Information Service (IIS) が使われますが、Apache も使われることがあります。
	**インターネットリクエスタ**	Web サーバと uniPaaS サーバの橋渡し役をするモジュールで、uniPaaS エンタープライズサーバ の一部として提供され、インストーラにより IIS に組み込まれるようセットアップされます。 インターネットリクエスタは Web サーバから起動され、Web サーバが受けたリクエストを、uniPaaS サーバへのリクエストとして変換します。エンタープライズサーバがリクエストを処理した後は、作成されたレスポンスデータを、Web サーバに返します。
	**リクエストブローカ (MRB)**	**(Magic Request Broker、MRB と略称します)** エンタープライズサーバへのリクエストの交通整理役となるモジュールで、サーバシステムの構成に大きな自由度と堅牢性とを実現します。

	**エンタープライズサーバ**	uniPaaS アプリケーション（ECF ファイル)を実行する実行エンジンで、uniPaaS エンタープライズサーバ 製品の中核となるモジュールです。
	**ライセンスサーバ**	エンタープライズサーバのライセンスを管理します。

3.2 各モジュールは、どのファイルに相当しますか？

それぞれ、以下のモジュールとなります。

エンタープライズサーバ	uniRTE.exe
インターネットリクエスタ	Scripts¥mgrqispi018.dll、および Scripts¥mgrqcgi018.exe
MRB	uniRQBroker.exe
ライセンスサーバ	c:¥FlexLM¥LMGRD.EXE

上記のうち、ライセンスサーバだけは c:¥FlexLM ディレクトリ（固定）にあります。それ以外は、uniPaaS エンタープライズサーバをインストールしたディレクトリにあります。

インターネットリクエスタは2種類あり、mgrqispi018.dll は IIS がサポートする ISAPI インターフェースにより起動されます。一方、mgrqcgi018.exe は、CGI インターフェースにより起動されます。

ISAPI インターフェースの方がオーバーヘッドが少なく、IIS を利用する場合にはこちらを利用することをお勧めします。インストーラのデフォルト設定では、IIS がインストールされていることを検出したら、mgrqispi018.dll だけをインストールします。

CGI インターフェースを使う mgrqcgi018.exe は、Windows 上で IIS 以外の Web サーバ（Apache など）を使う場合に利用します。

バージョンによる違い：インターネットリクエスタモジュールの「018」という数字は、uniPaaS のバージョン番号を示します。インターネットリクエスタは、MRB やエンタープライズサーバのバージョンを合わせておかないといけないので、バージョン番号をつけて、混同しないようになっています。

このため、uniPaaS のバージョンが異なると、インターネットリクエスタのモジュール名のバージョン番号のところが異なります。例えば、uniPaaS V1（1.5SP1）では、mgrqispi015.dll となっています。

その他、モジュール名もバージョンにより異なっています。以下にまとめます。

製品バージョン	eDeveloper V10	uniPaaS V1	uniPaaS V1Plus
エンタープライズサーバ	eDevRTE.exe	uniRTE.exe	uniRTE.exe
インターネットリクエスタ	mgrqispi101.dll mgrqcgi101.exe	mgrqispi015.dll mgrqcgi015.exe	mgrqispi018.dll mgrqcgi018.exe
ライセンスサーバ	c:¥FlexLM¥LMGRD.EXE		

3.3 Web アプリケーションでのリクエストはどのような形式ですか？

HTTP の GET あるいは POST で、インターネットリクエスタの URL とパラメータを指定します。

Web システムでのクライアントは Web ブラウザであり、Web ブラウザは Web サーバと HTTP プロトコルを使って通信します。

このため、Web システムにおいて、クライアントが発行するリクエストは、URL を指定して GET/POST する形となります。

3.3.1 GET を使った例

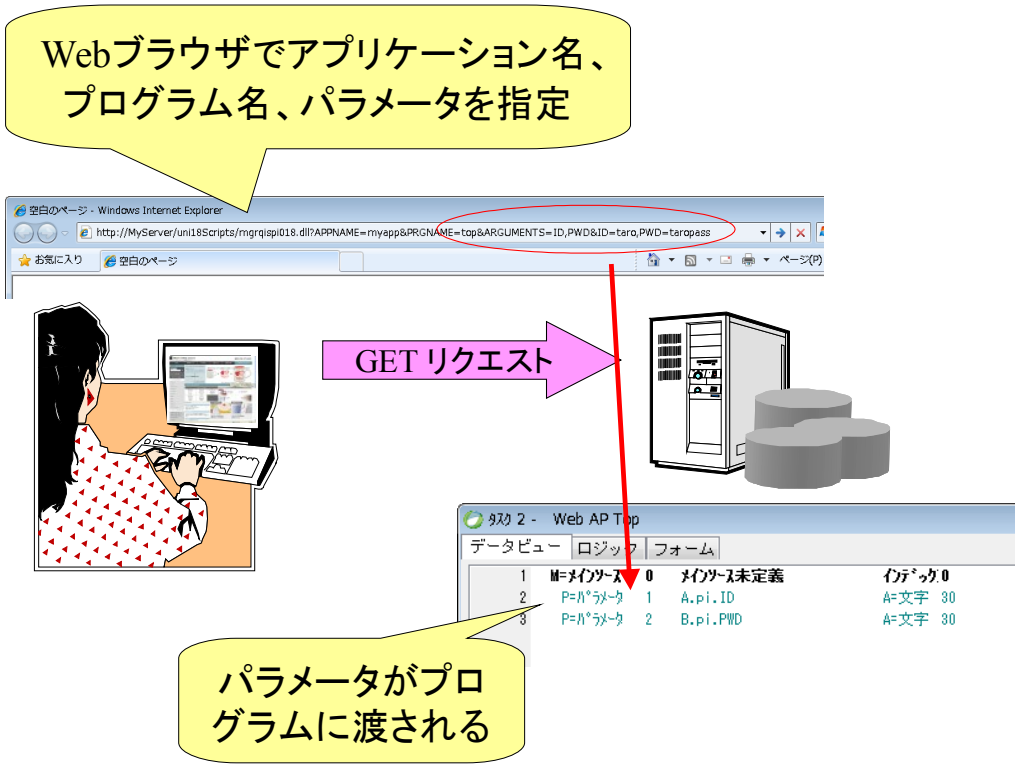
GET では、URL の中にインターネットリクエスタの URL とパラメータとを続けて書きます。(実際には 1 行で指定します)

URL:

```
http://MyServer/uni18Scripts/mgrqispi018.dll?
  APPNAME=myapp&PRGNAME=top&ARGUMENTS=ID,PWD&ID=taro,PWD=taropass
```

上例で、MyServer は Web サーバのホスト名、/uni18Scripts/mgrqispi018.dll は インターネットリクエスタの URL、「？」以下の部分は、インターネットリクエスタに渡すパラメータとなります。ここでは、myapp という名前のアプリケーションの中にある、top という公開名を持ったプログラムを起動することを要求しています。このときのパラメータは、ID と PWD の二つがあり、それぞれ、taro、taropass という値が指定されています。

アプリケーションのプログラム top では、この二つのパラメータを受け取れるように、文字型のパラメータ変数を二つ定義しておきます。



3.3.2 POSTを使った例

POST では、インターネットリクエスタの URL だけを指定し、パラメータは POST のデータとして、Web サーバに送ります。

URL:	http://MyServer/uni18Scripts/mgrqispi018.dll
POST データ:	APPNAME=myapp&PRGNAME=top&ARGUMENTS=ID,PWD&ID=taro,PWD=taropass

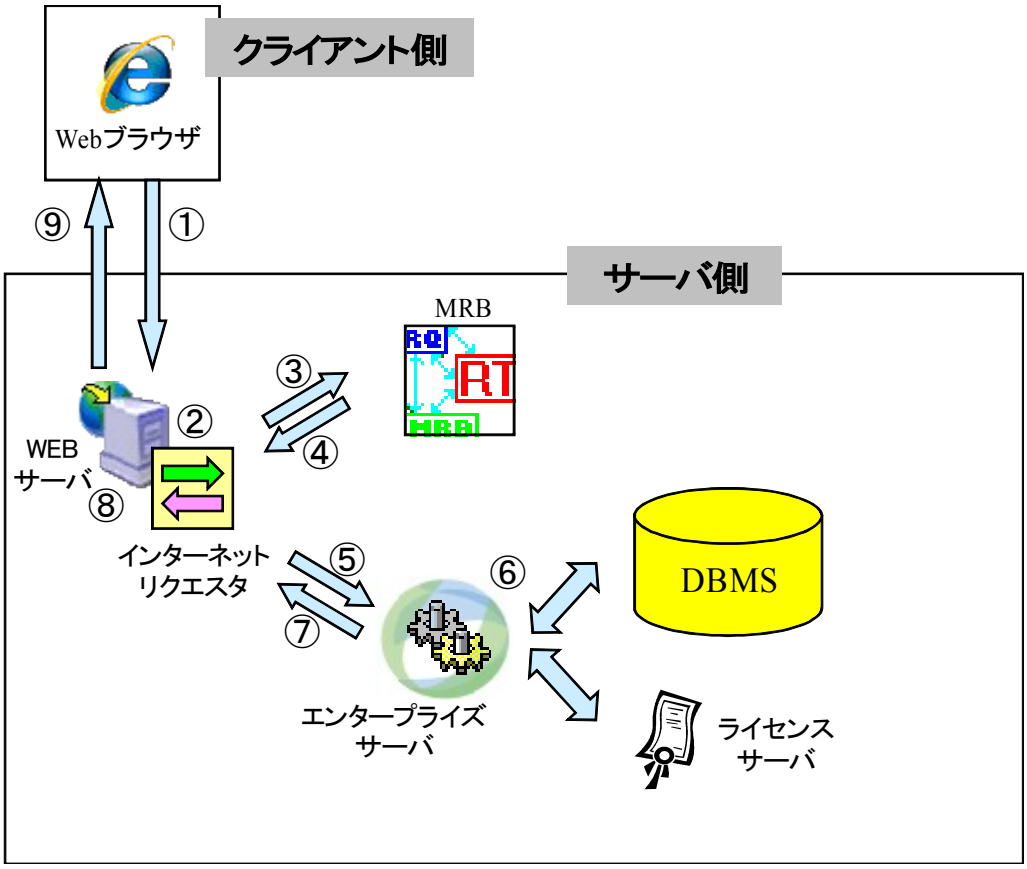
上例では、インターネットリクエスタの URL は GET の場合と同じです。パラメータは、URL には記述せず、POST データとして GET の場合と同様の形式で送信します。

3.4 リクエストの処理の流れは？

クライアントの Web ブラウザ → Web サーバ → インターネットリクエスタ → MRB → エンタープライズサーバ → DBMS の順で処理が流れます。

3.4.1 通常のリクエスト処理の流れ

一連のリクエスト処理の流れを図示すると、下図のようになります。



クライアントの Web ブラウザから、Web サーバに HTTP リクエスト（GET あるいは POST）が発行されると、次のような順序でリクエストが処理され、結果が Web ブラウザに返されます。

1. Web ブラウザから、Web サーバに HTTP リクエストが発行されます。
2. HTTP リクエストを受けた Web サーバは、ISAPI あるいは CGI インターフェースを通してインターネットリクエスタを呼び出します。

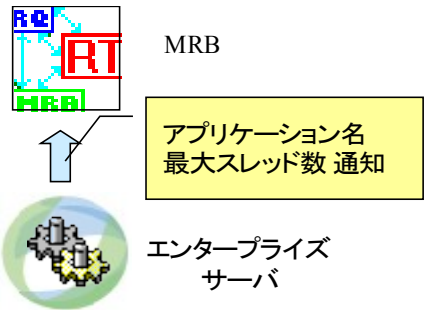
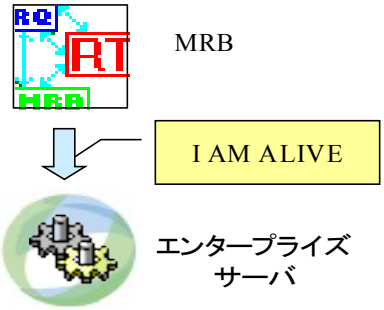
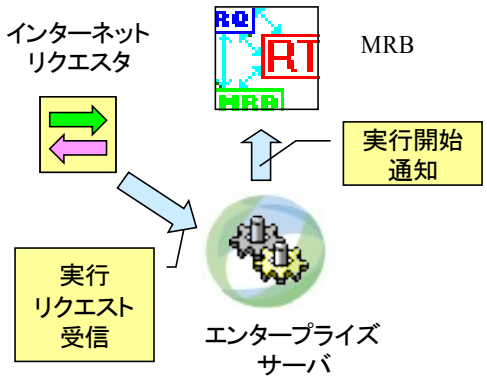
   インターネットリクエスタは、HTTP リクエストのデータ（アプリケーション名、公開プログラム名、パラメータ等）を Web サーバから受け取り、解析して、uniPaaS の内部形式に変換します。
3. インターネットリクエスタは MRB にリクエストを送ります。
4. MRB は、エンタープライズサーバの IP アドレスとポート番号とをインターネットリクエスタに通知します。
5. インターネットリクエスタは、MRB より通知された IP アドレスをポート番号を使って、エンタープライズサーバに直接接続し、リクエストのデータを送信します。
6. エンタープライズサーバは、受け取ったリクエストをもとに、指定された公開プログラムのプログラムを実行します。このとき、パラメータもプログラムに渡します。

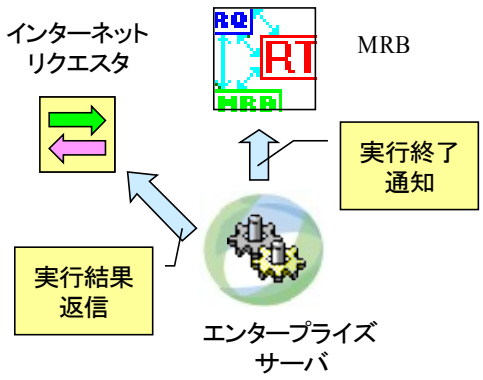
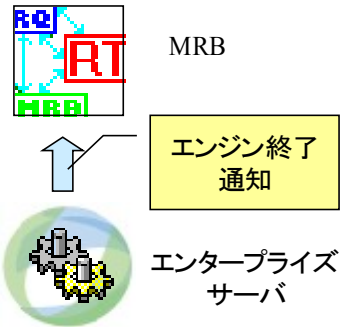
7. プログラムが終了したら、プログラムの出力結果（「入出力ファイル」テーブルで、「メディア」が「R=リクエスタ」である IO ファイルに出力したデータ）を、インターネットリクエスタに結果として返送します。
8. インターネットリクエスタは、Web サーバに結果を返します。
9. Web サーバはクライアントの Web ブラウザに結果を返します。

これで一連のリクエスト処理が終わります。

3.4.2 管理用情報の流れ

以上は、リクエストとその結果の流れのみを書きましたが、実際には、この他に、サーバの状態管理のための通信が、下記のように背後で行われています。

エンジン起動時：uniPaaS サーバ起動時には、オープンしているアプリケーション名、同時実行可能な最大スレッド数（リッチクライアントサーバの場合には、最大ユーザ数）などの情報を含むメッセージが、MRB に送られます。MRB はこのメッセージを受け取り、サーバプールにサーバを登録します。	
起動中、定期的に：MRB は、登録されている uniPaaS サーバが動作可能な状態になっているかを確認するため、KEEP-ALIVE メッセージを定期的に uniPaaS サーバに送ります。	
リクエスト開始時：MRB は常に、登録された uniPaaS サーバの状態を記録しています。上記、リクエスト処理のステップ 6（uniPaaS サーバが、指定された公開プログラム名のプログラムを実行する）では、uniPaaS サーバはプログラム実行開始のメッセージを MRB に送ります。	

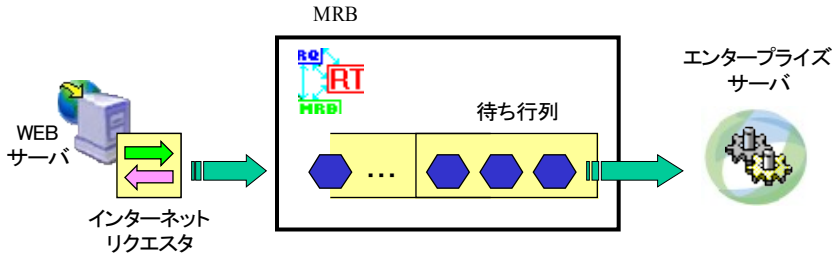
リクエスト終了時：プログラムの実行が終了して結果をインターネットリクエスタに送信し終わったら、今度は実行終了のメッセージを MRB に送ります。	
エンジン終了時：uniPaaS サーバが終了するときには、uniPaaS サーバは MRB に対して、サーバ終了のメッセージを送信します。MRB はそれを受けて、サーバプールからサーバを削除します。	

3.5 処理が追いつかない場合、リクエストはどうなりますか？

MRB の待ち行列に入れられ、FIFO（先入り先出し）で処理されていきます。

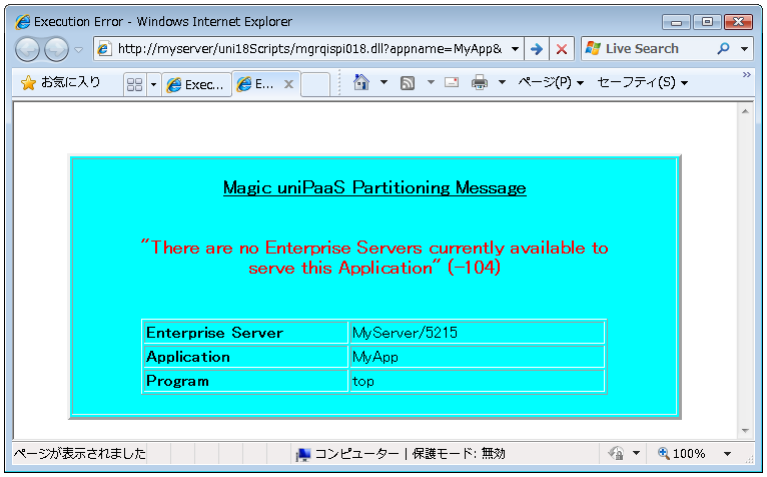
リクエストが短時間に大量に到着した場合、エンタープライズサーバのリクエストの処理が追いつかなくなることがあります。

MRB にはリクエストの待ち行列が用意されているので、このような場合には、後から来たリクエストは待ち行列に格納され、エンタープライズサーバの処理が終了したら FIFO(先入り先出し)で順番に処理されて行きます。



無制限に待ち続けることを防ぐため、待ち行列にはタイムアウト時間が設定されています。タイムアウトは、デフォルトで 10 秒ですが、設定ファイル MGREQ.INI で変更することもできます。(「6.16.3 リクエスタタイムアウト」参照)

タイムアウト時間内に順番が回ってこなかった場合には、リクエストは失敗となり、エラーステータス APP-IN-USE (-104) がリクエスト元に返されます。



Web アプリケーションの場合には、クライアントの Web ブラウザに、エラーメッセージが表示され、時間内にエンタープライズサーバで処理しきれなかったことがわかるようになっています。エラーメッセージはデフォルトのものをインターネットリクエスタが HTML として作成しますが、カスタマイズすることも可能です。

3.6 モジュール間の関係は、どこで定義しますか？

MGRB.INI、MAGIC.INI、MGREQ.INI などの設定ファイルで定義します。

uniPaaS のサーバシステムにおけるモジュール間の通信は TCP/IP を使って行いますので、モジュール間の通信は、ホスト名とポート番号を指定することになります。

3.6.1 ブローカ ポート番号

まず、設定の基礎となるのは、MRB の窓口となる、TCP/IP でのポート番号であり、これはブローカ ポート番号と呼びます。

ブローカ ポート番号は、リクエストブローカ (uniRQBroker.exe) があるのと同じディレクトリにある MGRB.INI ファイルの BrokerPort パラメータで設定します。

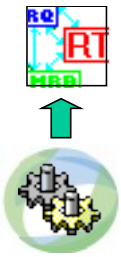
MGRB.INI 例

```
[MRB_ENV]
BrokerPort = 5215
PasswordSupervisor = password
```

この例では、ブローカ ポートは 5215 となります。

PasswordSupervisor というのは、エンタープライズサーバがリクエストブローカに接続する際に指定するパスワードで、エンタープライズサーバが不正にリクエストブローカに接続して、リクエストを横取りすることを防ぐためのものです。

3.6.2 エンタープライズサーバと MRB の間の設定



エンタープライズサーバは、起動した後、自分自身を MRB に登録し、リクエストを処理可能であることを伝達します。

エンタープライズサーバがどの MRB に登録するかは、MAGIC.INI ファイルの [MAGIC_SERVERS] セクションと、MessagingServer パラメータとで決定されます。

これらのパラメータは、「サーバ」テーブルと、動作環境→アプリケーションサーバ タブの「メッセージサーバ」パラメータとに対応します。

(1) MAGIC.INI での設定

MAGIC.INI での設定例

```
MessagingServer = Default Broker
[MAGIC_SERVERS]
Default Broker = 0,MRBServerName/5215,,password,10,,1
```

[MAGIC_SERVERS] セクションは、動作環境の「サーバ」テーブルに対応するもので、ここには複数のブローカ情報を登録することができます。上記の例では、「Default Broker」という名前で、MRBServerName というホスト名のサーバ上で、ブローカ ポート 5215 番で待ち受けているリクエストブローカが定義されています。

MessagingServer パラメータは、[MAGIC_SERVERS] セクションに登録されている MRB のうち、どれを利用する

のかを選択します。上記の例では「Default Broker」という名前で登録されている MRB を選択しています。

(2) サーバパスワード

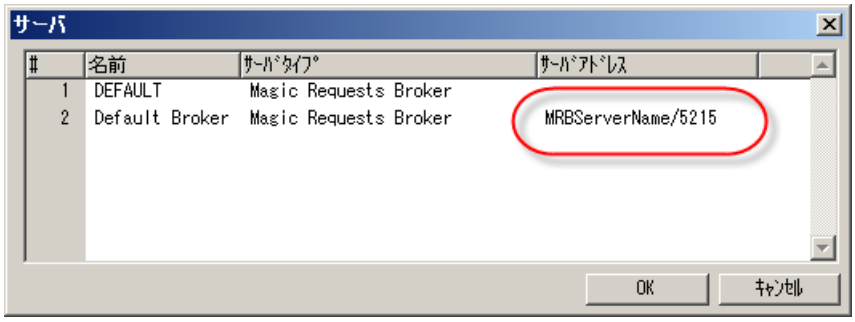
[MAGIC_SERVERS] セクションでは、MGRB.INI で指定されているのと同じパスワード password を指定する必要があります。もしパスワードが間違っていると、MRB はエンタープライズサーバの登録を拒否します。これはなりすましの不正なエンタープライズサーバが登録され、リクエストが横取りされるのを防ぐためのものです。

(3) エンタープライズサーバでの設定

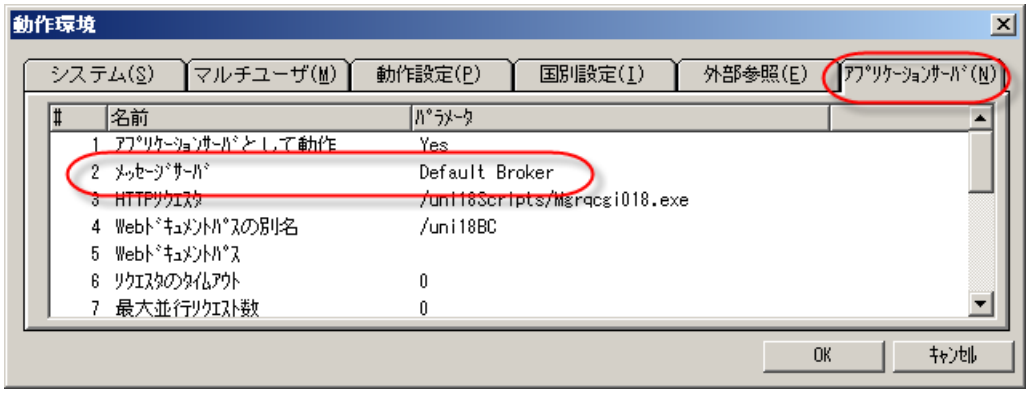
エンタープライズサーバをデスクトップから起動すると、フォアグラウンドで起動され、各種設定パラメータをテーブルで設定することができます。

上記の設定は、以下の「サーバ」テーブルと、「動作環境→アプリケーションサーバ」タブと同じになります。

「サーバ」テーブル

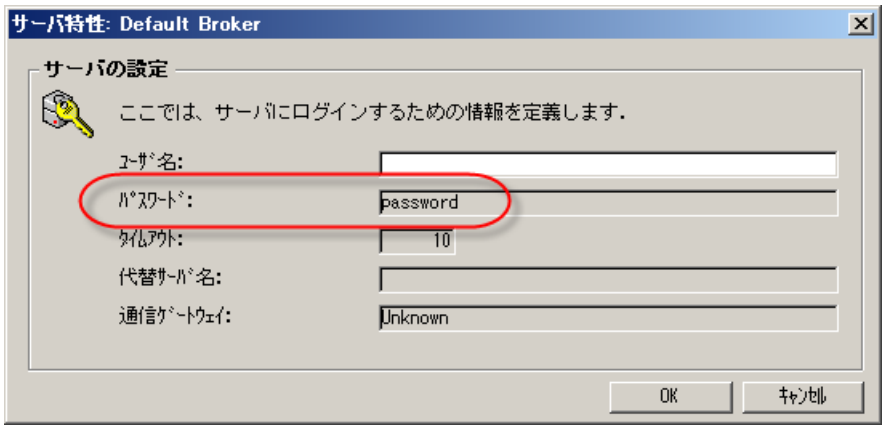


「動作環境→アプリケーションサーバ」タブ



サーバパスワードは、サーバテーブルから Alt+Enter でサーバ特性を開いて設定することができます。

サーバ特性→パスワード



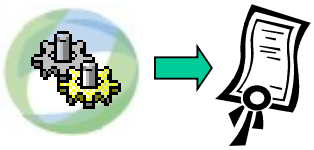
(4)　MRB サーバ名の省略

もし、エンタープライズサーバと MRB とが同じサーバで実行されている場合には、MRB のサーバ名を省略し、ブローカ ポート番号だけを指定するだけでも OK です。

MRB とエンタープライズサーバが同一マシン上にある場合の MAGIC.INI 例

```
MessagingServer = Default Broker
[MAGIC_SERVERS]
Default Broker = 0,5215,,password,10,,1
```

3.6.3　エンタープライズサーバ とライセンスサーバの間



エンタープライズサーバは、許可されたライセンスの範囲内で利用することができます。ライセンスを管理しているのがライセンスサーバです。

エンタープライズサーバとライセンスサーバの関係は、エンタープライズサーバの MAGIC.INI ファイルの LicenseFile と LicenseName パラメータにより定義されます。

MAGIC.INI の例

```
LicenseFile = 744@MyLicenseServerName
LicenseName = MGENT1P1
```

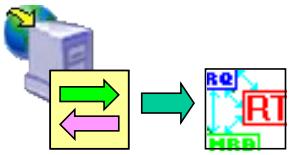
この例では、MyLicenseServerName というホスト名をもつサーバにライセンスサーバがある場合の設定を示しています。

ライセンス名は MGENT1P1 となっています。これは uniPaaS V1Plus におけるエンタープライズサーバを利用する際のライセンス名です。

ライセンス名は、uniPaaS のバージョンや、実行している製品等により異なりますので、取得されたライセンスのライセンス証書に指定されたライセンス名を記述してください。

なお、ライセンスサーバがエンタープライズサーバと同じサーバ上で実行されている場合でも、LicenseFile にはホスト名を省略することはできません。ホスト名を明示的に指定するか、あるいは localhost を指定します。

インターネットリクエスタ と MRB の関係



インターネットリクエスタは、Web サーバの実行可能な仮想ディレクトリに配置されます。uniPaaS V1Plus (V1.8SPx) では、/uni18Scripts という名前の仮想ディレクトリにあります。この仮想ディレクトリは、uniPaaS をインストールしたディレクトリの下の Scripts というサブディレクトリに対応するように、インストーラが設定します。

Web サーバがクライアントよりリクエストを受け取り、インターネットリクエスタを起動したとき、インターネットリクエスタは Web サーバより受け取った情報を uniPaaS エンタープライズサーバの内部形式に変換して、MRB にリクエストを転送します。このときに転送する MRB の定義をするのは、インターネットリクエスタと同じディレクトリにある MGREQ.INI ファイルの、MessagingServer パラメータです。ここには、MRB のあるサーバのホスト名と、ブローカ ポートとを指定します。

MGREQ.INI 例 (リクエスタのあるディレクトリ上)

```
[REQUESTER_ENV]
MessagingServer = MRBServerName/5215
```

この例では、MRBServerName というホスト名のサーバ上で、ポート番号 5215 で待ち受けている MRB に対して、リクエストを送ることを指定しています。

Web サーバと MRB とが同じサーバ上で実行されているときには、MessagingServer のホスト名を省略して、ブローカポート番号だけを指定することもできます。

MRB と同一マシン上にある場合の MGREQ.INI 例

```
[REQUESTER_ENV]
MessagingServer = 5215
```

ここで見るように、各モジュール間は TCP/IP で通信するので、モジュールの配置の構成に大きな自由度があります。これにより、配備負荷分散・耐障害性の必要要件に応じ、スケーラブルな構成を選択することが可能になります。

3.7 エンタープライズサーバが実行するアプリケーションはどこで指定しますか？

MAGIC.INI の StartApplication で、アプリケーションのキャビネットファイル（.ECF ファイル）を指定します。

エンタープライズサーバで実行するアプリケーションの指定は、MAGIC.INI の StartApplication パラメータで、アプリケーションのキャビネットファイル （ECF ファイル）の名前を指定することにより行います。

MAGIC.INI 例

```
StartApplication = c:¥Program Files¥uniPaaS¥EnterpriseServer¥Projects¥MyApp¥MyApp.ecf
```

この例では、指定された ECF ファイルが、エンタープライズサーバ起動時に自動的にオープンされて、実行開始されることを指定しています。

3.8 実行する ECF ファイルを、コマンドラインパラメータとして渡すことはできますか？

/StartApplication=（ECF ファイル名）とします。

StartApplication に限らず、MAGIC.INI に指定できるパラメータに一般的なことですが、コマンドラインパラメータとして、/（パラメータ名）=（パラメータ値） を指定することにより、MAGIC.INI での設定より優先して指定することができます。

StartApplication の場合には ECF ファイルを指定するのですが、ECF ファイル名にはパスの区切り文字である「￥」記号が入ることがあり、この記号はエスケープ文字として使われるので、この指定のしかたに注意が必要で、次のいずれかの方法により指定します。

- ディレクトリ名の区切り文字「￥」は、二重にして「￥￥」として指定する。

```
/StartApplication=c:¥¥Program Files¥¥uniPaaS¥¥EnterpriseServer¥¥Projects¥¥MyApp¥¥MyApp.ecf
```

- 「＝」の代わりに「＝＊」を使う。この場合には「￥」がエスケープ文字として解釈されなくなります。

```
/StartApplication=*c:¥Program Files¥uniPaaS¥EnterpriseServer¥Projects¥MyApp¥MyApp.ecf
```

この表記法は、StartApplication に限らず、パラメータ値中に「￥」記号やカンマ「、」などを含む場合に共通な方法です。

3.9 複数のアプリケーションを同時に実行させることはできますか？

複数のインスタンスを起動し、それぞれの StartApplication を指定することにより可能です。

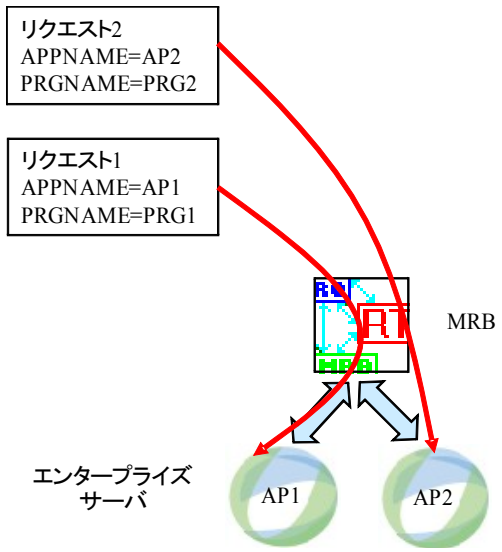
エンタープライズサーバは、一時には一つのアプリケーションしか実行することはできません。複数のアプリケーションを同時に実行したい場合には、複数のエンタープライズサーバを立ち上げ、マルチインスタンス構成（3.16「マルチインスタンス構成の設定は？」参照）とする必要があります。

この場合、各インスタンスが起動するアプリケーションの ECF ファイルを別々に指定します。これは、MAGIC.INI を分けて StartApplication パラメータにそれぞれの ECF ファイルを指定してもよいし、あるいはコマンドラインパラメータとして /StartApplication=（ECF ファイル名）を指定することもできます。

右図は、アプリケーション AP1 と AP2 とを、二つのエンタープライズサーバのインスタンスで別々に実行している様子を図示したものです。

APPNAME として AP1 を指定したリクエスト 1 は、左側にあるい、アプリケーション AP1 を実行しているインスタンスで実行され、AP2 を指定したリクエスト 2 は、右側のインスタンスで実行されます。

どちらのインスタンスで実行するかは、MRB が判断して、指定されたアプリケーションを実行しているインスタンスを判別します。



どうしても1インスタンスのみで複数のアプリケーションを混在させたいという場合は（あえてそのようにする必要性はあまりないとは思いますが）、次のような方法が考えられます：

- 複数アプリケーションをコンポーネント化して、両方を包み込むプロジェクトを定義し、それを実行します。（次節 3.10「複数のアプリケーションをまとめてひとつにすることはできますか？」参照）。
- 「アプリケーションオープン」イベントを利用して、アプリケーションを明示的に切り替えて実行させます。（性能面・使い勝手の面から、非推奨です）

3.10 複数のアプリケーションをまとめてひとつにすることはできますか？

アプリケーションをコンポーネント化して実現します。

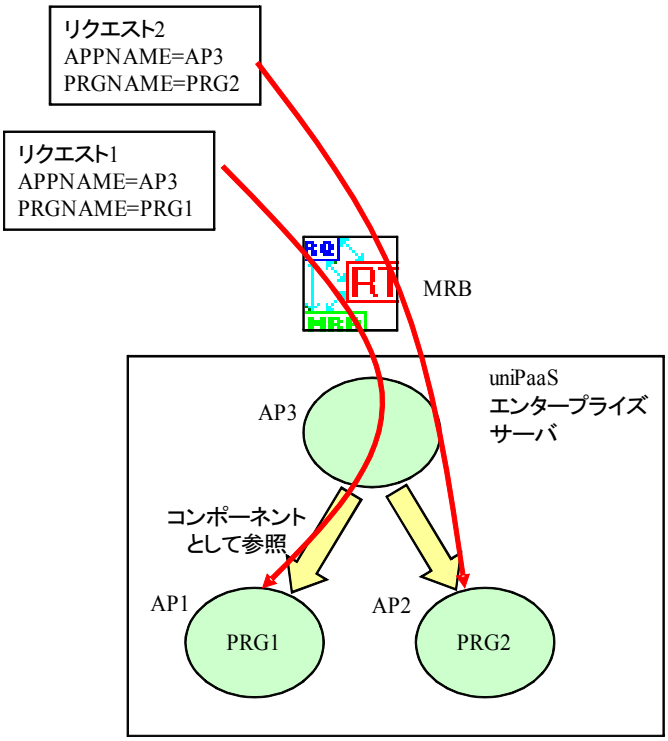
アプリケーションの管理上の便利さのために、別々のプロジェクトとして開発したアプリケーションを、実行時にはひとつのアプリケーションとして（つまり、共通のアプリケーションを持ったものとして）実行したい場合があります。これは、アプリケーションをコンポーネント化することにより実現します。

例えば、右図はプロジェクト AP1 と AP2 を、ひとつのアプリケーション（AP3 という名前）にまとめる例です。

- プロジェクト AP1 にプログラム(公開名)PRG1、プロジェクト AP2 にプログラム PRG2 があります。
- プロジェクト AP3 は、プロジェクト AP1 と AP2 とをコンポーネントとして参照しています。
- エンタープライズサーバは、プロジェクト AP3 をオープンして実行します。

このような設定とすると、PRG1 および PRG2 は、次のようにして呼び出せます。

- コンポーネント AP1 と AP2 は、AP3 の一部となって動作するので、リクエストの APPNAME には、いずれの場合にも AP3 を指定します。
- PRGNAME として PRG1 を指定すると、プロジェクト AP1 の PRG1 が実行され、PRG2 を指定すると、プロジェクト AP2 にある PRG2 が実行されます。

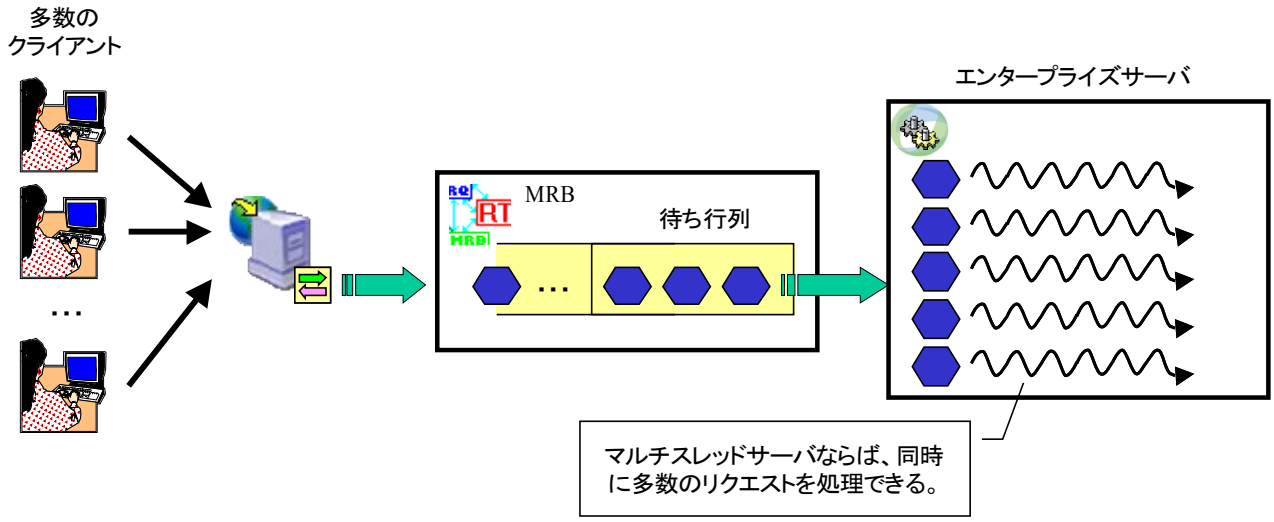


3.11 エンタープライズサーバの「スレッド」とは何ですか？

各インスタンスにおいて同時実行可能なタスクの最大数です。

エンタープライズサーバはマルチスレッドエンジンとして、同時並行的に複数のリクエストを処理することができます。

リクエストが多数来る場合には、一つずつ処理していたのでは追いつかないことがありますが、同時並行処理することにより、レスポンス時間を改善することができます。



エンタープライズサーバのライセンスは同時実行可能なスレッドの最大数に対して課されます。例えば、15スレッドのライセンスならば、同時に15個までのリクエストを並行して処理することができます。

スレッド数は、運用時のクライアント数とは直接関係しませんが、ユーザ数が多いほど、また処理内容が重くなるほど、多くのスレッド数が必要となります。一般に、大規模システムほど、多くのスレッド数が必要になります。

一般的には、スレッド数が多いほど、同時に実行するリクエストが多くなるので、全体の性能が向上しますが、1台のサーバで実行している限り、ハードウェアとして処理できる限度がありますから、ある程度のところまで行くと性能向上の効果が飽和してきます。具体的に何スレッドくらいで飽和するのかは、個々のアプリケーションの作り方やシステムの使われ方、ハードウェアのスペックに大きく依存することですので、一概には言えず、ケースバイケースで実測して決定する必要があります。

飽和状態になったならば、マルチサーバ構成（3.17「マルチサーバ構成の設定は？」参照）とすることにより、ハードウェア台数分の性能向上を実現することができます。

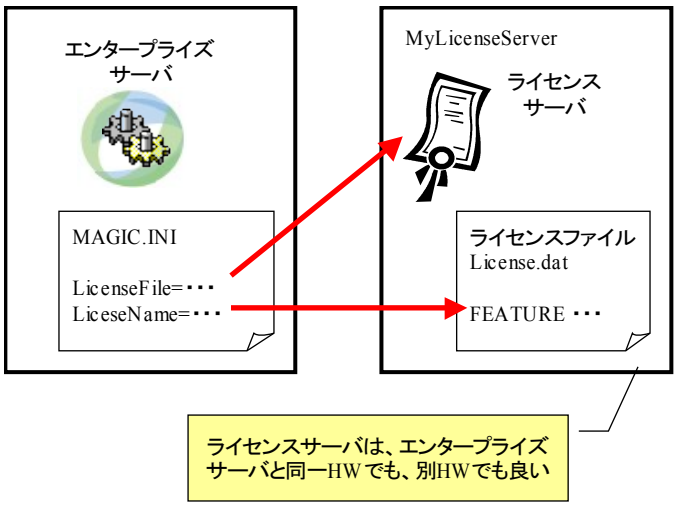
3.12 ライセンスはどのように管理されますか？

ライセンスサーバにチェックアウト、チェックインすることにより管理されています。

3.12.1 ライセンスサーバ

uniPaaS サーバ製品のライセンスは、ライセンスサーバというソフトウェアモジュールにより管理されています。ライセンスサーバは、uniPaaS サーバ製品をインストールすると、同時にインストールされます。また、個別にインストールすることもできます。具体的なインストール方法は、インストレーションガイドを参照してください。

下図は、ライセンスサーバを MyLicenseServer という名前のサーバ HW にインストールして、エンタープライズサーバから参照する場合を示しています。ここでは別サーバ HW にライセンスサーバを置いていますが、エンタープライズサーバと同一サーバ HW にインストールすることもできます。



3.12.2 ライセンスファイル

ライセンスの定義は、ライセンスファイル license.dat で定義されます。これは ライセンスサーバをインストールしたマシンの c:¥FlexLM ディレクトリにあります。

エンタープライズサーバのライセンスは、FEATURE 名が MGENT1P1 という名前で登録されています。この名前は、uniPaaS 製品の種別およびバージョンにより変わります。以下に、uniPaaS エンタープライズサーバライセンス例を示します。

license.dat

```
FEATURE MGENT1P1 MAGIC 1.800 01-jan-0 10 123ABC123DEF123BCD89 \
        VENDOR_STRING=PT=MGENT1,C=3FFFFFF,P=S,M=0,SSL=Y OVERDRAFT=0 \
        DUP_GROUP=NONE ISSUER=MAGICUSER ck=188 SN=190000001
```

3.12.3 サーバでの設定

エンタープライズサーバでは、MAGIC.INI の LicenseFile と LicenseName とで、ライセンスサーバとライセンス名とを指定します。

MAGIC.INI 例

```
LicenseFile = 744@MyLicenseServer
LicenseName = MGENT1P1
MaxConcurrentRequests = 5
```

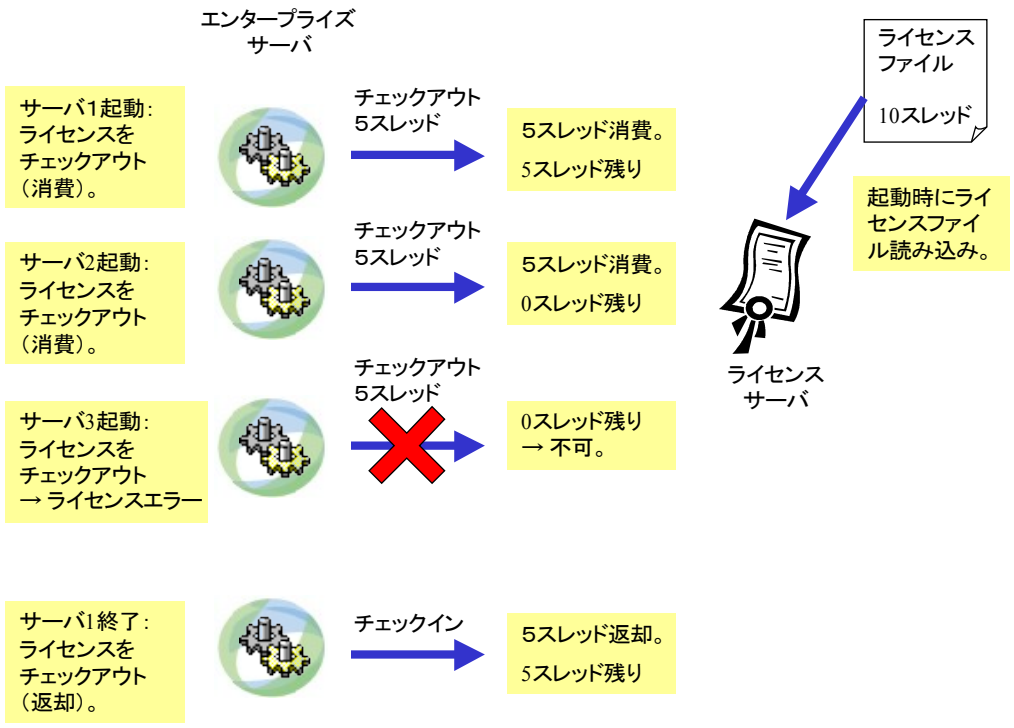
MAGIC.INI にはこの他に、このエンタープライズサーバに割り当てたい最大スレッド数を MaxConcurrentRequets パラメータにより指定します。上記の例では 5 スレッドが割り当てられます。

- リッチクライアントサーバの場合には、ライセンスは同時実行ユーザ数で管理されます。MAGIC.INI には、スレッド数ではなく、各リッチクライアントサーバに割り当てるユーザ数として、MaxConcurrentUsers パラメータで設定します。
- MaxConcurrentRequests/MaxConcurrentUsers のデフォルト値は 0 です。0 というのは、「無制限」という意味で、その時点でライセンスサーバに残っているライセンスをすべて消費します。

3.12.4 チェックアウト/チェックイン

ライセンス数の管理は、ライセンスサーバへのチェックアウト/チェックインにより行われます。

下図は、10 スレッドのライセンスがある場合に、エンタープライズサーバ が 5 スレッドづつライセンスを消費する様子を示しています。



ライセンスサーバは起動時に、ライセンスファイルを読み込みます。ここで、合計で 10 スレッドが許可されていることを認識します。

エンタープライズサーバ 1 が起動すると、ライセンスサーバに5スレッドライセンスのチェックアウトを要求します。これにより、ライセンスサーバは5スレッドが消費されたことを認識し、残りは１０ － ５ ＝ ５スレッドとなります。

次に同様に、エンタープライズサーバ2が起動すると、ライセンスサーバに5スレッドライセンスのチェックアウトを要求します。これにより、ライセンスサーバはライセンスが消費されたことを認識します。この時点で、残りスレッド数は0となります。

この状態のときに、別のエンタープライズサーバ3がライセンスのチェックアウトを要求しても、ライセンスオーバーで拒否されます。エンタープライズサーバ3は、ライセンスエラーで起動できません。

エンタープライズサーバが終了するときには、ライセンスサーバにライセンスのチェックインを要求します。これにより、ライセンスが解放され、ライセンスサーバは再び、ライセンスのチェックアウトを受け付けることができる状態になります。

3.13 エンタープライズサーバではどのような構成が可能ですか？

代表的な構成として、次のようなものがあります。

- オールインワン
- Web サーバのみ別 HW
- マルチインスタンス・マルチサーバ
- 代替ブローカによる二重化
- ロードバランサによる多重化

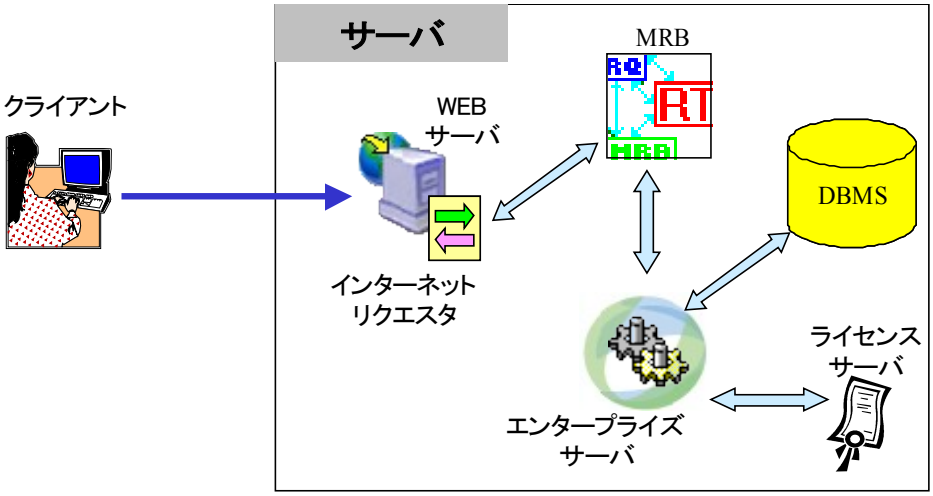
エンタープライズサーバシステムにおけるモジュール間は、TCP/IP で接続されているので、HW 構成やインスタンス数などの設定に非常に大きな自由度があります。

インターネットに公開するシステムでは、様々な攻撃から守るためのセキュリティの考慮が非常に重要で、Web サーバだけを DMZ に置く構成がよく使われます。この場合には、Web サーバのみを別 HW に配置する構成となります。この場合には、uniPaaS の設定の他に、ファイアーウォールの設定も必要になります（6.13「ファイアーウォールの設定方法は？」参照）。

3.14 オールインワンの場合の設定は？

全モジュールを同一 HW 上に置きます。サーバホスト名は指定省略可能。

もっとも簡単な構成としては、Web サーバ、MRB、エンタープライズサーバまですべて1台のサーバに配置する構成があります。ここでは「オールインワン」と呼びます。



DBMS は小規模なものなら同じサーバにインストールすることもできますが、DBMS は負荷が大きいので、オールインワン構成でも、別途 DB サーバを立てるのが一般的です。

エンタープライズサーバでは、同一サーバ上にある MRB を指定するので、MRB のホスト名は省略し、ブローカポート番号 だけを指定します。もちろん、ホスト名を明示的に指定してもかまいません。

MAGIC.INI

```
MessagingServer = Default Broker
[MAGIC_SERVERS]
Default Broker = 0,5215,,password,10,,1
```

インターネットリクエスタのあるディレクトリ上の MGREQ.INI では、同様に、MRB のブローカ ポート番号だけを指定します。これにより、インターネットリクエスタは同一サーバ上の MRB にリクエストを発行します。

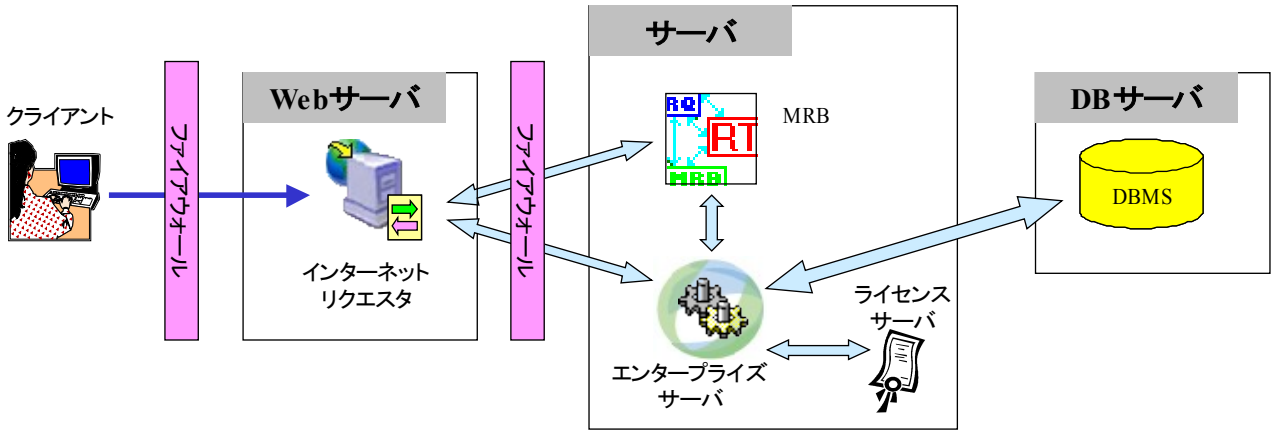
MGREQ.INI (リクエスタのあるディレクトリ上)

```
[REQUESTER_ENV]
MessagingServer = 5215
```

3.15 Web サーバのみ別 HW にしたい場合には？

仮想ディレクトリを手作業で設定し、インターネットリクエスタのみを Web サーバに配置します。
DMZ を作る場合には、ファイアーウォールの設定も必要となります。

インターネットからの不正な攻撃に対応するため、Web サーバは DMZ に置いて、インターネットとイントラネットとを二つのファイアーウォールで分離する方法がよく採られます。下図は、uniPaaS サーバシステムにおいて、この構成を採用した場合の基本的な構成図です。



この場合には、前節で説明した設定の他に、次の設定を行う必要があります。

- Web サーバ上での仮想ディレクトリの設定：これについては、第 7 章「Web サーバの設定の詳細」を参照してください。
- ファイアーウォールの設定：これについては、6.13「ファイアーウォールの設定方法は？」を参照してください。
- インターネットリクエスタの MGREQ.INI で、MessagingServer に MRB のサーバ名を指定します。

  MGREQ.INI（リクエスタのあるディレクトリ上）

```
[REQUESTER_ENV]
MessagingServer = MRBServerName/5215
```

3.16 マルチインスタンス構成の設定は？

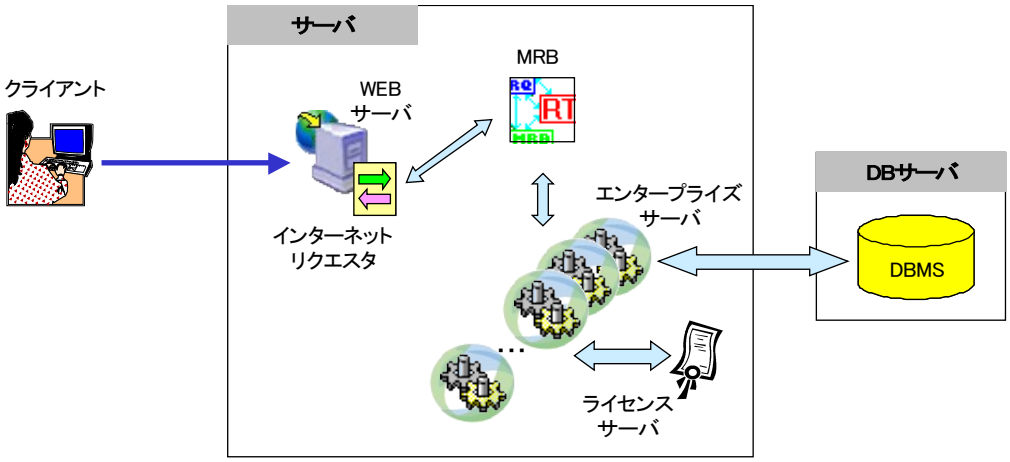
同一の MRB を指定して、複数のエンタープライズサーバを起動します。

uniPaaS のエンタープライズサーバシステムでは、一つの MRB を指定して、複数のエンタープライズサーバを起動することができます。これを「マルチインスタンス構成」と呼びます。

ここで言う「インスタンス」とは、エンタープライズサーバのモジュール uniRTE.exe のプロセスのことを言います。すなわち、マルチインスタンス構成では、複数の uniRTE.exe プロセスが起動することになります。

最大いくつまでのインスタンスを起動可能かについては、uniPaaS の動作方式自体には制限はありません。ライセンスとサーバ HW のキャパシティにより制限されます。

同一 HW 上でマルチインスタンスを起動すると、レスポンスの向上はあまり期待できませんが、可用性を高めることができます。すなわち、万一インスタンスの一つが何らかの原因で異常終了やハングアップした場合にも、別のインスタンスが正常に実行しているので、システムとして稼働を続けることができます。単一インスタンスだけの構成では、そのインスタンスが動作を停止すると、システム全体が止まってしまうことになります。



同一 HW 上で、エンタープライズサーバをマルチインスタンス立ち上げる構成では、uniPaaS の設定は、オールインワンの場合と同じになります。異なるところは、uniRTE.exe が複数起動する、ということです。

3.16.1 ライセンス分割

マルチインスタンス構成にする場合には、ライセンスの分割を適切に行うことが必要です。

デフォルトの設定では、エンタープライズサーバが起動するとき、ライセンスサーバに接続して、ライセンスで許可されているスレッド数のすべてをチェックアウトします。この状態でマルチインスタンスを起動しようとすると、最初のインスタンスでライセンスのすべてを消費してしまい、二つ目以降のインスタンスではライセンスを取得することができず、そのまま終了してしまいます。

これを防ぐためには、インスタンス毎に、消費するライセンスを明示的に設定する必要があります。

各インスタンスあたりのスレッド数は、MAGIC.INI の MaxConcurrentRequests パラメータで設定します。

MAGIC.INI

```
MaxConcurrentRequests = (スレッド数)
```

例えば、ライセンスで15スレッドが許可されている場合に、1 インスタンスあたりには 5 スレッド割り当てて、3 インスタンス起動するときには、以下のように指定します。

```
MaxConcurrentRequests = 5
```

このように設定すると、最初のインスタンスが起動した時には、ライセンスサーバから 5 スレッドだけチェックアウトします。その後、2 番目・3 番目のインスタンスでもそれぞれ 5 インスタンスチェックアウトするので、合計して 15 スレッドを消費することになります。

ライセンス数を分割する場合、すべてのインスタンスで均等に割り当てる必要はありません。インスタンス毎に異なった MaxConcurrentRequests 値を設定することにより、異なったスレッド数を割り当てることが可能です。

いずれにしても、スレッド数の合計が、ライセンスで許容されているスレッド数以下になることが必要です。

MaxConcurrentRequests が 0（デフォルト）の場合には、ライセンスサーバに問い合わせして、現時点でチェックアウト可能な残スレッド数をすべて割り当てます。

3.16.2 複数アプリケーションの実行

複数のインスタンスを起動する場合、インスタンス毎に異なるアプリケーションを実行することができます。これにより、同時に異なるアプリケーションを実行させることができます。

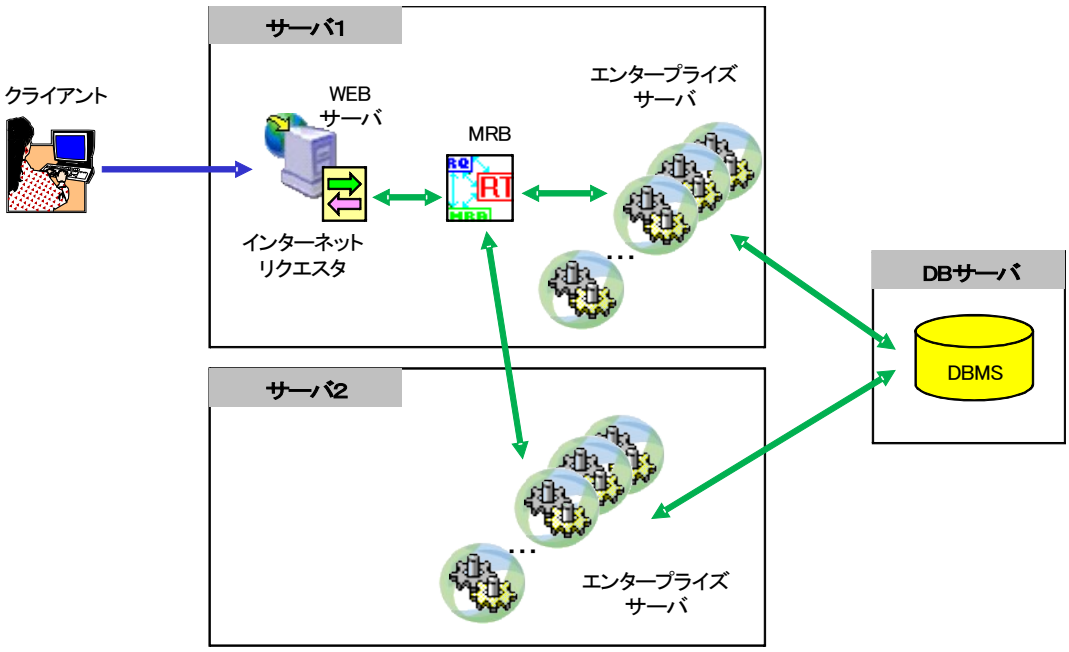
エンタープライズサーバが実行するアプリケーションは、MAGIC.INI の StartApplication パラメータで、ECF ファイルを指定しますので、インスタンスごとに異なった StartApplication を指定します。

異なるアプリケーションを実行させる時には、アプリケーションの負荷の重さにより、スレッド数の割り当てを考慮することも検討すべきでしょう。一般には、リクエスト数の多いアプリケーション、時間のかかる処理が多いアプリケーションを実行するインスタンスには、多めのスレッド数を割り当てておくのが適当でしょう。

3.17 マルチサーバ構成の設定は？

同一の MRB を指定して、複数のサーバ上で複数のエンタープライズサーバを起動します。

別 HW 上でマルチインスタンスを実行するのは、レスポンスを改善することが大きな目的です。単純に言えば、1台の HW で実行するよりも、2台の HW で実行する方が、レスポンスが2倍になります。もちろん、同一 HW 上で実行する場合と同様な可用性向上も達成できます。



別 HW 上で、エンタープライズサーバを立ち上げる場合には、MRB の指定に、ホスト名を明示する必要があります。

MAGIC.INI 例

```
MessagingServer = Default Broker
[MAGIC_SERVERS]
Default Broker = 0,ServerName1/5215,,password,10,,1
```

ここで、ServerName1 というのは、MRB が実行しているサーバ（上図ではサーバ1）のホスト名です。

上図では、サーバは2台しか書かれていませんが、同様の設定で 3 台目、4 台目のサーバを増設することも可能です。この柔軟性が、uniPaaS サーバシステムのスケーラビリティを実現しています。

マルチサーバ構成の場合も、マルチインスタンス構成の一種ですので、前節で説明したような、ライセンス分割の考慮が必要です。

3.18 マルチインスタンス構成の場合ブローカはリクエストをどう振り分けますか？

リクエストに指定されているアプリケーション名と、エンタープライズサーバの負荷状況（実行中のスレッド数、過去の処理時間）により配分します。

MRB がリクエストを受け取ると、次の要素を考慮して、リクエストを実行するためのエンタープライズサーバが選択されます。

- アプリケーション名：リクエストに指定されているアプリケーションを実行しているエンタープライズサーバを選びます。指定されたアプリケーションを実行しているエンタープライズサーバが一つもない場合には、エラーとなります。
- 実行中のスレッド数：指定されたアプリケーションを実行しているエンタープライズサーバが複数ある場合には、それぞれの実行中のスレッド数が少ないものを選びます。これにより、負荷が全エンタープライズサーバに均等に割り当てられ、特定のエンタープライズサーバに偏らないようにします。
- リクエストのレスポンス時間：それでも決まらない場合には、過去のリクエストの処理にかかった時間などを考慮して、なるべく負荷が分散するように決定します。

コンテキストを使っている場合には、そのコンテキストの存在するアプリケーションサーバを選択します。コンテキストについては、5.2「リッチクライアントサーバでの「コンテキスト」とは何ですか？」および 6.15「エンタープライズサーバには、「コンテキスト」はないのですか？」を参照してください。

3.19 代替ブローカによる二重化

MRB とエンタープライズサーバを2系統（標準系、待機系）準備しておき、インターネットリクエスタに代替ブローカを指定します。

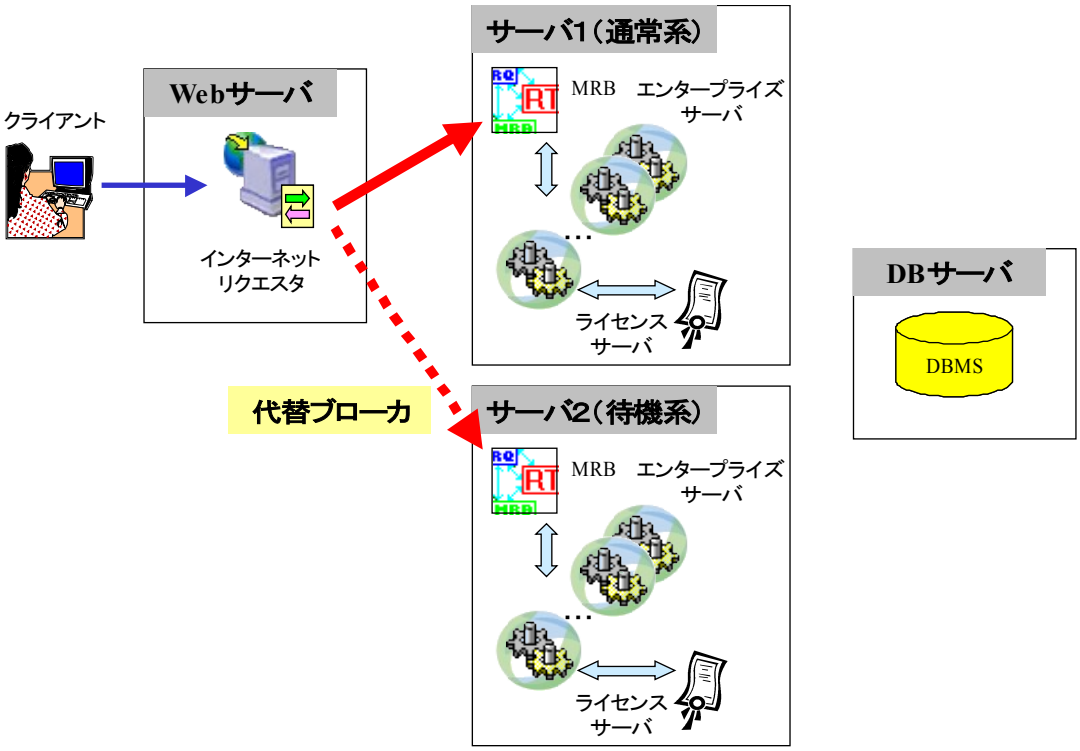
ミッションクリティカルなシステムの場合、可用性をできるだけ高めておくことが重要になります。可用性向上の基本は冗長構成（多重化）で、SPOF（single point of failure、システム上のあるコンポーネントが異常を来たすと、そのシステム全体が障害に陥ってしまうようなコンポーネント）を極力なくすことが必要です。

uniPaaS エンタープライズサーバでは、マルチインスタンス構成とすることにより、エンタープライズサーバを多重化することが可能ですが、MRB はすべてのリクエストの窓口であり、ここが SPOF となっています。

MRB は非常に安定したソフトウェアモジュールで、動作停止する可能性は低いですが、万一のことを考えて、「代替ブローカ」の機能を用いて、二重化の構成にすることも可能です。

3.19.1 代替ブローカの構成

次のように構成します。



- MRB → エンタープライズサーバ の構成を 2 系統用意します。一方が通常系であり、他方は待機系となります。
- インターネットリクエスタが参照する MGREQ.INI（インターネットリクエスタと同じディレクトリにあるもの）で、次のように AltMessagingServer パラメータで、待機系の MRB を指定します。

MGREQ.INI

```
AltMessagingServer = （代替ブローカホスト名）/（ブローカポート番号）
```

3.19.2 代替ブローカの動作

このようにすると、MRB に障害があったときに、次のように待機系に切り替えます。

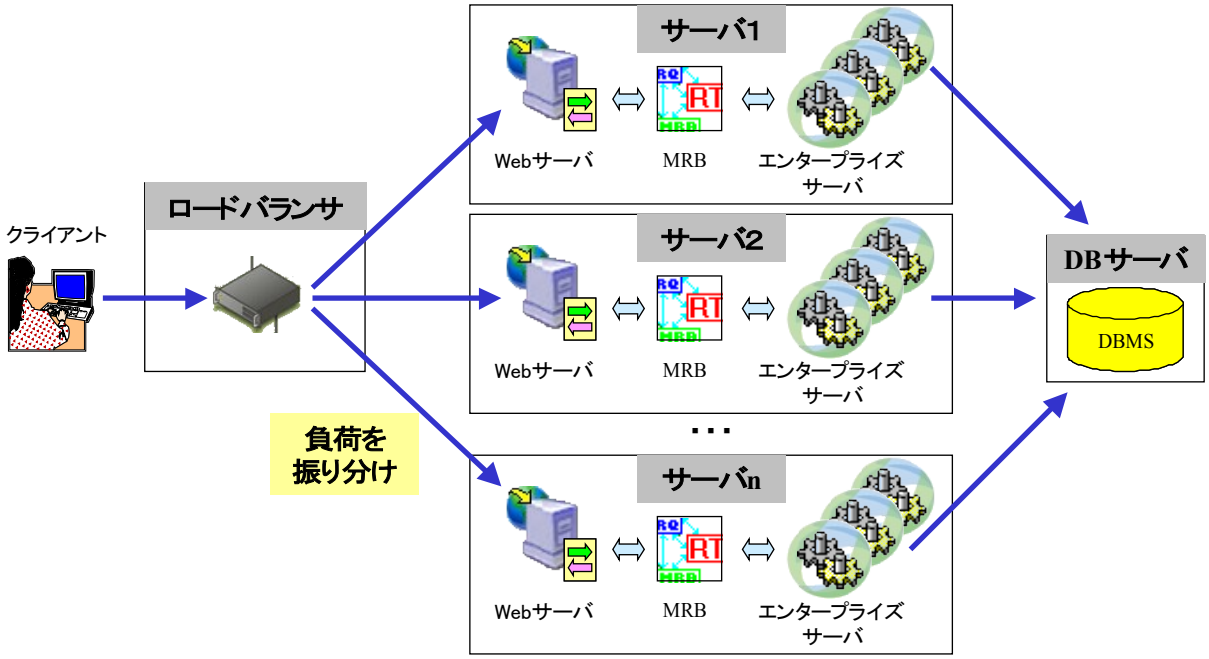
- インターネットリクエスタがリクエストを通常系の MRB に転送し、一定時間レスポンスがなかったときに、MRB に障害があったものと認識します。
- AltMessagingServer の記述があれば、そちらに設定されている MRB に、リクエストを転送します。リクエストが正常に処理されれば、以後のリクエストはすべてそちらの方に転送するようにします。

ライセンスは、通常系と待機系のそれぞれに必要になるので、2倍必要となりますが、このような冗長構成にする場合には、特別な「スタンバイライセンス」を利用することで割安になります。

3.20 ロードバランサによる多重化

前面にロードバランサを置き、その背後に Web サーバ→MRB→エンタープライズサーバを多系統配備。

可用性の向上と、負荷分散によるレスポンスタイムの向上のために、ロードバランサを利用することもできます。ロードバランサを利用するには、次のような構成とします。



- Web サーバ → MRB → エンタープライズサーバ という系統を複数用意します。ハード障害の可能性を考慮し、通常はそれぞれに別々のハードウェアを割り当てます。
- クライアントからのリクエストを受け付ける窓口となるサーバに、ロードバランサを配置します。
- ロードバランサが、Web サーバにリクエストを分散させるように設定します。

リッチクライアントサーバシステムの場合には、セッション維持のための設定がロードバランサに必要となります（5.21「ロードバランサを使うとき、セッション維持のためにどのように設定しますか？」参照）。

3.21 エンタープライズサーバが異常終了した場合、何が起こりますか？

実行中のリクエストは終了し、コミットされていないトランザクションはロールバックされます。MRB はリクエストのリトライを試みます。

エンタープライズサーバが異常終了して、インスタンスがなくなることがあります。これは、uniPaaS アプリケーションの問題のときもありますし、uniPaaS サーバ本体に不具合があって異常終了してしまうケースもあります。

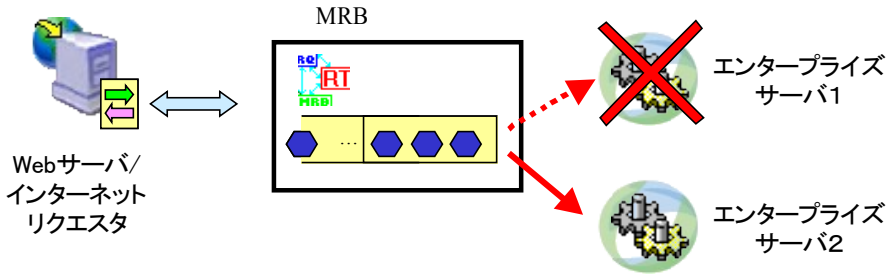
MRB がエンタープライズサーバの異常終了を検知すると、次のように回復処理を行います。

- マルチインスタンスの場合、他のインスタンスでリクエストが処理できれば、そのインスタンスにリクエストを送ります。

  リクエストが処理できるかどうか MRB は、次で判断します。

  - 同一アプリケーションが実行されている。
  - そのインスタンスに割り当てられた最大同時実行スレッド数に達していない。
- エンタープライズサーバの自動再起動の設定がされていれば、エンタープライズサーバを再起動します（→6.11「uniPaaS サーバの異常終了時、自動的に再起動させるようにできますか？」参照）。

このような回復処理が行われるので、ユーザから見ると、問題なく処理が行われたように見えます。



リッチクライアントサーバの場合、各ユーザに対応するコンテキストは、インスタンスのメモリ内に保存されているので、インスタンスが動作停止すると、そのインスタンスが保持しているコンテキストもすべて破棄されます。その結果、対応するクライアントはエラーで終了します。（5.17「クライアントが動作停止・切断した場合、コンテキストはどうなりますか？」参照）

3.22 リクエスタとは何ですか？どんな種類がありますか？

リクエスタとは MRB に対してリクエストを発行するモジュールを言います。

uniPaaS における「リクエスタ」というのは、MRB に対して各種の処理を実行するようリクエストを出すモジュールの総称です。

技術的により正確に言えば、リクエスタというのは、次のような uniPaaS 汎用通信 DLL の API を通して、MRB にリクエストを発行するソフトウェアモジュールです。

汎用通信 DLL：

- uniPaaS のインストールディレクトリに Mgrqgnrc018.dll という名前でインストールされます。ここで「018」というのは、uniPaaS のバージョン番号であり、V1.8 を意味します。uniPaaS のバージョンが異なると、この番号も異なります。
- uniPaaS モジュール間の低レベルの通信を司ります。
- uniPaaS モジュールの通信のための API が定義されています。MRB へのリクエストもすべてこの API を通して実行されます。この API は一般には公開されていません。

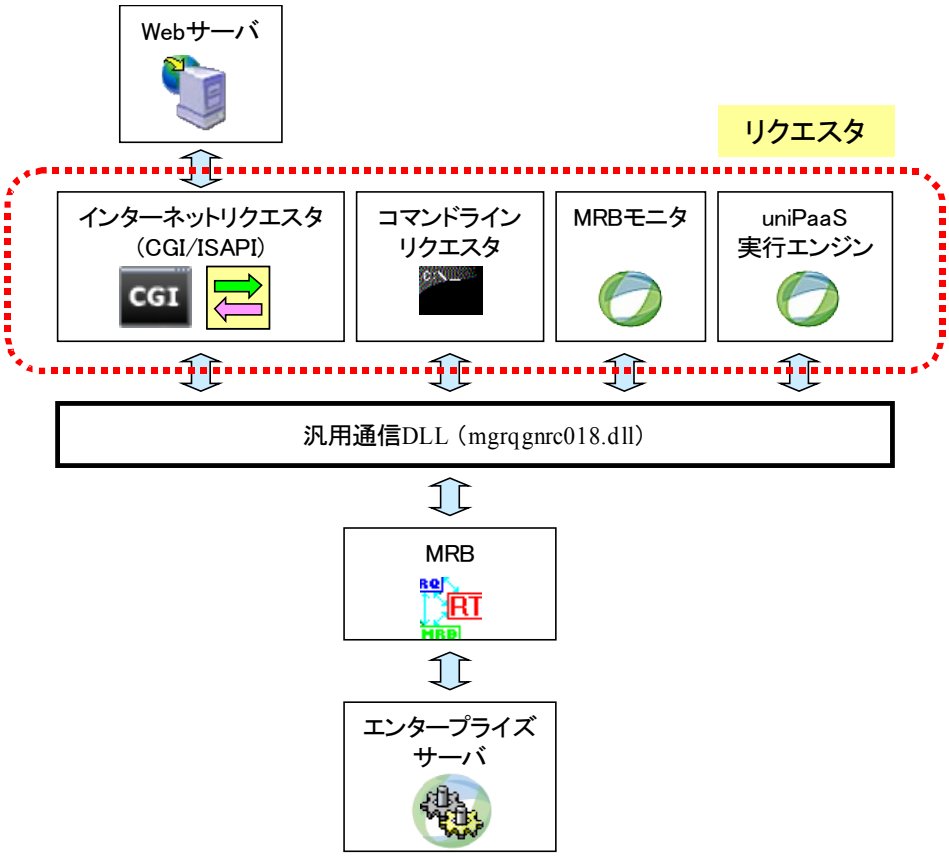
具体的には、この定義にあてはまるリクエスタとしては、次のようなものがあります。

- インターネットリクエスタ：3.1「Web システムにおけるソフトウェアモジュールとそれぞれの機能は？」で説明したように、インターネットリクエスタは、Web サーバの ISAPI/CGI インターフェースを通して、Web サーバへのリクエストを MRB へのリクエストへと変換し、結果を Web サーバに返すモジュールです。

  uniPaaS V1Plus においては、インストールディレクトリの下の Scripts ディレクトリにインストールされる Mgrqispi018.dll（ISAPI 用）、あるいは Mgrqcgi018.exe（CGI 用）です。
- コマンドラインリクエスタ：コマンドラインリクエスタとは、文字通り、コマンドライン（DOS 窓）からコマンドとしてリクエストを発行するものです。リクエストの内容は、コマンドラインパラメータとして渡します。DOS 窓から手で入力して uniPaaS サーバの動作テストを行ったり、.bat ファイルやシェルスクリプトから実行して結果をファイルに格納することができるので、定期的な監視やデバッグ情報収集などに利用できます。

  インストールディレクトリの下に Mgrqcmdl.exe という名前のモジュールがあるのが、これです。
- MRB モニタ：MRB モニタは、MRB の現在の状態（サーバエンジン、各サーバごとの負荷、コンテキスト数、リクエスト数の統計、リクエスト処理の履歴など）を表示するものです。

  インストールディレクトリの下にある uniRQMonitor.exe がこれです。
- uniPaaS 実行エンジン：uniPaaS 実行エンジン uniRTE.exe は、同一モジュールで、ライセンスの違いにより、uniPaaS クライアント、uniPaaS エンタープライズサーバ、uniPaaS リッチクライアントサーバとして動作します。この実行エンジンが「コールリモート」コマンドを実行して、別のエンタープライズサーバにあるプログラムを呼び出すこともできます。この場合には、実行エンジンは MRB に対してリクエストを発行するリクエスタとして動作することになります。この場合にも、汎用通信 DLL を通してリクエストを発行します。



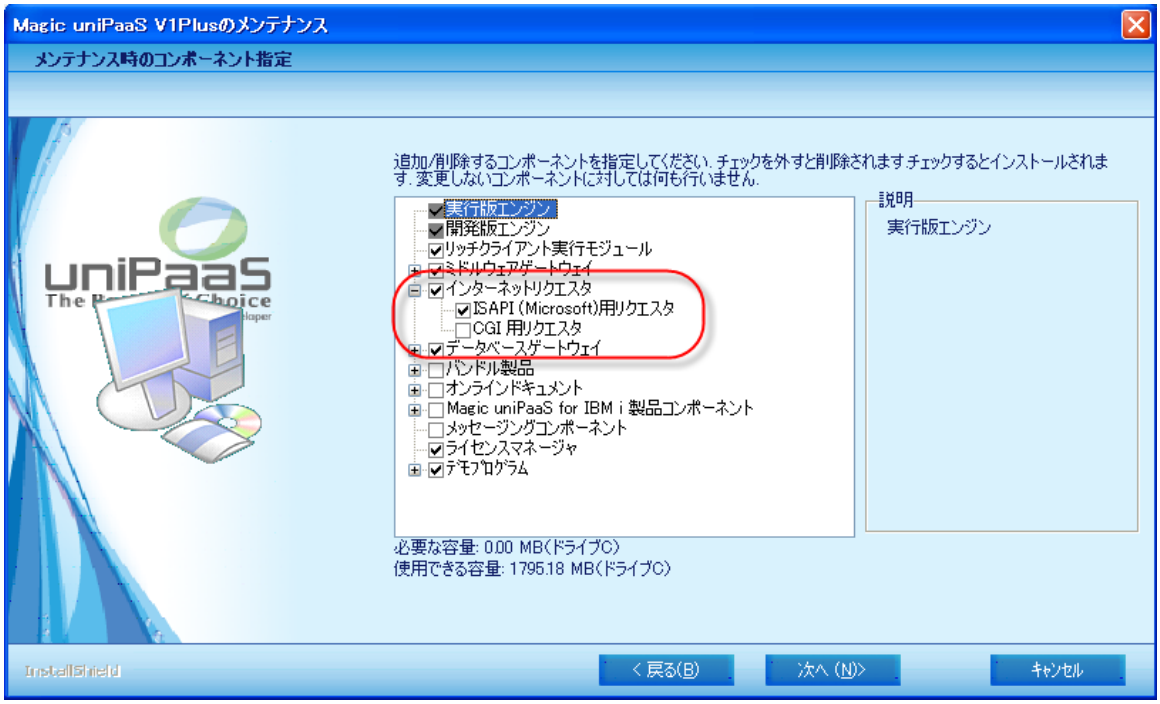
汎用通信モジュールが提供する API では、次のようなリクエストを発行することができます。

- アプリケーション名と公開プログラム名を指定して、uniPaaS サーバエンジンでタスクを実行させる。
- MRB の状態について問い合わせをする。これには次のような情報を取得することができます。
  - MRB が受け付けたリクエストの処理状況（実行中、実行済み、待機中、失敗、および統計情報など）
  - MRB が管理している uniPaaS サーバの状況（アプリケーション名、最大スレッド数/ユーザ数、現在処理中のスレッド数/ユーザ数、現在アクティブなコンテキスト、など）
- MGRB.INI に登録されているサーバを起動させる（6.5「MGRB.INI に定義されたコマンドを手動で起動することもできますか？」参照）
- 現在実行中のサーバを終了させる。
- MRB を終了させる。MRB の終了に先立って、登録されている uniPaaS サーバもすべて終了させられます。

3.23 ISAPI リクエスタと CGI リクエスタのどちらを使いますか？

IIS を使う場合には、ISAPI リクエスタを使います。ISAPI が使えない Web サーバの場合にだけ、CGI リクエスタを使います。

uniPaaS サーバをインストールする際、カスタムインストールを選択すると、インターネットリクエスタとして、ISAPI 用リクエスタと CGI 用リクエスタとを選択できます（下図）。



Web サーバとして、IIS（Internet Information Server）を使う場合には、ISAPI 用リクエスタを選択するようにしてください。インストーラは、IIS をシステム上で検出すると、デフォルトとして、ISAPI 用リクエスタを選択するようになっています。

CGI 用リクエスタは、IIS を使わず、別の Web サーバを使う場合にだけ選択するようにしてください。

ISAPI リクエスタと CGI リクエスタを比較すると、CGI リクエスタは EXE モジュールとして、リクエストのたびごとに起動・実行・終了するので、オーバーヘッドが大きく、Web サーバの負荷が高くなります。一方、ISAPI リクエスタは DLL として IIS のプロセスプールに一度ロードされると、IIS が停止するまで、そのまま処理できますので、オーバーヘッドが低くなります。

4 パーティショニングサーバ

パーティショニングサーバというのは、その実体はエンタープライズサーバですが、Web アプリケーションシステムとは使われ方が異なるだけです。このため、パーティショニングサーバも、多くの点において Web アプリケーションサーバと共通です。

本章では、前章での Web アプリケーションサーバの構成方法を念頭に置いて、パーティショニングサーバとしてエンタープライズサーバを使った場合、どのような点が異なるのか、について説明します。

4.1 パーティショニングサーバでのリクエストはどのような形式ですか？

uniPaaS のコール リモート コマンドにより、リクエストが発行されます。内部的には uniPaaS 独自の内部形式となっています。

パーティショニングサーバの場合、リクエストは uniPaaS の「コール リモート」コマンドを実行する uniPaaS の実行エンジンです。Web ブラウザは使いません。

パーティショニングサーバ自体は、uniPaaS エンタープライズサーバ製品を使って実現されます。一方で、uniPaaS エンタープライズサーバ自身がクライアントとして、コールリモートコマンドを使って、別の uniPaaS エンタープライズサーバをパーティショニングサーバとして呼び出すことができます。

ややこしいですが、「クライアント」と「サーバ」とは、相対的な概念で、サービスを提供するのがサーバであり、サービスを享受するのがクライアントである、と考えれば、サーバとして動いているアプリケーションが、別のサービスを呼び出した場合には、クライアントとして動作していることになります。このような「サーバが別のサーバを呼び出す」という関係は、基本的には、何段でも可能です。

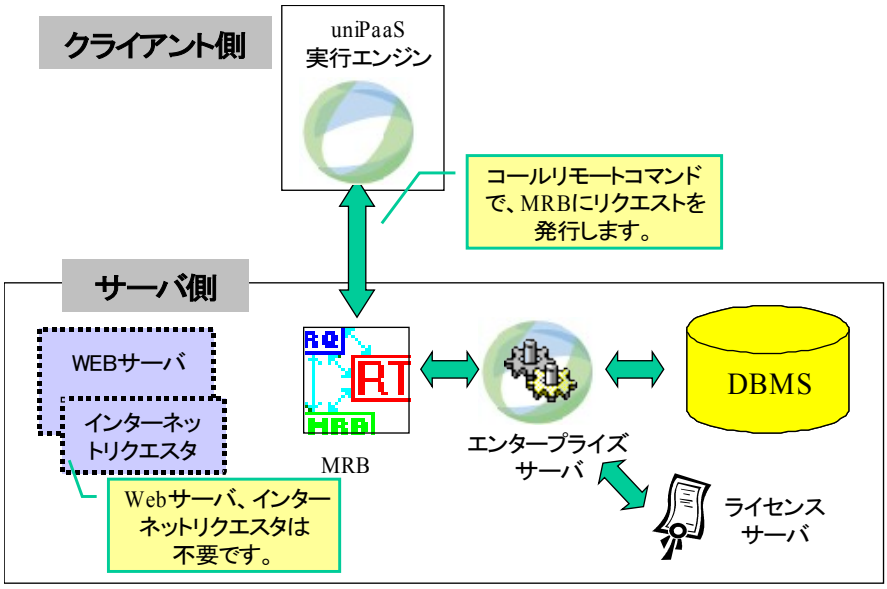
本章のパーティショニングサーバの説明中では、「クライアント」と言うのは、コール リモートコマンドを実行する uniPaaS 実行エンジンのことを指します。それは Client 製品でも Server 製品でも特に区別はしません。

4.2 パーティショニングサーバにおけるソフトウェアモジュールとそれぞれの機能は？

クライアント側は uniPaaS クライアントあるいはサーバ、サーバ側は MRB、エンタープライズサーバ、DBMS、ライセンスサーバが必要です。

パーティショニングサーバシステムでは、次のような点が Web アプリケーションサーバシステムと異なります。

- クライアント側は、Web ブラウザではなく、uniPaaS の実行エンジンです。
- リクエストは、URL で出すのではなく、uniPaaS のコール リモート コマンドにより発行されます。
- クライアントからのリクエストは、Web サーバを経由せず、直接 MRB に送られます。
- サーバ側には、Web サーバおよびインターネットリクエスタが不要です。



4.3 パーティショニングサーバの構成と設定はどうなりますか？

MRB、エンタープライズサーバ、ライセンスサーバを設定します。

パーティショニングサーバの構成は、Web サーバおよびインターネットリクエスタの設定がないことを除けば、Web アプリケーションサーバの場合と同じです。

MRB

MRB の設定は、エンタープライズサーバと同じで、ブローカ ポート番号を MGRB.INI で設定します。

エンタープライズサーバ

エンタープライズサーバの設定も同じで、MAGIC.INI で以下の情報を設定します。

- MRB 情報をサーバテーブルに定義
- MessageingServer でサーバ名（Default Broker など）を指定

Web サーバおよびインターネットリクエスタ

パーティショニングサーバでは、Web サーバやインターネットリクエスタは使いません。従って、Web サーバやインターネットリクエスタの設定は必要ありませんし、インストールする必要もありません。

ライセンスサーバ

ライセンスサーバは、Web アプリケーションサーバの場合と同じです。

具体的な設定については、構成により、3.14「オールインワンの場合の設定は？」～3.19「代替ブローカによる二重化」を参照してください。

4.4 パーティショニングサーバはどのような構成が可能ですか？

オールインワン、マルチインスタンス、マルチサーバ、代替ブローカ構成が可能です。ロードバランサは利用できません。

オールインワン構成

エンタープライズサーバと同じ設定で構成できます。

マルチインスタンス／マルチサーバ構成

エンタープライズサーバと同様、マルチインスタンス構成が可能です。同一 HW 上にマルチインスタンスを立ち上げることもできるし、複数の HW 上に複数のインスタンスを立ち上げることも可能です。

代替ブローカ構成

パーティショニングサーバでも代替ブローカ構成が可能ですが、代替ブローカを指定する場所が異なり、MGREQ.INI ではなく、リモートコールコマンドを実行する uniPaaS 実行エンジンの MAGIC.INI のサーバテーブルに設定します。

代替ブローカ構成にするための設定は、4.6「パーティショニングサーバでの代替ブローカはどこに設定しますか？」を参照してください。

パーティショニングサーバでは、リクエストが uniPaaS 独自の内部形式で送られます。一方ロードバランサは、リクエストの内容をある程度理解して処理する必要があるので、HTTP プロトコルなどの一般的なプロトコルにしか対応していません。このため、ロードバランサをパーティショニングサーバのリクエストの窓口として利用することはできません。

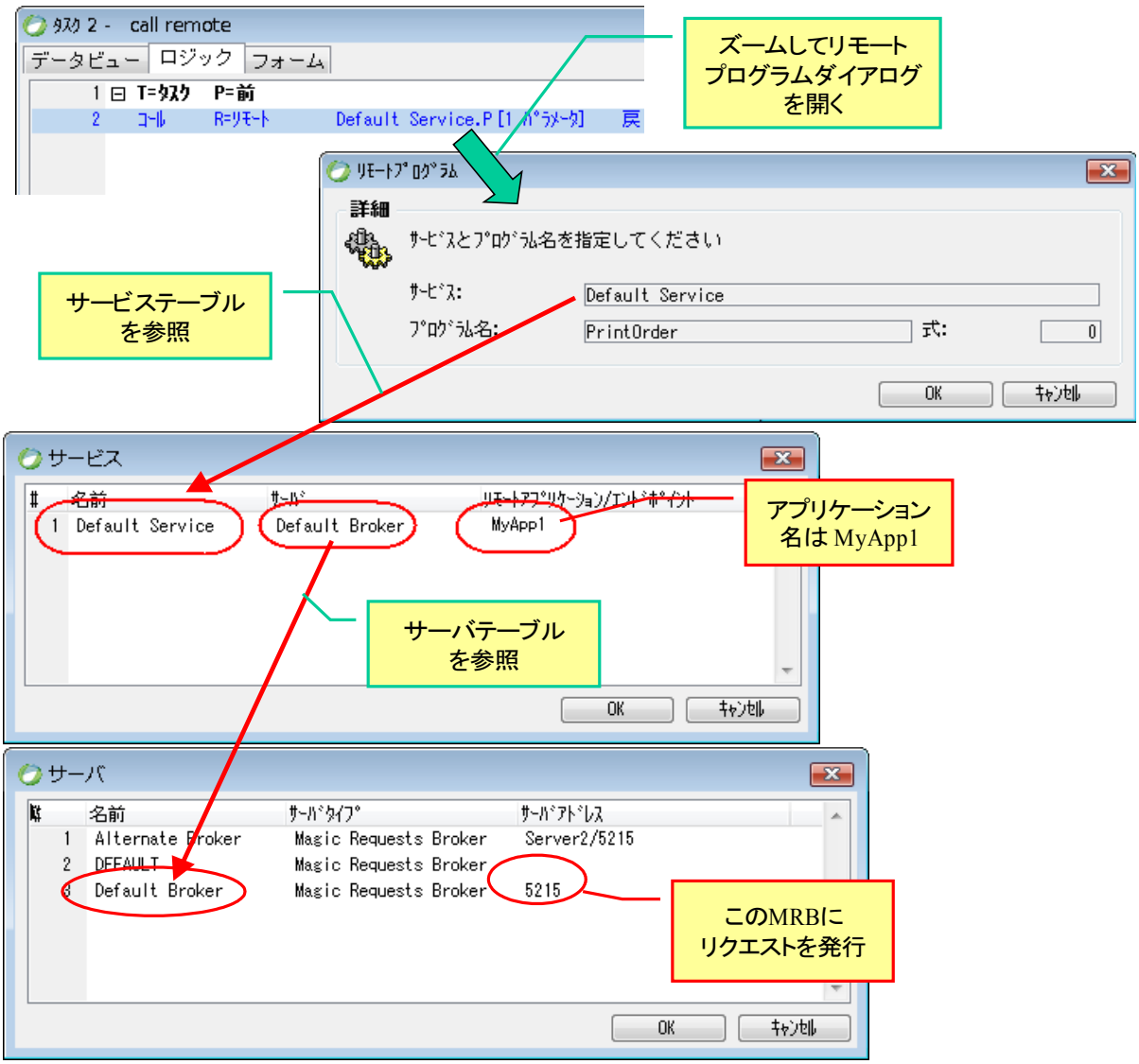
4.5 コールリモートで呼び出されるエンタープライズサーバはどのようにして指定しますか？

uniPaaS クライアントの「サーバ」テーブルおよび「サービス」テーブルにより指定します。

コール リモート でリクエストを発行するとき、どの MRB にリクエストを送るのかは、次の設定により決定されます。

- サーバテーブル：リクエストを送信する MRB を指定します。
- サービステーブル：実行すべきアプリケーション名を指定します。
- リモートコール コマンド：サービス名と呼び出す uniPaaS プログラムの公開名を指定します。

下図は、コール リモートコマンドを使った例です。



- コール リモートコマンドからズームすると、「リモートプログラム」ダイアログが開きます。この「サービス」欄で、サービステーブルに登録されているサービスを選びます。ここでは「Default Service」が選択されています。また、実行するプログラムの公開プログラム名（ここでは PrintOrder）も指定します。

- サービステーブルでは、サーバ（MRB）とアプリケーションとを定義します。「Default Service」サービスは、サーバとして「Default Broker」、アプリケーション名として「MyApp1」が設定されています。
- サーバテーブルで「Default Broker」が参照されます。ここでは、「サーバアドレス」が単にブローカポート番号 5215 だけ設定されているので、同一 HW 上で動作する MRB を参照しています。

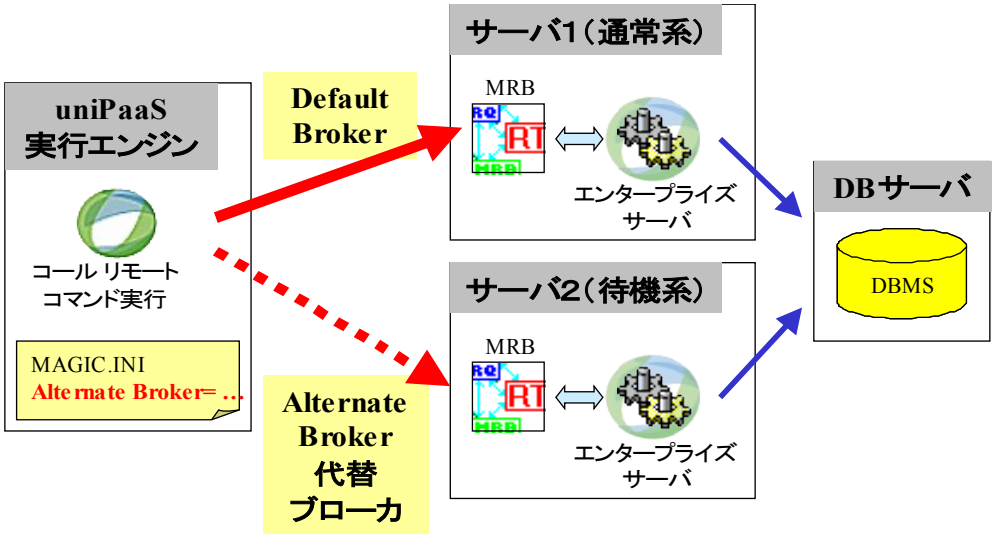
従って、コールリモートが実行されると、同一 HW 上で動作している MRB にリクエストが送られます。このリクエストのアプリケーション名は MyApp1 であり、公開プログラム名は PrintOrder です。

4.6 パーティショニングサーバでの代替ブローカはどこに設定しますか？

リモートコールコマンドを実行する uniPaaS 実行エンジンの MAGIC.INI のサーバテーブルに設定します。

4.6.1 MAGIC.INI での設定

下記は代替ブローカを設定した例です。



MAGIC.INI 代替ブローカ設定例

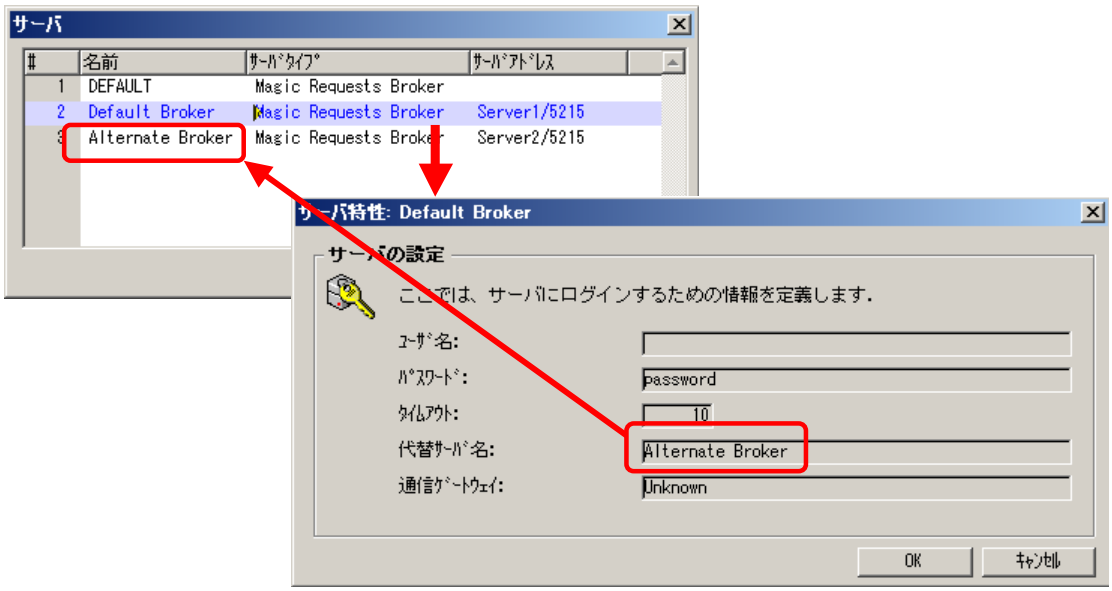
```
MessagingServer = Default Broker
[MAGIC_SERVERS]
Default Broker = 0,Server1/5215,,password,10,Alternate Broker,1
Alternate Broker = 0,Server2/5215,,,0,,1
```

通常系の MRB は、MessagingServer パラメータで指定されている「Default Broker」という名前の MRB で、これは、同一 HW 上でブローカ ポート 5215 でリクエストを待ち受けています。

待機系の代替 MRB は、[MAGIC_SERVERS] の Default Broker の 6 番目のパラメータとして指定されている「Alternate Broker」という名前のものです。これは、ホスト名 Server2 という名前のサーバ HW 上で、ブローカ ポート 5215 でリクエストを受け付けています。

4.6.2 uniPaaS エンタープライズサーバ画面での設定

代替ブローカの設定は、サーバテーブルからサーバ特性ダイアログを開いたときの、「代替サーバ名」に対応します。



4.6.3 障害時の動作

このように代替ブローカを登録してある場合の障害時の動作は、エンタープライズサーバの場合と同様です。（3.19.2「代替ブローカの動作」参照）

uniPaaS クライアントは、コール リモートコマンドでリクエストを発行する際、最初に通常系の MRB にリクエストを発行します。

一定のタイムアウト時間内に、通常系の MRB が応答しなかった場合には、待機系の 代替 MRB にリクエストを再送します。こちらが応答したならば、以後のリクエストは代替 MRB に発行します。

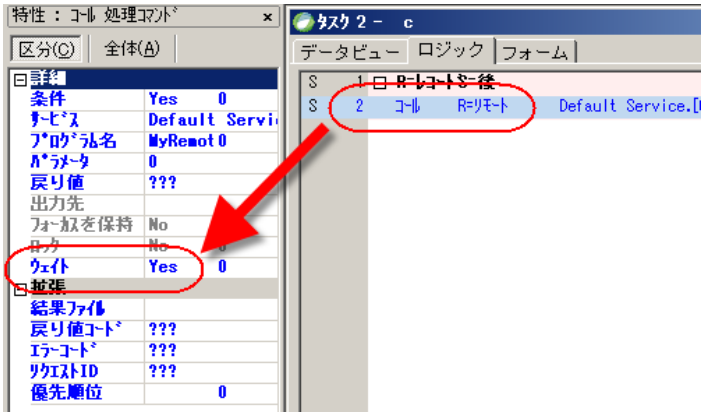
一定時間おきに、通常系の MRB が反応するかをチェックします。正常に反応するようになったならば、正常系が復旧したものとして、元通り、リクエストを通常系に発行するようにします。

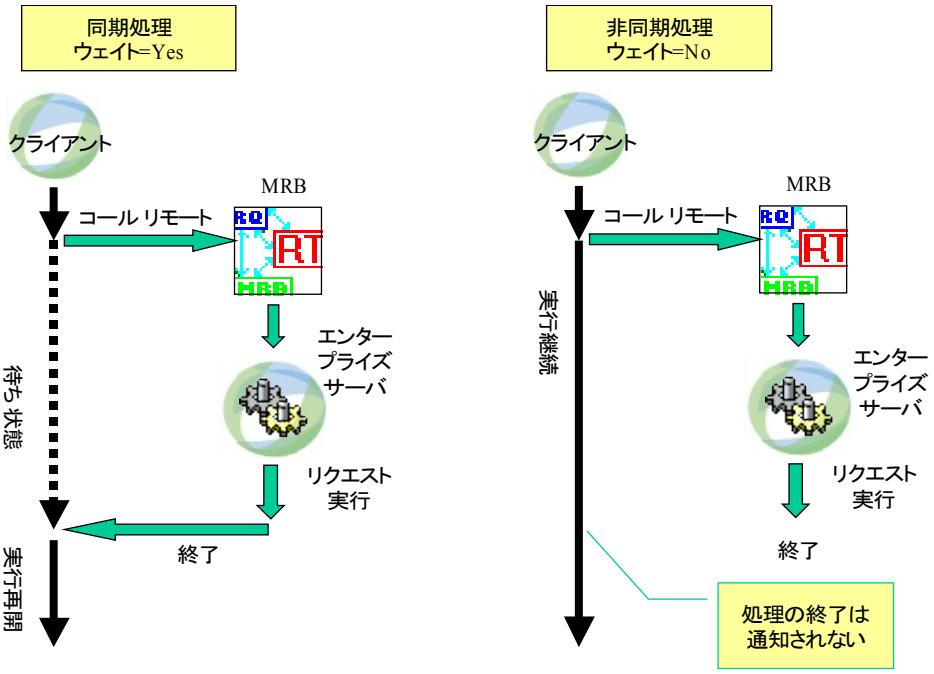
4.7 リクエスト処理が終了するまで、クライアントは待たされるのですか？

同期・非同期処理の区別があり、非同期処理では終了を待たずにクライアントに制御が戻ります。

コールリモートコマンドには、「ウェイト」という特性があり、これにより、同期処理か非同期処理かを指定できます。

- 同期処理では、パーティショニングサーバがリクエストの処理を終了するまで、コールリモートコマンドは待ち状態になります。「ウェイト」が Yes の時には同期処理となります。
- 非同期処理では、MRB にリクエストを発行し次第、コールリモートコマンドは終了し、次のコマンドに進みます。「ウェイト」が No の時には非同期処理となります。





非同期処理の場合には、クライアント側とサーバ側とは全く独立して実行が進むので、次のような点に留意してプログラムを作成する必要があります。

- エンタープライズサーバでの処理が終わっても、クライアント側に処理の終了は通知されません。サーバ側での処理が終了したかどうかは、サーバ側のプログラムで、管理用テーブルを設け、フラグを設定するなどして、アプリケーションで処理の状況を記録・監視するように設計する必要があります。
- サーバ側でパラメータ値を変更しても、クライアント側には反映されません。また、「戻り値」を設定しても、クライアント側には渡りません。
- 「結果ファイル」特性は、非同期の場合には無効になり、指定することができません。

4.8 パーティショニングサーバのライセンスは？

エンタープライズサーバと同じです。

パーティショニングサーバの実体はエンタープライズサーバそのものですので、パーティショニングサーバでもエンタープライズサーバのライセンスを使います。
ライセンスの体系もエンタープライズサーバと同じで、スレッドに対して課されます。

5 リッチクライアントサーバ

5.1 リッチクライアントサーバでの動作方式は、Web アプリケーションサーバやパーティショニングサーバと比べて、どう違いますか？

リッチクライアントサーバの動作方式は、ずっと複雑です。

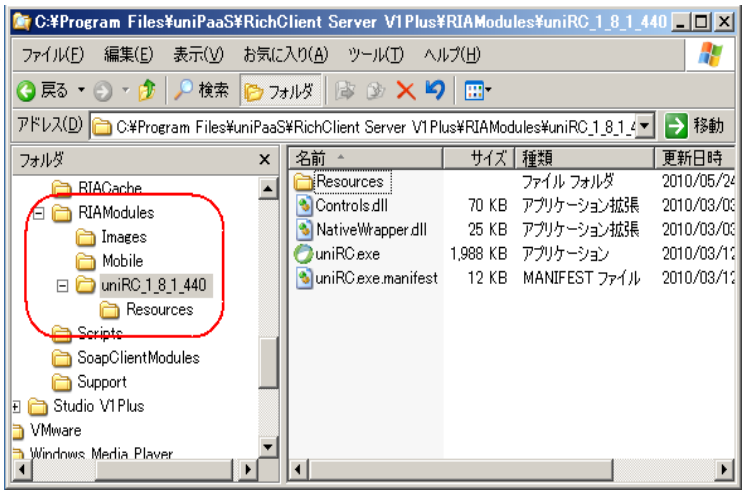
Web アプリケーションサーバやパーティショニングサーバと比べたとき、リッチクライアントサーバシステムは以下の点で動作方式が複雑です。

- クライアント側には、uniPaaS リッチクライアント専用のモジュール uniRC.exe が使われます。
- 起動には .NET Framework の ClickOnce の機能を利用します。
- クライアント モジュールと、サーバ側のリッチクライアントサーバモジュールとが、一連のセッションを作成して、協調しながら実行を進めていきます。

以下に、それぞれの点について説明します。

5.1.1 クライアント モジュール

クライアント側では、uniPaaS リッチクライアント専用のモジュール uniRC.exe が使われます。Web ブラウザは、リッチクライアントを起動の時に、ClickOnce の機能を利用するためだけに利用されます。いったんクライアントモジュールが起動されると、クライアントモジュールが自分でサーバと直接通信を行い、Web ブラウザは必要ありません。



クライアントモジュールは uniPaaS リッチクライアントサーバ製品に添付されており、デフォルトのインストールでは、インストールディレクトリのサブディレクトリ RIAModules の下にインストールされます。より正確には、更に、クライアントモジュールのバージョン番号をつけたサブディレクトリにあります。

このディレクトリは /uni18RIAModules という仮想ディレクトリとして、Web サーバで公開されています。実行時にはサーバからクライアント側にダウンロードされます。

5.1.2 ClickOnce

起動には .NET Framework の ClickOnce の機能を利用します。ClickOnce を使うことにより、uniRC.exe をはじめとする関連モジュールの自動配布、自動更新という、リッチクライアントの大きな利点を実現することができます。

一方、ClickOnce を使うことによる不便な点もあります。

- 起動するために、署名付きのマニフェストファイルを作成しなければならない。
- 認証のあるプロキシサーバでは、統合認証しかサポートされていない。

これらは、セキュリティの向上を目的とした、Microsoft 社による仕様です。uniPaaS も ClickOnce を使っている関係で、この仕様が適用されるようになります。

uniPaaS Studio 製品には、マニフェストファイルの作成のために専用のウィザードが用意されてあり、必要なパ

ラメータを指定するだけで、起動用のマニフェストファイルを作成することができますから、開発者の負担が大きくなることはありません。

マニフェストファイルをどうしても手作業で変更したい、というような場合には、ClickOnce でのマニフェストファイルに関する最低限の知識が必要となります。

5.1.3 クライアントモジュールとサーバモジュールの協調動作

Web アプリケーションサーバあるいはパーティショニングサーバでは、リクエスト→レスポンスという処理は1回ごとに完結した独立したものとなっているので、動作は比較的単純でした。

リッチクライアント サーバ システムでは、クライアント側の uniRC.exe と、サーバ側のリッチクライアントサーバモジュール uniRTE.exe とが、協調しながら実行を進めていきます。このため、サーバ側にも、クライアント側での実行状態や一時データ（5.2「リッチクライアントサーバでの「コンテキスト」とは何ですか？」参照）を保持しておいて、協調・同期しながら処理を進めていく必要があります。

5.2 リッチクライアントサーバでの「コンテキスト」とは何ですか？

クライアントごとの実行状態や一時データからなる、サーバ側に保持されるデータです。

リッチクライアントサーバにおいては、「コンテキスト」について考慮が必要になります。

リッチクライアントシステムでは、タスクの開始から終了までの一連の処理（タスク初期化→タスク前処理→レコード読み込み→レコード前処理→ユーザの入力→イベントハンドリング → レコード後処理 → タスク後処理 → タスクの終了化処理）が、クライアントモジュールとサーバモジュールとの両方で、正確に同期しながら行われなければなりません。また、一つのタスクの実行中に、別のタスク(リッチクライアントタスク、バッチタスク）が呼び出されるのが一般的です。

このことから、以下のような実行状態データを、サーバ側でも管理しておく必要があります。

- クライアントで実行中のタスクの状態（レベル、ハンドラ、フロー）、およびそのタスクでのデータビュー
- タスクの呼び出し関係（コールスタック）

このほかに、次のような一時データも、クライアント毎に管理しておく必要があります。

- メモリテーブルのデータ
- オープンしている DBMS テーブルのカーソルなどの内部データ
- グローバルパラメータ（GetParam/SetParam で扱う）

このようなデータをまとめて、**コンテキスト** と呼びます。

コンテキストは、次のような特性があります。

- 一つのクライアントに対し、一つ作成されます。
- 特定のリッチクライアントサーバ インスタンスのメモリ内に作成され、保持されます。
- 一意な ID（コンテキスト ID)が付加され、識別されます。
- クライアントからのリクエストにはコンテキスト ID が指定されています。MRB はリクエストのコンテキスト ID を認識して、リクエストを適切なリッチクライアントサーバ インスタンスに転送します。
- コンテキストは、uniPaaS サーバが扱う内部データであり、開発者が直接アクセスして扱えるものではありません。

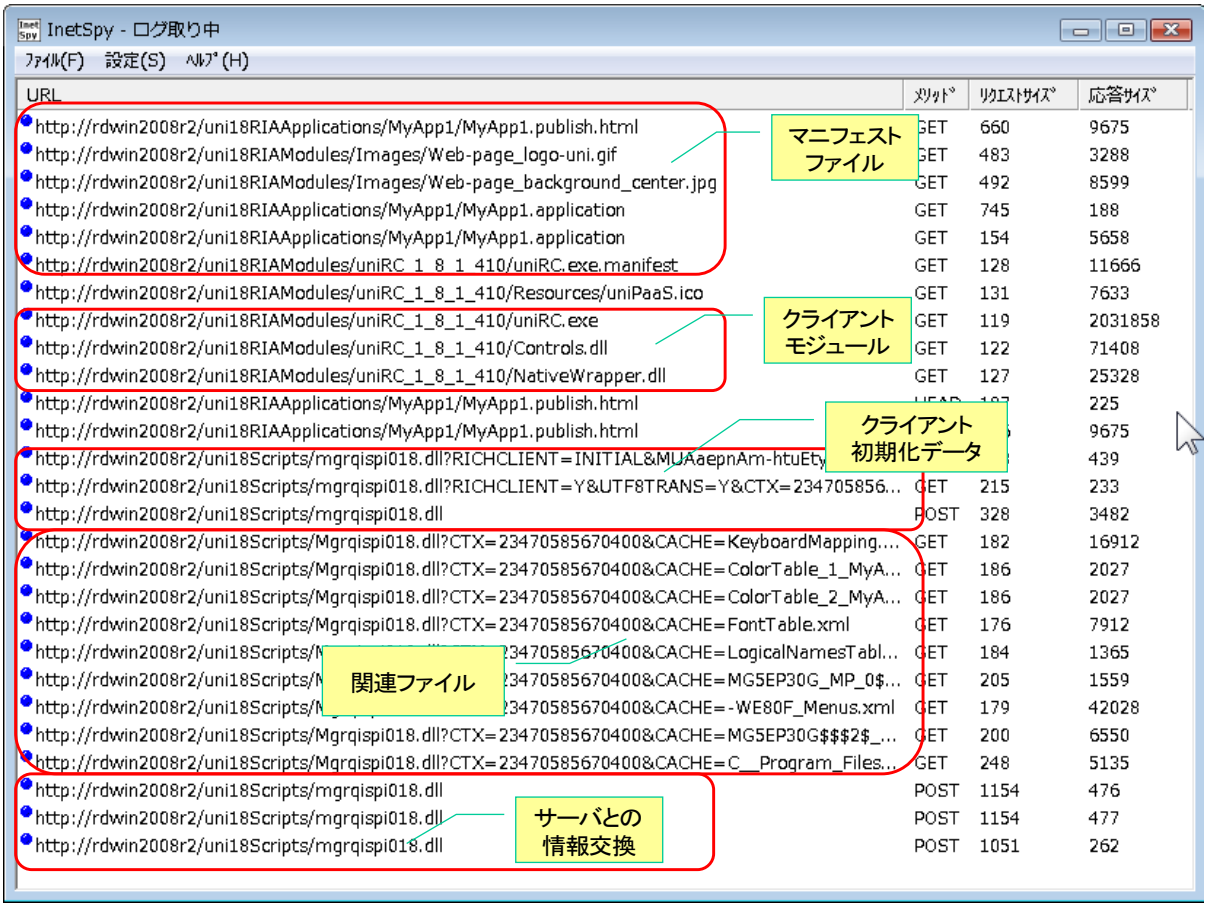
5.3 リッチクライアントサーバでのリクエストはどのような形式ですか？

起動時には ClickOnce、その後は HTTP を使った内部形式でリクエストをやりとりします。

uniPaaS リッチクライアントでは、起動には ClickOnce を使い、その後は、独自形式の XML データをインターネットリクエスタに POST することにより実行が進みます。

通常は、開発者はマニフェストファイルをウィザードを使って作成すれば良く、ひとつひとつのリクエストの内容を気にする必要はありません。クライアントモジュールとサーバの間のデータは、uniPaaS が自動的に作成します。もちろん、開発者が複雑な XML データを自分で作成する必要もなく、明示的に Web サーバにリクエストを発行する必要もありません。

リクエストがどのようなものかというイメージをつかむために、クライアントとサーバのやりとりをキャプチャした例を次に示します。



詳細は説明しきれませんが、次のような情報がやりとりされていることがわかります。

ClickOnce のマニフェストファイル：

- MyApp1.publish.html（起動用 HTML）
- MyApp1.application（デプロイメント マニフェスト）

- uniRC.exe.manifest (クライアントモジュールのアプリケーション マニフェスト)

クライアントモジュール:

- uniRC.exe
- Controls.dll
- NativeWrapper.dll

クライアント初期化データ:

- これには、クライアントモジュールの初期化のための情報や、メインプログラム・初期起動プログラムのタスク定義、データビューなどが含まれます。

関連ファイル:

- キーボードマッピングファイル、色定義ファイル、フォント定義ファイル、論理名定義ファイル、メニュー定義ファイル、その他がダウンロードされます。

サーバとの情報交換:

- インターネットリクエスタに対して、POST によりデータが送られています。

  この例では、3行だけしか記録されていませんが、実際のアプリケーションにおいては、この部分がプログラムの終了まで続きます。

5.4 リッチクライアントサーバでのソフトウェアモジュールは？

クライアント側：

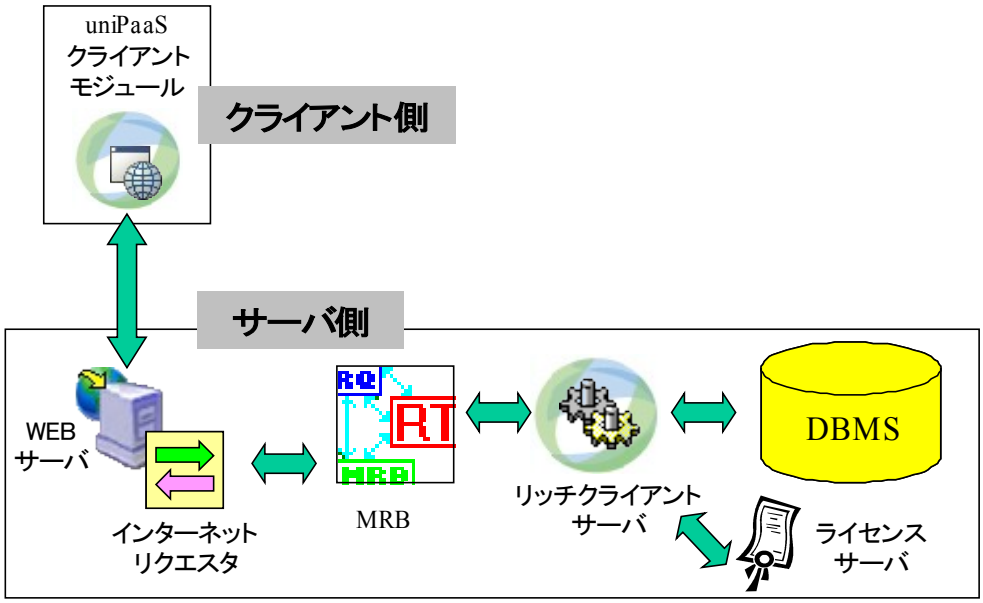
- uniPaaS リッチクライアント専用クライアントモジュール
- .NET Framework 2.0

サーバ側：

- Web サーバ
- インターネットリクエスタ
- MRB
- リッチクライアントサーバ
- DBMS

リッチクライアントサーバシステムでのソフトウェアモジュールは、Web アプリケーションサーバシステムの場合と似ていますが、次の点が異なっています。

- クライアント側は、Web ブラウザではなく、uniPaaS リッチクライアント専用のクライアントモジュールを使います。
- サーバ側では、エンタープライズサーバではなく、リッチクライアントサーバを使います。



5.5 マニフェストファイルとは何ですか？

ClickOnce での起動時に使うファイルで、

- アプリケーション マニフェスト
- デプロイメント マニフェスト

があり、その他に、.publish.html ファイルが使われます。

uniPaaS では、Studio のリッチクライアント インターフェース ビルダにより作成されます。

それぞれの内容と機能については、ClickOnce の技術情報などを参照していただくこととして、uniPaaS のリッチクライアントでは、次のようなファイルがそれぞれに該当します。

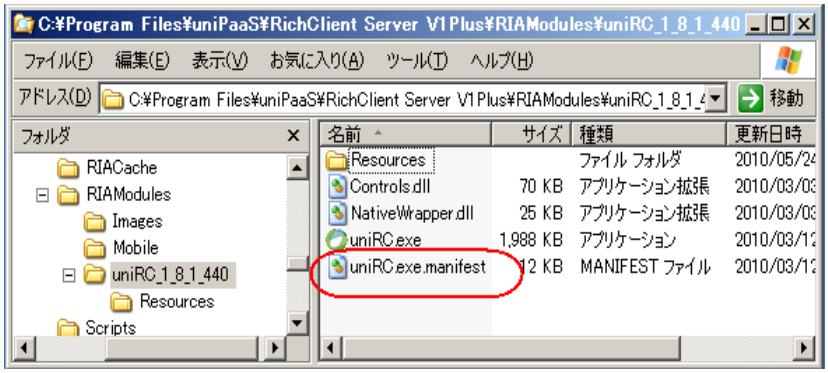
5.5.1 アプリケーション マニフェスト

アプリケーションマニフェストファイルは、XML の形式でクライアントモジュール uniRC.exe についての情報と、依存する DLL、.NET Framework その他のソフトウェアモジュールに関する情報が記述されています。

アプリケーション マニフェストファイルは以下の場所にあり、拡張子が .manifest となっています。

```
(インストールディレクトリ)¥RIAModules¥uniRC_1_8_1_xxx¥uniRC.exe.manifest
```

ここで、xxx というのは、uniRC.exe モジュールのバージョン番号で、uniPaaS のバージョンやサービスパック等により変わります。



アプリケーションマニフェストファイルは、証明書により署名されているので、テキストエディッタ等で修正すると無効になってしまいます。

このファイルは、開発元の Magic Software Enterprises 社から uniRC.exe ファイルと共に提供されるもので、開発者やユーザが作成したり手を加えたりする必要はありません。

5.5.2 デプロイメントマニフェスト

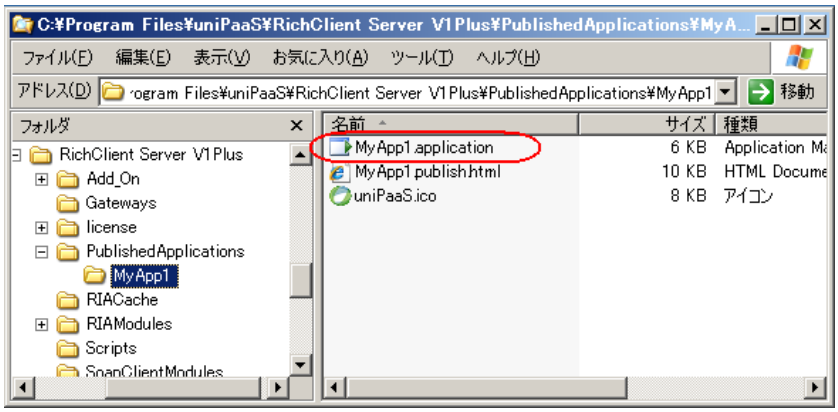
デプロイメントマニフェストは、uniPaaS で開発したアプリケーションに関する情報や、利用するクライアントモジュールについての情報 (ダウンロード先の URL やバージョンなど)が XML 形式で記述されています。

uniPaaS リッチクライアントでのデプロイメント マニフェストは、開発者が、Studio の リッチクライアント インターフェース ビルダを使って、ウィザード形式でパラメータを設定して、作成します。

作成されたデプロイメントマニフェストファイルは以下の場所にあり、拡張子が .application になっています。

(インストールディレクトリ)¥PublishedApplications¥(アプリケーション名)¥(アプリケーション名).application

ここで、(アプリケーション名) というのは、Studio で開いているプロジェクト名と同じにすることが普通ですが、リッチクライアント インターフェース ビルダ で、別の名前を指定することも可能です。



デプロイメントマニフェストは、ClickOnce で uniPaaS リッチクライアントを起動する際に URL として指定するファイルとなります。

PublishedApplications ディレクトリは、インストーラが /uni18RIAApplications という名前の仮想ディレクトリを割り当てますから、ユーザが指定する URL は、例えば、

http://MyRCServer/uni18RIAApplications/MyApp1/MyApp1.application

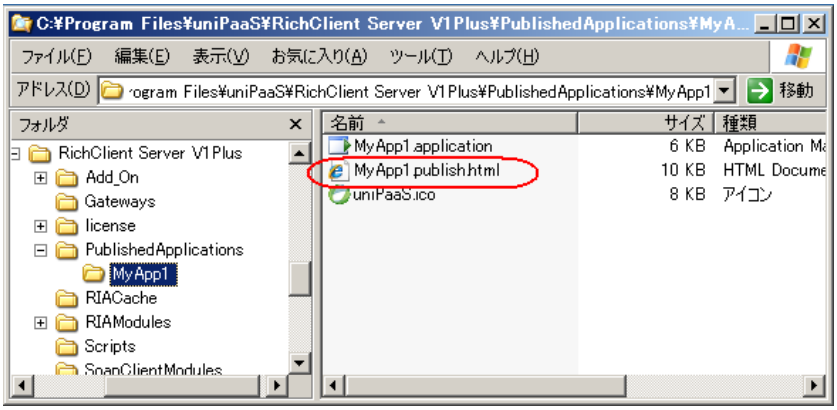
というようなものになります。

デプロイメントマニフェストは、通常、一つのアプリケーションに対して一つ作成しますが、同一のアプリケーションで、異なった公開プログラムから開始したい場合には、複数作成することも可能です。

デプロイメント マニフェスト ファイルにも、証明書で署名がされますので、テキストエディッタ等で直接修正すると無効になり、リッチクライアントの起動時にセキュリティエラーとなります。デプロイメントマニフェストファイルの内容を修正したい場合には、Studio のリッチクライアント インターフェース ビルダを使って、再作成してください。

5.5.3 .publish.html ファイル

このファイルは、ClickOnce で言うところのマニフェストファイルではありませんが、ClickOnce によるリッチクライアント起動の手軽な入り口として使うことができるので、ここで説明しておきます。



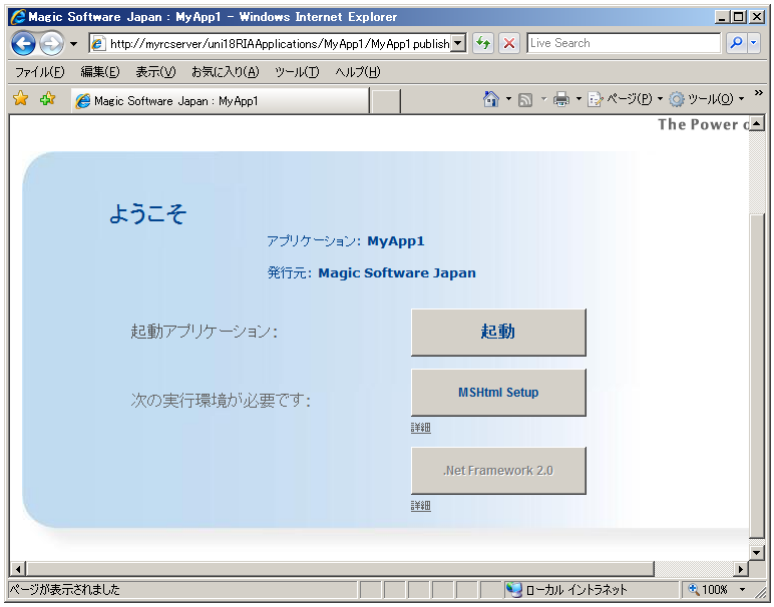
このファイルは、ごく単純に言ってしまえば、デプロイメント マニフェスト ファイルを ボタンクリックにより呼び出すことができるような .html ファイルです。JavaScript が使ってありますが、要は onClick により、上記のようなデプロイメント マニフェストファイルの URL を参照するものです。

.publish.html ファイルは、デプロイメントマニフェストと同様、開発者が、Studio のリッチクライアント インターフェース ビルダを使って作成し、デプロイメント マニフェストファイルと同じディレクトリに作成されます。

ファイル名は次のようになっています。

(インストールディレクトリ)¥PublishedApplications¥(アプリケーション名)¥(アプリケーション名).publish.html

これを Web ブラウザで表示すると、次のようになります。



この画面で、「起動」のボタンを押せば、リッチクライアントが起動されます。

付加的な機能として、依存ソフトウェアがインストールされているか否かをチェックし、インストールされていなければ、インストーラをダウンロードしてインストールできるようになっています。

uniPaaS リッチクライアントを実行する際に、クライアント モジュールが依存するソフトウェアモジュールには次のようなものがあり、起動に先立って、クライアント側でインストールされている必要があります。

- .Net Framework 2.0
- MSHTML

このいずれも、インストーラが uniPaaS Studio や リッチクライアントサーバ 製品に添付されており、RIAModules ディレクトリに格納されています。

RIAModules ディレクトリは、インストーラが /uni18RIAModules という仮想ディレクトリに割り当てますので、クライアント側の利用者は、まだインストールされていなければ、リッチクライアント サーバからダウンロードしてインストールすることができます。

.puclish.html ファイルでは、利用者の便を考え、これらのソフトウェアモジュールを onClick で参照するボタン（画面中の「MSHtml Setup」および「.NET Framework 2.0」ボタン）が定義されていますので、ユーザはこのボタンを押すことにより、インストーラをダウンロードし、インストールすることができます。

これらのソフトウェアモジュールがすでにインストールされている場合には、.ダウンロードのボタンは無効化されています。これは、JavaScript と OCX により実装されています。

このファイルは前述のマニフェストファイルとは異なり、HTML エディッタなどを使って、自由に編集することができます。

ただし、.publish.html ファイルには、画面上には表示されませんが、起動するアプリケーション名と公開プログラム名などの情報も記述されていますので、その部分は変更しないでください。（→5.8「起動するアプリケーション名とプログラム公開名等はどこに指定されているのですか？」参照）

5.6 デプロイメント マニフェストファイルと、.publish.html ファイルと、どちらを使って起動するのが良いですか？

依存ソフトウェアモジュールがすでにインストールされているのが確実ならば、デプロイメントマニフェスト .application を呼び出し、そうでないならば、.publish.html を呼び出すのが良いでしょう。

.publish.html も、ボタンを押すと結局デプロイメント マニフェストファイルを呼び出すので、最終的には同じ結果となりますが、使い勝手に一長一短があります。大きな違いは、リッチクライアントのクライアント側モジュールに必要な依存ソフトウェアモジュールのインストールをどうするかというところにあります。

5.6.1 依存ソフトウェアモジュール

リッチクライアントのクライアント側モジュールが依存するソフトウェアモジュールには、次のものがあります。

- .NET Framework 2.0
- MSHTML

これらは、いずれも uniPaaS リッチクライアントサーバ製品に同梱されており、RIAModules ディレクトリにインストールされます。

5.6.2 .publish.html からの起動

.publish.html は、クライアントの PC に、依存ソフトウェアモジュールがインストールされているかわからない場合に、起動用画面として利用します。

.publish.html を web ブラウザで開くと、下図のような画面が表示されます。この画面は、HTML で記述されているものなので、カスタマイズが可能です。

この画面では、JavaScript と ActiveX とを使って、上記の依存ソフトウェアモジュールがインストールされているかをチェックします。もしインストールされていないことを検出すると、それぞれのボタンが有効化されます。上記の画面では、.NET Framework 2.0 はすでにインストールされているので、「.NET Framework 2.0」のボタン（ー

番下のもの）は無効化されていますが、MSHtml はインストールが検出されなかったので、「MSHtml Setup」のボタン（2 番目のもの）が有効化されます。

利用者は、この画面を見て、ボタンを押すことにより、必要モジュールをダウンロード・インストールすることができます。

先の図では、「MSHtml Setup」ボタンを押すと、MSHtml のインストーラがリッチクライアントサーバの uni18RIAModules 仮想ディレクトリからダウンロードされ、インストールされます。

依存ソフトウェアモジュールが両方インストールされれば、「起動」ボタンを押すことにより、デプロイメントマニフェスト（.application）ファイルがダウンロードされ、ClickOnce が起動されて、リッチクライアントのクライアントモジュールの実行が開始されます。

5.6.3 デプロイメント マニフェスト ファイルからの起動

依存ソフトウェアモジュールがすでにインストールされていることがわかっている状況ならば、直接、デプロイメントマニフェストファイルを呼び出す方が良いでしょう。リッチクライアントを使う PC が特定されており、予めインストールしておくことができるような場合(小規模な社内システムなど)などがこれにあたります。

.publish.html を使う場合には、一旦 Web ブラウザで HTML が表示され、「開始」ボタンを押して、初めてリッチクライアントタスクが開始されます。すなわち、HTML の表示→ボタンクリック、という操作が 1 ステップ入ります。

これに対して、デプロイメント マニフェスト ファイルを直接参照する場合には、すぐにリッチクライアントが開始しますから、ボタンを押す手間が省けます。

依存ソフトウェアモジュールがインストールされていないところで、.application を直接呼び出すと、起動時に エラーとなります。現場での運用状況を考慮して、いずれかを決めてください。

5.7 バージョンアップ時にマニフェストファイル再作成が必要ですか？

はい。開発環境（Studio 製品）の MAGIC.INI を変更し、ウィザード再実行の必要があります。

uniPaaS リッチクライアントサーバ製品をバージョンアップすると、クライアント側ソフトウェア uniRC.exe のバージョンが変わります。

uniRC.exe は、Magic をインストールしたディレクトリのサブディレクトリ RIAModules の、更にサブディレクトリ uniRC_1_8_1_xxx の下に格納されています。ここで、「1_8_1_xxx」というのは、uniRC.exe のバージョン番号を反映したもので、製品のバージョンにより変わってきます。通常このディレクトリ名は、インストーラが自動的に作成します。

マニフェストファイルには、uniRC.exe およびそのマニフェストファイル uniRC.exe.manifest の URL を参照している箇所があるので、uniRC.exe が変わったら、マニフェストファイルも変更しなければなりません。マニフェストファイルはデジタル署名されているので、単にテキストエディッタで変更することはできません。このため、ウィザードを再起動する必要があります。

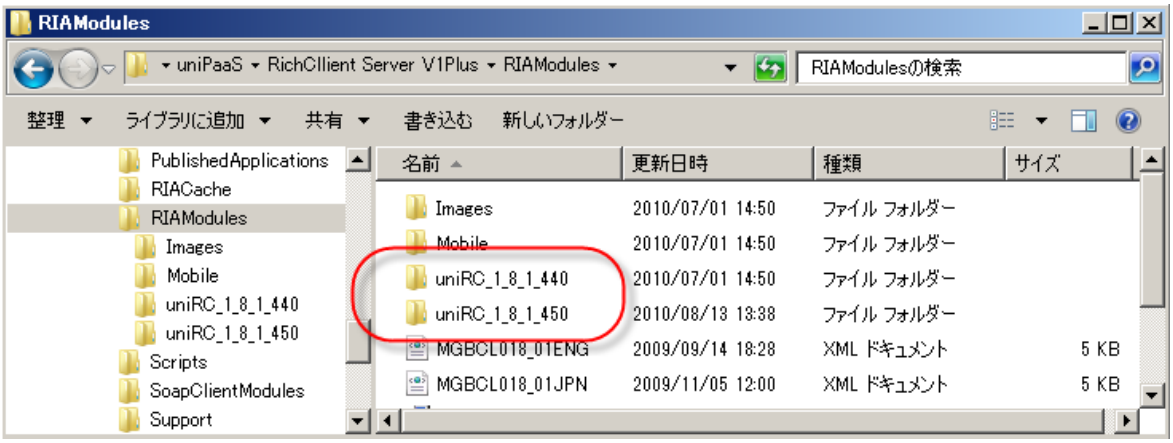
また、ウィザードは MAGIC.INI ファイルの設定を参照しているところがあるので、MAGIC.INI の設定も一部変更する必要があります。

以下に変更方法を説明します。

uniRC.exe の最新のディレクトリの確認

1. まず、バージョンアップは、開発環境（Studio 製品をインストール）と、実行環境（uniPaaS リッチクライアントサーバ）の両方について行う必要があります。
2. バージョンアップが終了したら、RIAModules ディレクトリを見ます。ここに uniRC_1_8_1_xxx という名前のサブディレクトリが複数あるはずですので、その中で最新のもの（一番数値が大きいもの）を確認します。

   例えば、下図の例では uniRC_1_8_1_450 が最新のディレクトリ名です。



Studio の MAGIC.INI の変更

3. 開発環境で、Studio 製品の MAGIC.INI を開き、[MAGIC_RIA] セクションに移動します。
4. ClientModulesPath パラメータを、最新の uniRC.exe のあるディレクトリ名に変更します。（uniPaaS インストールディレクトリからの、相対パスで指定します）。

```
[MAGIC_RIA]
ClientModulesPath=RIAModules¥uniRC_1_8_1_450¥
```

5. MAGIC.INI を保存します。

ウィザードの再実行

6. Studio を起動します。
7. プロジェクトを開き、「リッチクライアント インターフェース ビルダ」を再実行します。指定パラメータ等は特に変更する必要はありません。
8. PublishedApplications 下のマニフェストファイル（.application および .publish.html ファイル）が更新されているはずですので、実行環境にコピーします。

5.8 起動するアプリケーション名とプログラム公開名等はどこに指定されているのですか？

publish.html ファイル中に XML の形式で格納されています。

Web アプリケーションの場合と異なり、リッチクライアントの場合には、アプリケーション名や公開プログラム名を URL などで直接指定することはありません。リッチクライアントを呼び出す場合に使う URL は、デプロイメント マニフェスト ファイル (MyApp.application など、拡張子 .application を持つファイル)です。

それでは、実際に呼び出されるアプリケーション名や公開プログラム名がどこに指定されているかというと、.publish.html ファイルにあります。

.publish.html ファイルをテキストエディッタで開き、HTML のソースを見てみると、後ろの方に次のような記述があります。この中に、server、appname、prgname などの名前で定義されているプロパティが、それぞれ、サーバ名、アプリケーション名、公開プログラム名です。

```
・・・
<body onload="initialize()">
   <xml id="rcExecProps">
      <properties>
        <property key="protocol" val="http"/>
        <property key="server" val="MyRCServer"/>
        <property key="requester" val="uni18Scripts/mgrqispi018.dll"/>
        <property key="appname" val="MyApplication"/>
        <property key="prgname" val="top"/>
        <property key="envvars" val=""/>
        <property key="DisplayStatisticInformation" val="N"/>
        <property key="LogClientSequenceForActivityMonitor" val="N"/>
        <property key="InternalLogLevel" val=""/>
        <property key="InternalLogFile" val=""/>
        <property key="UseWindowsXPThemes" val="Y"/>
      </properties>
    </xml>
・・・
```

このファイルは Studio のリッチクライアント インターフェース ビルダにより、デプロイメントマニフェストファイル (.application ファイル) と共に作成されます。サーバ名やアプリケーション名等に変更があった場合には、Studio のウィザードにより再作成してください。

5.9 リッチクライアントプログラムの起動から終了までのサイクルは？

大別して、次のような4段階からなります。

- ClickOnce による起動
- 初期化処理
- 実行の進行
- 終了化処理

5.9.1 ClickOnce による起動

リッチクライアントが開始されるにあたっては、ClickOnce の機能を借りて、クライアントモジュールのダウンロードとインストール・起動が行われます。ここでは、次のようなステップがあります。

1. 利用者が、Web ブラウザから .publish.html ファイルを呼び出します。
2. Web ブラウザ上で .publish.html ファイル が表示されると同時に、ActiveX により、.Net Framework 2.0 および MSHTML がインストールされているかチェックが行われます。インストールされていないことが検出されたら、ユーザにダウンロード・インストールするよう、促します。
3. .application ファイルの呼び出し。ここでクライアントモジュールのバージョンチェックが行われ、必要ならばダウンロード ・インストールします。
4. クライアントモジュールが起動されます。

これは、初めてリッチクライアントを起動する場合の動作です。2 回目以降は、必要なモジュールがインストールされているので、プログラムメニューに作成されたショートカットから起動することができます。この場合にはもう Web ブラウザは必要ありません。

5.9.2 初期化処理

クライアントモジュールが起動されたら、リッチクライアントサーバと通信を行いながら、初期化処理が行われます。ここではクライアントモジュールが次のようなステップを行います。

1. クライアントモジュールが、リッチクライアントサーバに対し、プログラム起動を要求します。
2. リッチクライアントサーバは内部にコンテキストを作成し、タスク定義、必要なファイルの URL などの情報をクライアントモジュールに返送します。必要ファイルには、キーボード定義ファイル、色定義ファイル、フォント定義ファイル、メニュー定義などがあります。
3. クライアントモジュールが必要ファイルをダウンロードします。
4. クライアントモジュールにより、最初のタスク（ルートタスク）の実行が開始されます。

5.9.3 実行の進行

1. クライアントモジュールが、リッチクライアントサーバと交信を行いながら、実行を進行していきます。

5.9.4 終了化処理

ユーザがルートタスクをクローズすると、次のように終了化処理が行われます。

1. クライアントモジュールがタスクを終了し、リッチクライアントサーバに通知します。
2. クライアントモジュールが終了します。
3. リッチクライアントサーバで、コンテキストが削除されます。

リッチクライアントが必要なくなったら、アンインストールを行って、ダウンロードされたモジュールやショートカットなどを削除することができます。アンインストールは、通常のソフトウェアのアンインストールと同様、コントロールパネルから行います。

5.10 リッチクライアントサーバのライセンスは？

リッチクライアントサーバは同時実行ユーザ数に対して課され、エンタープライズサーバとは異なるライセンスが必要です。

Web サーバシステムや、パーティショニングサーバシステムで使われるエンタープライズサーバ製品と、リッチクライアントサーバ製品とでは、異なるライセンスが必要となります。

エンタープライズサーバ製品では、同時実行可能スレッド数により、ライセンスが決められていました。このライセンスでは、システムを利用しているユーザ数とは直接は関係なく、むしろ、処理の負荷や必要とされるスループットなどをもとにして必要ライセンスが決まってきます。

それに対し、リッチクライアントサーバシステムでは、同時実行ユーザ数に対してライセンスが課されます。このライセンスでは、処理の負荷ではなく、システムを利用しているユーザ数によりライセンスが決まってきます。

ライセンスの基礎となるのは、同時利用の最大ユーザ数です。利用する可能性のあるユーザ数ではありません。例えば、システムを利用する可能性のある社員が 100 人いたとしても、常時利用しているのが 50 人程度で、他の 50 人は時々必要のあるときにだけ利用するのであれば、必要となるライセンス数は 75 くらいで済むかもしれません。ライセンス数の決定に当たっては、実際の運用場面をよく考察・シミュレーションして、適切な数を決定する必要があります。

また、一台の PC 上で複数のクライアントモジュールを起動して、同一アプリケーションを実行する場合には、1 ユーザとカウントされます。uniPaaS の並行実行の機能を利用して、複数の SDI ウィンドウを開いて同時実行させる場合にも、1 ユーザとカウントされます。

リッチクライアントサーバのライセンスは、エンタープライズサーバと同様、ライセンスサーバを利用します。

ライセンスの名前は、MGRIA1P1 です。(リッチクライアントサーバ製品のバージョンによりライセンス名が異なりますので、正確には、各製品のインストールガイドなどを参照してください)

ライセンスは、MAGIC.INI の LicenseFile および LicenseName パラメータに設定します。

MAGIC.INI 例

```
LicenseFile = 744@MyLicenseServer
LicenseName = MGRIA1P1
```

ここで、MyLicenseServer というのは、ライセンスサーバがインストールされ実行されているサーバ HW のホスト名です。もしリッチクライアントサーバと同一のサーバ HW 上でライセンスサーバが実行されている場合には、ここには localhost と書くことも可能です。

リッチクライアントサーバライセンスは、次のようなもので、これは、製品添付のライセンスマネージャにより、c:¥FlexLM¥license.dat ファイルに書き込まれます。(このライセンスはサンプルですので、このまま利用することはできません)

license.dat 例

```
FEATURE MGRIA1P1 MAGIC 1.800 01-jan-0 50 12312312312312312312 \
      VENDOR_STRING=PT=MGRIA,C=00FF,P=S,B1=N HOSTID=ANY \
      DUP_GROUP=NONE ISSUER=MAGICUSER SN=190000001
```

5.11 リッチクライアントサーバでは、Web サーバにどのような仮想ディレクトリが必要ですか？

次のようなファイルを公開するための仮想ディレクトリが必要となります。

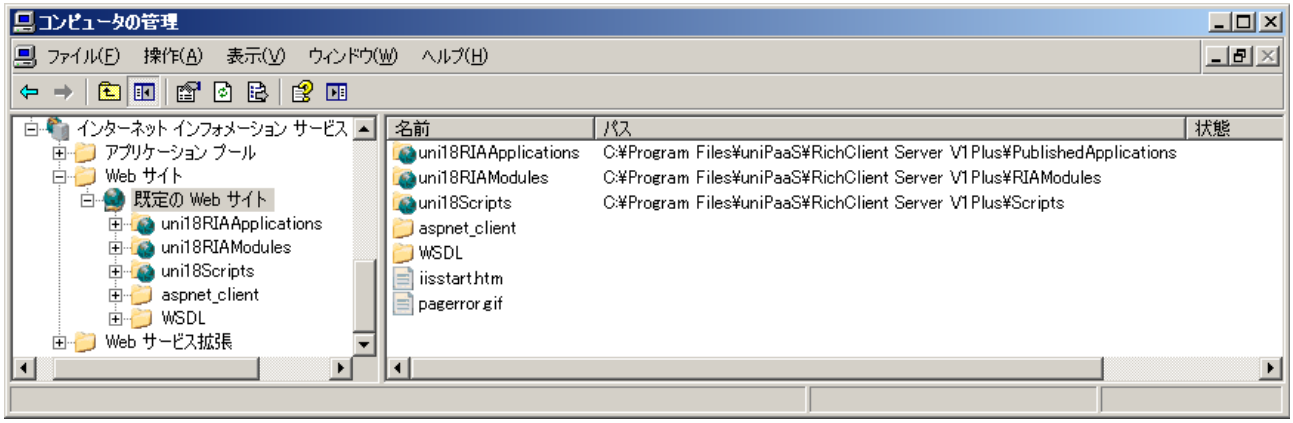
- インターネットリクエスタ
- クライアントモジュール
- リッチクライアントアプリケーション用リソース

リッチクライアントサーバ製品をインストールすると、エンタープライズサーバ製品とだいたい同じようなファイル類と各種の設定がセットアップされますが、リッチクライアント製品に固有のものもいくつかあります（下記）。

仮想ディレクトリ名 （※1）	物理ディレクトリ名 （※2）	用途
/uni18Scripts	Scripts	インターネットリクエスタがあります。
/uni18RIAModules	RIAModules	クライアントモジュールがあります。
/uni18RIAApplications	PublishedApplications	publish.html、.application ファイル、その他アプリケーションで使う静的ファイルがあります。

- 仮想ディレクトリ名にある「18」というのは、uniPaaS のバージョン番号（この場合、uniPaaS 1.8)ですので、uniPaaS 製品のバージョンにより変わります。例えば、uniPaaS 1.5 の場合には、/uni15Scripts などとなっています。
- 物理ディレクトリ名は、uniPaaS のインストールディレクトリを基準としたサブディレクトリ名です。

このうち、/uni18Scripts と /uni18RIAModules 以下にあるファイルは、uniPaaS Studio あるいはリッチクライアントサーバ 製品とともに提供され、インストーラがセットアップします。/uni18RIAApplication 下にあるファイルは、アプリケーションが利用するファイルですので、開発者が作成して適切に配置します。



バージョンによる違い:

- V10 SP4 では、PublishedApplications というディレクトリはなく、Projects ディレクトリが /Magic101RCProjects という仮想ディレクトリ名で公開されていました。
- uniPaaS1.5 以降では、セキュリティ保護のために、プロジェクトディレクトリ全体を公開するのではなく、別途ディレクトリ PublishedApplications を設けて、実行時に必要なファイルのみを公開するようになりました。
- V10 SP4 および uniPaaSV1 (Ver1.5) では、この他に RIA キャッシュディレクトリが必要でした。このディレクトリは、uniPaaS1.5 では 仮想ディレクトリ名 /uni15RIACache、物理ディレクトリ名 RIACache でした。
  uniPaaS1.8 からは RIA キャッシュディレクトリは仮想ディレクトリとして公開されなくなりました。ただし、物理ディレクトリはそのまま存在し、利用されています。
  これは、(1) セキュリティの向上のため(Web サーバを DMZ に配備するときに、ファイル共有のためにポートを開ける必要がなくなる)、(2) 設定の容易さのため、という二つの理由によります。

5.12 静的なコンテンツとアプリケーションサーバを別の Web サーバに分けることができますか？

リッチクライアント インターフェース ビルダで、別 Web サーバを指定します。

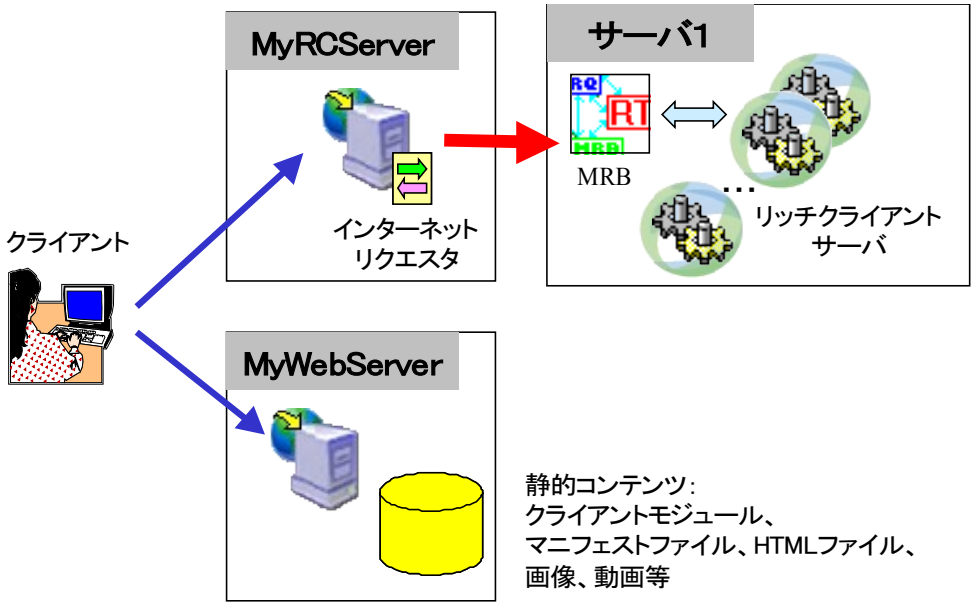
5.11 で説明した仮想ディレクトリは、すべて同一 Web サーバ上に定義することを前提としていましたが、同一 Web サーバにすることは必ずしも必要ではなく、セキュリティや負荷分散の目的で、静的なコンテンツ（マニフェストファイル、HTML ファイル、画像、動画など）を供給する Web サーバと、uniPaaS リッチクライアントサーバへのリクエストの受け口となる Web サーバとを分ける、という構成を考えることができます。

下記の仮想ディレクトリは、インターネットリクエスタを通して、リッチクライアントサーバへリクエストを発行するためのものですので、これを MyRCServer という名前の Web サーバで公開することにします。

仮想ディレクトリ名	用途
/uni18Scripts	インターネットリクエスタがあります。

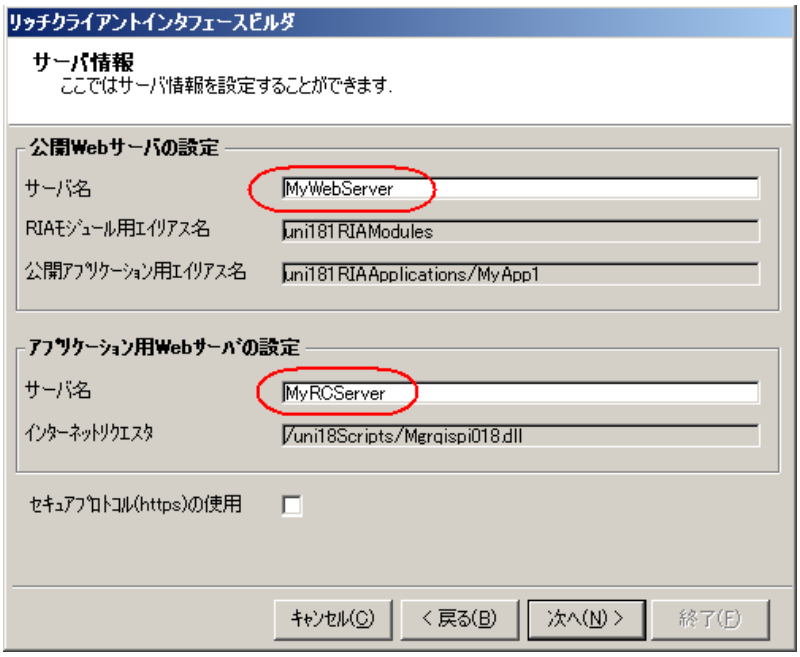
一方、以下の仮想ディレクトリは、静的コンテンツのみを供給しますので、これを MyWebServer という名前の Web サーバで公開するとします。

仮想ディレクトリ名	用途
/uni18RIAModules	クライアントにダウンロードするクライアントモジュール（uniRC.exe など）があります。
/uni18RIAApplications	publish.html、.application ファイル、その他アプリケーションで使う静的ファイルがあります。



このような構成にした場合には、アプリケーションの開発と配備において、次のように設定する必要があります。

- アプリケーションでは、静的コンテンツを参照する場合に、サーバ名（MyWebServer）を明示的に指定した URL を使って参照するようにします。
- リッチクライアント インターフェース ビルダでマニフェストファイルを作成する際に、「公開 Web サーバ」と「アプリケーション用 Web サーバ」として、それぞれ MyWebServer、MyRCServer を指定します。



本節の内容についてのより詳しい情報は、以下のリファレンスヘルプの項目を参照してください。

- コンポーネント > コンポーネントインタフェースビルダ > リッチクライアントインタフェースビルダ > [サーバ情報]ダイアログ

5.13 リッチクライアントサーバでもマルチインスタンスなどの構成は可能ですか？

もちろん可能です。

リッチクライアントサーバでも、エンタープライズサーバと同様、次のような構成が可能です。これにより、セキュリティ、スケーラビリティ、および可用性を向上させることができます。

- Web サーバを別サーバ HW 上で実行させる
- マルチインスタンスで実行させる
- 複数のサーバ HW 上に、リッチクライアントサーバエンジンを分散させて実行させる
- 代替ブローカにより二重化する
- ロードバランサーにより多重化する

ただし、リッチクライアントサーバにはコンテキストがサーバ中に作成されるので、エンタープライズサーバと異なり、次のような注意が必要です。

- ロードバランサを使う場合、リクエストとサーバとの紐付けが正しく行われるように、ロードバランサにセッション維持の設定を行う必要があります。（5.21「ロードバランサを使うとき、セッション維持のためにどのように設定しますか？」参照）。
- 万一、リッチクライアントサーバのインスタンスが動作停止してしまった場合、そのインスタンス中に存在していたコンテキストはすべて失われ、クライアント側はエラー終了となります（5.16「リッチクライアントサーバが動作停止した場合、何が起こりますか？」参照）

5.14 マルチインスタンスの場合、ライセンスはどのように割り振りますか？

MAGIC.INI の MaxConcurrentUsers に、各インスタンスで扱う最大ユーザ数を設定します。

基本的には、エンタープライズサーバの場合（3.12「ライセンスはどのように管理されますか？」）と同じ考え方で行います。ただし、次の違いがあります。

- エンタープライズサーバの場合には、ライセンスは同時実行可能最大スレッド数で課され、各インスタンスに割り当てるスレッド数は MAGIC.INI の MaxConcurrentRequests パラメータで設定しました。
- リッチクライアントサーバの場合には、ライセンスは同時実行可能ユーザ数で課され、各インスタンスに割り当てるユーザ数は MAGIC.INI の MaxConcurrentUsers パラメータで設定します。

より詳しくは、以下のようになります。

リッチクライアントサーバシステムをマルチインスタンスで動作させるときには、ライセンスは次のように消費されます。

- ライセンスはライセンスサーバで管理されます。
- リッチクライアントサーバのインスタンスが起動するときに、ライセンスが消費されます。
- 消費されるライセンス数は、MAGIC.INI の MaxConcurrentUsers に指定されている数値によります。
- MaxConcurrentUsers が 0（デフォルト）の場合には、その時点で許容されている最大ユーザ数すべてが割り当てられます。
- MaxConcurrentUsers が正数に設定されている場合には、その数値と、その時点で許容されている最大ユーザ数のうち、小さい方の数値分だけ、ライセンスが消費されます。
- リッチクライアントサーバのインスタンスは、割り当てられたライセンス数分だけのクライアントに対応することができます。
- ライセンス数が残ったら、別のインスタンスに割り当てることができます。

このことから、マルチインスタンスでリッチクライアントサーバシステムを構成する場合には、MaxConcurrentUsers を適切に設定して、すべてのインスタンスで負荷がなるべく均等になるように考慮しながら、全インスタンスの最大ユーザ数の合計が、ライセンスで許可されているユーザ数を超えないように設定します。

例1：50 ユーザライセンスを5インスタンス → MaxConcurrentUsers = 10
（5 インスタンス × 10 ユーザ = 50 ユーザ）

例2：200 ユーザライセンスを 2 台のサーバ HW に分割して、各サーバ HW では 4 インスタンスづつ実行 → MaxConcurrentUsers = 25（合計 8 インスタンス × 25 ユーザ = 200 ユーザ）

5.15 マルチインスタンスの場合、リクエストとコンテキストはどのように紐付けされますか？

MRB が判別して、適切なインスタンスにリクエストを送ります。

リッチクライアントサーバシステムを、マルチインスタンスで構成した場合には、特定のクライアントに対するコンテキストは、特定のリッチクライアントサーバのインスタンスに作成されるので、このクライアントからのリクエストは、すべて対応するコンテキストのあるリッチクライアントサーバのインスタンスに送られなければなりません。

クライアントからのリクエストが、対応するコンテキストのあるインスタンスに正しく送られるようにするためのメカニズムは、以下のように、コンテキスト ID を使って実現されています。

1. リッチクライアントサーバがコンテキストを作成した時、コンテキストにはユニークなコンテキスト ID が割り振られます（18 桁の数値）。
2. コンテキスト ID は、MRB に伝えられます。これにより、MRB はどのインスタンスがどのコンテキスト ID を保持しているのかを知ることができます。
3. また、コンテキスト ID は、クライアントモジュールの初期化処理の一環として、サーバからクライアントに伝えられます。
4. クライアントモジュールがリクエストを出す時には、常に、コンテキスト ID がリクエストと共に送られます。具体的には、クライアントモジュールからのリクエストは、HTTP の POST メソッドにより Web サーバに送られるのですが、その中のデータとして、CTX=（コンテキスト ID） というデータが含まれています。
5. MRB が POST データ中のコンテキスト ID を判別して、コンテキストを保持しているリッチクライアントサーバを選択します。

以下に、クライアントモジュールからのリクエストのデータ（HTTP POST による）の例を示します。ここで見るように、インターネットリクエスタ `/uni18Scripts/Mgrqispi018.dll` へ POST されたデータの中に、「**CTX=950200768720896**」という形でコンテキスト ID が入っているのが分かります。

POST データキャプチャ例：

```
POST http://MyRCServer/uni18Scripts/mgrqispi018.dll HTTP/1.1
uniRCglobalUniqueSessionID: BFEBFBFF000006FD_3224
Host: 192.168.75.150
Content-Length: 3057
Expect: 100-continue
Proxy-Connection: Keep-Alive

RICHCLIENT=Y&CTX=950200768720896&SESSION=9&DATA=%3cxml+id%3d%22MGDATA%22%3e
%0a+++%3ccontext+id%3d%22950200768 ・・（以下省略）
```

5.16 リッチクライアントサーバが動作停止した場合、何が起こりますか？

コンテキストは失われ、クライアント側はエラーで終了します。

コンテキストは、特定のリッチクライアントサーバのインスタンスのメモリ中に作成され、保存されています。このため、万一そのインスタンス（プロセス）が動作を停止した場合（異常終了、無限ループ、ハングアップなど）には、そのインスタンス中に保存されているコンテキストは失われます。

この時には、次のことが起こります。

- クライアント側は実行を継続できなくなります。具体的には、クライアントモジュールがサーバへリクエストを出したタイミングで、コンテキストが失われたことを検出し、エラーメッセージを出して、クライアントモジュールが終了します。
- コミットされていないトランザクションは、すべてロールバックされます。
- 中間データ（メモリテーブル、グローバル変数など）はすべて破棄されます。

このようなことがあるので、可用性向上のためには、マルチインスタンス構成が推奨されます。

例えば、50 ユーザをすべて1インスタンスで処理している構成と、10 ユーザずつ 5 インスタンスで処理している構成とを比較してみましょう。1インスタンス構成の場合には、このインスタンスが動作停止した場合、すべてのユーザのコンテキストが失われてしまいます。一方、5 インスタンス構成の場合には、1 インスタンスが動作停止したら、そのインスタンスで処理していた 10 ユーザ分のコンテキストは失われてしまいますが、他の 40 ユーザのコンテキストには影響がありません。後者の方が、運用の方法として好ましいでしょう。

それでは、「インスタンスは多いほど良いのか？極端には、1 インスタンスあたり 1 ユーザだけ割り当てるようにして、50 インスタンス立ち上げるのが良いのか？」ということにもなりますが、可用性だけを考慮すればその通りとなりますが、インスタンスが増えるとシステムに必要とされるメモリ量も増えますので、最大ユーザ数、HW のメモリ量などを勘案して、実際には適度なところでまとめるのが無難でしょう。

サーバの自動再起動の設定（6.4「uniPaaS サーバを自動的に起動させられますか？」参照）がされているとき、MRB がリッチクライアントサーバの異常終了を検知すると、MRB はリッチクライアントサーバを再起動します。

5.17 クライアントが動作停止・切断した場合、コンテキストはどうなりますか？

コンテキストタイムアウトの後、コンテキストは削除されます。

前節とは逆に、リッチクライアントサーバのインスタンスが正常に動作しているが、クライアントモジュールが動作を停止してしまった場合にはどうなるか？という点についてです。

クライアントモジュールが動作停止したと判断される理由には、以下のようなことが考えられます。

- クライアントモジュールが異常終了、無限ループ、ハングアップなどして、正常に動作が継続できなくなった。
- クライアント PC が動作停止した（Windows の終了、スリープ状態への移行、Windows の異常終了、バッテリ切れなど）
- クライアント PC とサーバとのネットワーク回線が切断された。このケースは、正確には、クライアントモジュールの動作停止ではないのですが、サーバから見るとクライアントモジュールからリクエストが来なくなってしまう状態となりますので、動作停止しているのと同じことになります。
- ユーザがクライアントモジュールを起動したまま、何も行わずに長時間放置しておいた。これも、正確には動作停止ではないのですが、上のケースと同様、リクエストがサーバへ来なくなるので、サーバとしては、クライアントが動作停止しているように見えます。

クライアントが動作停止した場合、サーバのコンテキストは適当なタイミングで破棄されなければなりません。

もし破棄せずに、永久に保持したままにしておくと、必要のないコンテキストがゴミとしていつまでもサーバインスタンスに残ってしまいますので、次のような問題が出てきます。

- サーバ HW のメモリを消費する。
- 実際には使っていないライセンスを消費したままになる。

後者は特に重大で、例えば、50 ユーザ分割り当てられたリッチクライアントサーバのインスタンスがあったときに、10 人のクライアントモジュールが上記のような理由で動作停止した場合、10 コンテキストが残されたままとなり、10 ユーザ分のライセンスが使われたままの状態になっています。従って、このインスタンスでは、50 ユーザ分割り当てられているにも関わらず、40 ユーザまでしか使えないということになってしまいます。

このため、リッチクライアントサーバには、「コンテキストタイムアウト」という機構により、不要なコンテキストと、それに伴なうサーバリソース(メモリ、DBMS のトランザクション等)がいつまでも残らないように、定期的に掃除をするようになっています。

コンテキストタイムアウトは、MAGIC.INI の ContextInactivityTimeout パラメータで、0.1 秒単位で指定できます。インストール直後のデフォルトでは、36000 に設定されていますので、36000 × 0.1 秒=60 分となっています。

リッチクライアントサーバは、各コンテキストについて、最後のリクエストを処理してからの時間を定期的にチェックしています。もし、最後のリクエストを処理してから、コンテキストタイムアウトの時間が経過しても次のリクエストが来なかった場合には、クライアントモジュールが動作を停止していると判断し、コンテキストタイムアウトを発生させます。

コンテキストタイムアウトが発生すると、以下のことが起こります。

- コンテキスト中のデータ（内部データ、メモリテーブルのデータ、グローバルパラメータ）はすべて破棄されます。
- コンテキスト中の、コミットされていないトランザクションはすべてロールバックされます。
- コンテキストは削除され、コンテキスト ID はなくなります。
- MRB にコンテキスト ID が消去されたことを通知します。
- そのユーザが消費していたライセンスが解放されます。

5.18 コンテキストタイムアウトの他に、コンテキストが消滅する事由がありますか？

コンテキストの強制削除によっても、コンテキストが消滅します。

コンテキストは、通常のリッチクライアントプログラムの終了（5.9「リッチクライアントプログラムの起動から終了までのサイクルは？」参照）、コンテキストタイムアウトの他に、強制削除によっても消滅することがあります。

コンテキストの強制削除は、次のような場合に発生します。

- リッチクライアントサーバのインスタンスの終了。異常終了の場合（5.17「クライアントが動作停止・切断した場合、コンテキストはどうなりますか？」参照）にはもちろん、管理者が意図的に終了させる場合（6.7「サーバを停止させるにはどうしますか？」参照）にも、その時点で残っていたコンテキストは強制削除されます。
- プログラムで、CtxKill 関数を使って、特定のコンテキストを強制削除させることができます。具体的な使い方については、リファレンスヘルプ 式エディタ → 関数ディレクトリ → CtxKill を参照してください。
- Broker モニタからも削除可能です。Broker モニタを起動し、メニューで 動作 → コンテキスト破棄 を選びます。

5.19 クライアントを長時間放置しても問題ありませんか？

コンテキストタイムアウトが発生し、エラーとなります。これを防ぐには、定期的にサーバへのアクセスを行います。コンテキストタイムアウトを 0（無制限）にするのは、一般には不適当です。

クライアントを起動したまま長時間放置しておくと、クライアントからリッチクライアントサーバへリクエストが行かないので、サーバがコンテキストタイムアウトを起こし、コンテキストが削除されてしまいます。

この状態では、クライアントモジュールはまだ生きているものの、対応するコンテキストがないので、実行の継続が不可能になってしまいます。

従って、クライアントモジュールからサーバへリクエストを発行したタイミングで、「タイムアウトのためこのセッションはクローズされました.」というエラーが発生し、クライアントモジュールが終了します。

リッチクライアントシステムにおいては、クライアントモジュールを長時間放置したままにしておくことは推奨されませんが、システムの要件上どうしても必要な場合には、クライアントモジュールから、定期的にダミーのリクエストをサーバに出すようにして、コンテキストタイムアウトが発生しないように設計します。

具体的には次のような方法が考えられます。

- 長時間放置されるプログラム（メニュー画面など）、あるいはメインプログラムで、タイマーイベントを定義します。
- タイマーの間隔は、コンテキストタイムアウトよりも少し短い時間にします。例えば、コンテキストタイムアウトが 10 分ならば、5～8 分程度がいいでしょう。あまり短すぎると、ダミーのリクエストが必要以上に頻繁に出るようになるので、好ましくありません。
- タイマーイベントの中で、サーバ側の処理を行うオペレーションを定義します。例えば、ダミーのバッチタスク（何もせずに、すぐに終了する）を呼び出すコールコマンドなどです。

コンテキストタイムアウトの設定を 0（無制限）にすると、クライアント側で放置しておいてもコンテキストタイムアウトが発生しなくなります。しかし、本当にクライアント側やネットワークでの障害が起こった場合に、残されたコンテキストがサーバ終了時まで解放されません。従って、サーバのリソース（メモリ）を占有する、ライセンスを消費し続ける、などの問題が起こります。コンテキストを解放するには、サーバインスタンスを終了させなければなりません。このため、一般的にはコンテキストタイムアウトを 0 にすることは不適当です。

5.20 リッチクライアントタスクから長時間のバッチタスクを起動できますか？

長時間のバッチは、パーティショニングサーバに非同期で実行を委託するべきです。

リッチクライアントタスクがバッチタスクを呼び出したとき、バッチタスクはリッチクライアントサーバ上で実行されます。この間バッチタスクが終了するまで、リッチクライアントタスクの実行は待ち状態になっています。すなわち、クライアントモジュールは、砂時計を出したまま、反応なしとなっています。

ごく短時間で終了するバッチタスクならば、このような待ち状態も短いので操作上支障はないでしょう。

しかし、大量印刷とか大量データの集計など、長時間実行するバッチタスクをリッチクライアントタスクから実行すると、その間クライアント側は待ち状態、反応なしになり、操作性が悪くなるので好ましくありません。また、ネットワークの通信には各種のタイムアウト（TCP/IP 自体、HTTP プロトコル、途中に入るプロキシサーバ、セキュリティソフト、ロードバランサなどのタイムアウト）があるので、長時間待ち状態にしておくのはトラブルの元となり易く、極力避けるべきです。

どれくらいが「短時間」あるいは「長時間」なのかというと、環境により異なって一概には言えないのですが、操作性なども考慮した目安として、リッチクライアントでは1分を超える待ちが発生する場合には「長時間」と見るべきでしょう。

長時間の待ちを防ぐためには、パーティショニングサーバ（第 4 章「パーティショニングサーバ」を参照）を使って、コール リモート コマンドによりバッチタスクを実行させ、処理を委託して行わせるようにしてください。

このときに、コール リモート コマンドの「ウェイト」パラメータは No に設定し、バッチタスクを非同期で実行するようにしてください。こうすれば呼び出し側のリッチクライアントタスクは待ち状態になることなく、次の処理を進めることができます。

パーティショニングサーバを使うと、別のサーバ HW でバッチタスクの処理を行わせることができます。従って、負荷の大きい印刷処理やファイル出力などを伴なう場合には、パーティショニングサーバによる方法はサーバの負荷分散の観点からも好ましい構成であると言うことができます。

5.21 ロードバランサを使うとき、セッション維持のためにどのように設定しますか？

HTTP ヘッダ uniRCglobalUniqueSessionID を使って、セッションを判別させます。

リッチクライアントシステムでは、特定のクライアントに対するコンテキストは、特定のサーバインスタンスに作成・保管されるので、そのクライアントからのリクエストは、コンテキストを保持しているサーバインスタンスに送られなければなりません。

リッチクライアントのコンテキストには一意なコンテキスト ID がつけられ、コンテキスト ID を通信データに入れることにより、MRB に判断させて、リクエストが正しいサーバインスタンスに送られるようになっています。

しかし、ロードバランサがある場合には、特別な設定が必要です。クライアントからのリクエストはいったんロードバランサが受け入れますが、デフォルトの設定ではロードバランサはリクエストをランダムに（あるいはロードバランサによる判定方法により）リクエストを Web サーバに転送するので、リクエストがどの Web サーバ→MRB に行くのかわかりません。間違った Web サーバ → MRB にリクエストが転送されてしまったら、MRB はコンテキスト ID がわからず、「Context not found」のエラーとなってしまいます。

これを防ぐためには、ロードバランサにも uniPaaS リッチクライアントのセッションを認識させる必要があります。

uniPaaS リッチクライアントでは、ロードバランサにセッションを認識させるために、次のいずれかの方法を利用します。

- クライアント IP アドレスにより
- HTTP ヘッダにより

5.21.1 クライアント IP アドレスによる方法

ロードバランサの基本機能として、クライアント IP アドレスによりクライアントを識別する機能がありますので、この機能を利用してセッション維持を行うことができます。

クライアントとロードバランサの間にルータやプロキシサーバなどがある場合には、ロードバランサが認識する IP アドレスはルータやプロキシサーバの IP アドレスとなり、クライアント PC の IP アドレスではないので、有効に負荷分散を行うことができないことがあります。

5.21.2 HTTP ヘッダによる方法

uniPaaS リッチクライアントモジュールは、リクエストを発行する際に、uniRCglobalUniqueSessionID という名前の HTTP ヘッダを付加しています。以下は、クライアントモジュールが発行する POST リクエストのキャプチャ例です。

```
POST http://MyRCServer/uni18Scripts/mgrqispi018.dll HTTP/1.1
uniRCglobalUniqueSessionID: BFEBFBFF000006FD_3224
Host: 192.168.75.150
Content-Length: 3057
Expect: 100-continue
Proxy-Connection: Keep-Alive

RICHCLIENT=Y&CTX=950200768720896&SESSION=9&DATA=%3cxml+id%3d%22MGDATA%22%3e
%0a+++%3ccontext+id%3d%22950200768 ・・（以下省略）
```

この HTTP ヘッダは、各クライアントモジュールのプロセス（uniRC.exe）ごとにユニークな値であり、クライアントからの HTTP リクエストには、必ずこの HTTP ヘッダが付加されてサーバ側に送信されます。

従って、ロードバランサでは、この HTTP ヘッダによってクライアントを識別させるよう、設定しておきます。

具体的な設定方法はロードバランサにより異なりますが、例えば、Pound というロードバランサでは、次のような設定を行います。

```
Session
   Type     Header
    ID       "uniRCglobalUniqueSessionID"
    TTL      300
End
```

同様な設定を、利用しようとしているロードバランサでも行うようにしてください。

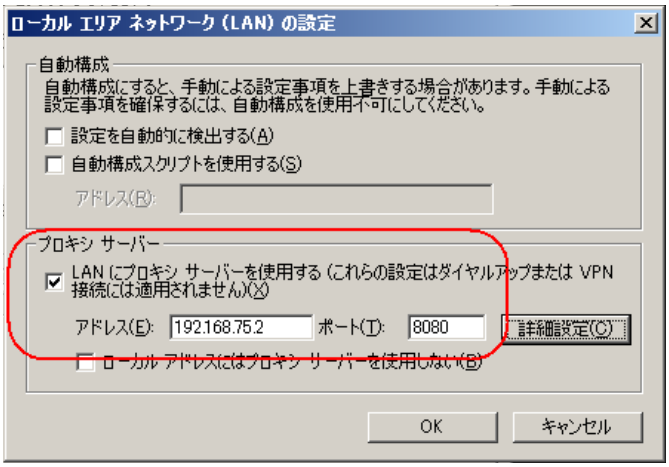
この方法によれば、途中にルータやプロキシサーバがあっても、有効に負荷分散が可能であり、かつ、セッション維持も正しく行うことができるようになります。

Pound というのはオープンソースのソフトウェアロードバランサであり、Unix/Linux 系の OS 上で動作します。ここでは、オープンソースのフリーソフトウェアであり、入手・テストが簡単なために例として紹介しました。このソフトウェアの利用を特に推奨しているわけではありません。高機能・高性能なロードバランサが各種出回っていますので、運用のニーズに応じ選択してください。

セッションを認識させるにはコンテキスト ID を利用することが考えられますが、コンテキスト ID はクライアントからの POST データの中に埋め込まれている形になっているため、ロードバランサが POST データを解析してコンテキスト ID を見つけ出す、というような処理が必要になってしまいます。これは効率的ではないし、利用しているロードバランサがそのような機能を持っているとは限りませんので、一般には利用できません。

5.22 プロキシサーバを使う場合に注意することがありますか？

ClickOnce は Windows 統合認証だけに対応していることに注意する必要があります。



uniPaaS リッチクライアントのクライアントモジュールは、Windows の「インターネット オプション」のプロキシサーバの設定（右図）を認識して、利用します。この他には特に設定は必要ありません。

プロキシサーバを利用する際には、次の点に留意する必要があります。

プロキシサーバが認証を要求する場合

セキュリティの厳しい環境では、クライアント PC が不正な利用者によって利用されるのを防ぐために、プロキシサーバが ID/パスワードの認証を要求するように構成されていることがあります。

uniPaaS リッチクライアントが利用している ClickOnce では、プロキシサーバの認証機構として、Windows の統合認証しかサポートされていません。従って、プロキシサーバの認証方式が基本認証等の方法だと、リッチクライアントプログラムを起動する段階でエラーとなって実行できないことになります。

これは ClickOnce の仕様事項であり、回避する方法はありません。プロキシサーバで認証を行う場合には、Windows 統合認証の方式で行う必要があります。

ロードバランサを使う場合

サーバ側でロードバランサを使う場合、単純なクライアント IP アドレスによるセッション維持機能を使っていると、有効に負荷が分散されないことがあります。この問題については、5.21「ロードバランサを使うとき、セッション維持のためにどのように設定しますか？」を参照してください。

バージョンによる違い:

Magic eDeveloper V10 のリッチクライアントでは Java を使っていたので、Java CPL でプロキシサーバを設定する必要がありました。uniPaaS V1 からは、クライアントモジュールとして .NET のモジュールを使うようになったので、インターネットでの設定がそのまま利用されます。

5.23 リッチクライアントシステムでは、どのようなキャッシュが使われますか？

サーバ側とクライアント側でキャッシュが使われます。

uniPaaS リッチクライアントシステムでは、各種のキャッシュが利用されます。これは、サーバからファイルを一旦ダウンロードしたら、同じファイルは繰り返しダウンロードしないようにして、パフォーマンスの向上と、ネットワークトラフィックの減少を目的としたものです。

通常、キャッシュの管理は uniPaaS や Windows が自動的に行うもので、ユーザが意識する必要はないのですが、ハードディスクの領域の節約や、セキュリティ上の考慮、あるいはシステムの安定稼働などのために、適当なタイミングで掃除することも、良い習慣と思われます。また、システムのアップグレードや、ネットワークおよびシステムの構成変更などの場合、キャッシュ管理がうまくできなくて動作がおかしくなることも間々あります。このようなときに、システムをリセットする意味でも、キャッシュの掃除を行いたいことがあります。

ここでは、どこにどのようなキャッシュが使われているのかについて、ファイルとしてハードディスクに保存されるものをリストアップします。

5.23.1 サーバ側

リッチクライアントサーバの稼動しているサーバマシンには、「動作環境」ダイアログ →「アプリケーション」タブ →「リッチクライアントのキャッシュパス」で指定されているディレクトリに、キーボード定義、色定義、フォント定義、メニュー定義、タスク定義などのファイルが作成されます。これらのファイルは、インターネットリクエスタを介して、クライアント側にダウンロードされます。

このキャッシュをクリアしたい場合には、通常のファイル削除コマンド（DEL コマンド、あるいは Explorer より削除）により、ディレクトリ中の全ファイルを削除して構いませんが、ディレクトリ自体は残しておいてください。また、削除に先立って、リッチクライアントサーバは停止しておいてください。リッチクライアントサーバが稼働中にキャッシュファイルを削除すると、クライアント側においてエラーになります。

5.23.2 クライアント側

クライアント側では、次の3つのソフトウェアモジュールがそれぞれキャッシュが使われます。

- ClickOnce
- uniPaaS クライアントモジュール（uniRC.exe）
- Web ブラウザ（Internet Explorer）

以下に、それぞれについて簡単に説明します。

ClickOnce

Windows の ClickOnce の一般的な機能として、サーバ側から必要ソフトウェアモジュールやマニフェストファイル等をダウンロードして、ローカル HD にセーブしておき、2回目以降からは毎回ダウンロードせずにする機能があります。uniPaaS のリッチクライアントでも ClickOnce を利用しているので、このキャッシュを利用しています。

このキャッシュは、以下に示す Windows のログインユーザ毎のアプリケーションディレクトリに作成されます。

- Windows XP では、C:¥Documents and Settings¥（ユーザ名）¥Local Settings¥Apps¥2.0
- Windows Vista 以降では C:¥Users¥(ユーザ名)¥AppData¥Local¥Apps¥2.0

このキャッシュをクリアしたい場合には、Windows のコントロールパネルより、「プログラムの追加と削除」によっ

て、アンインストールするのが一番安全な方法です。

また、簡単には、上記ディレクトリ以下のファイルを Explorer などから直接削除してしまうことも可能です。ただし、この場合には、ClickOnce 機能を使っているアプリケーションはすべて（uniPaaS リッチクライアント以外のアプリケーションも）削除してしまうことになるので、十分に気をつけて行ってください。

uniPaaS クライアントモジュール

uniPaaS クライアントモジュールでは、実行しているタスクの定義、キーボード定義、色定義、フォント定義、メニュー定義などのファイルをサーバからダウンロードしますが、これらのファイルはクライアントのローカル HD に保存され、2回目以降は同じファイルをダウンロードしないようにしています。

このファイルは、各ユーザ毎のテンポラリディレクトリの下に uniRIACache というサブディレクトリを作成し、更にその下にサーバ名によりサブディレクトリを作成して、セーブされます。

- Windows XP では、C:¥Documents and Settings¥(ユーザ名)¥Local Settings¥Temp¥uniRIACache
- Windows Vista 以降では、C:¥Users¥（ユーザ名）¥AppData¥Local¥Temp¥uniRIACache

このキャッシュをクリアするには、エクスプローラなどから、上記ファイルを直接削除して行ないます。このときには、削除に先立って、リッチクライアントのクライアントを終了させてください。リッチクライアントの実行中に削除するとエラーが発生します。

Web ブラウザ（Internet Explorer）

Web ブラウザのキャッシュには、.publish.html ファイルや マニフェストファイルなどがセーブされます。これらのファイルを更新した場合には、Web ブラウザのキャッシュもクリアした方がよいでしょう。

このキャッシュをクリアするには、Web ブラウザのキャッシュ削除機能を利用して行ないます。ファイルをエクスプローラなどから直接削除することはお勧めしません。

6 共通事項

本章では、uniPaaS サーバ製品に共通する設定や構成の方法について説明します。

共通事項についての説明なので、エンタープライズサーバとリッチクライアントサーバの総称として、uniPaaS サーバという名称を使います。

6.1 同一 ECF を別アプリケーション名で実行させることはできますか？

MAGIC.INI で ApplicationPublicName パラメータを使います。

uniPaaS で開発するアプリケーションの管理と運用の便のために、同一のアプリケーション（ECF ファイル）を別のアプリケーション名として実行させたいことがあります。

例えば、リッチクライアントシステムとしてエンドユーザには利用させるが、印刷や集計などのバッチプログラムはパーティショニングサーバで実行させたい、というような場合、開発時には同一プロジェクトとしてモデル、テーブル、基本プログラムなどを共用し、実行時には、リッチクライアントサーバとパーティショニングサーバと別に運用するため、混乱を避けるために別アプリケーション名として区別したい、というような場合です。

通常、アプリケーション名は ECF ファイルのプロジェクト名なのですが、プロジェクト名は Studio によりプロジェクトを最初に作成した時に定義され、その後変更することはできず、ECF ファイルを作成する時にも、内部に書き込まれるので変更することができません。ECF ファイルのファイル名を変更しても、プロジェクト名は変わりません。

従って、そのまま同じ ECF ファイルを使ってリッチクライアントサーバとエンタープライズサーバを起動すると、どちらも同じアプリケーション名として実行されてしまうことになります。

これを防ぐには、MAGIC.INI で ApplicationPublicName パラメータを使うことにより、アプリケーションの別名を与えることができます。

uniPaaS サーバで、ApplicationPublicName が指定されている場合、実行されるアプリケーション名は、そこで指定された名前が使われ、ECF ファイルに書き込まれているプロジェクト名とは異なる名前にすることができます。

例えば、MyApp1 という名前のプロジェクト名を持つ ECF ファイル MyApp1.ecf を実行する際に、次のような MAGIC.INI ファイルを使ったとすると、このアプリケーションは MyApp2 という名前で実行されます。

MAGIC.INI ファイル例

```
StartApplication = Project¥MyApp1¥MyApp1.ecf
ApplicationPublicName = MyApp2
```

従って、リッチクライアントサーバとして起動する際には、.publsh.html の記述は次にようになります。

```
・・・
<body onload="initialize()">
   <xml id="rcExecProps">
      <properties>
        <property key="protocol" val="http"/>
        <property key="server" val="MyRCServer"/>
        <property key="requester" val="uni18Scripts/mgrqispi018.dll"/>
        <property key="appname" val="MyApp2"/>
        <property key="prgname" val="top"/>
・・・
```

6.2 ライセンスはいつ消費されますか？

ライセンスサーバからは、インスタンスの起動時にライセンスが消費されます。
各インスタンスに割り当てられたライセンスは、スレッドまたはユーザが起動されるたびに消費され、終了するたびに解放されます。

ライセンスの消費とについて考えるとき、

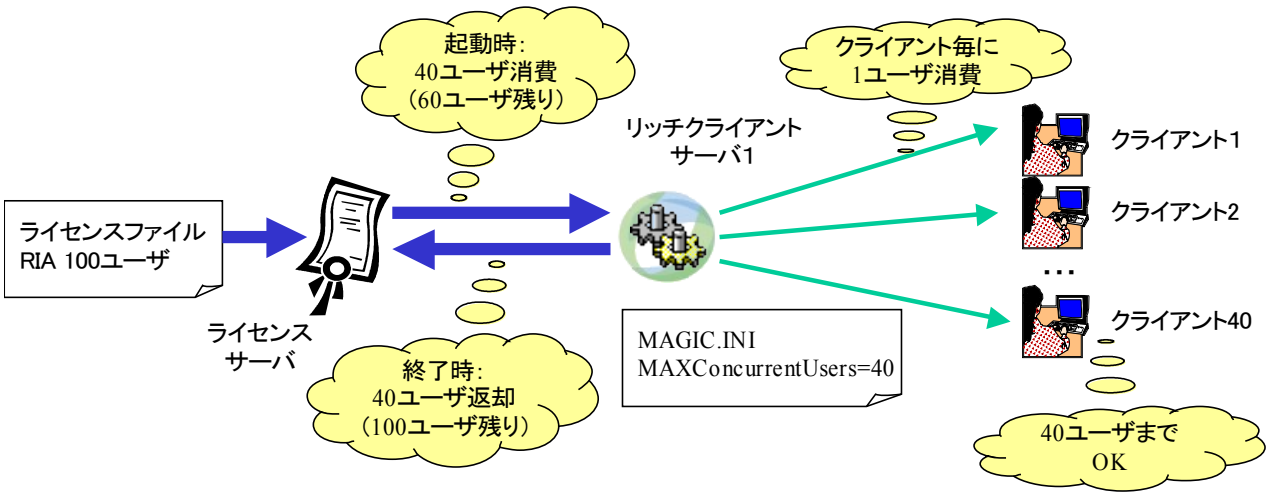
- ライセンスサーバから、各インスタンスにライセンスを割り当てる
- 各インスタンスに割り当てられたライセンスが、各スレッドあるいはユーザに割り当てられる

という2段階を考える必要があります。

ライセンスサーバから各インスタンスにライセンスを割り当てるのは、インスタンスの起動時に行われます。より正確には、インスタンスの初期化が終了し、アプリケーションがオープンされた瞬間に行われます。

ここで、各インスタンスに設定された MaxConcurrentRequests（エンタープライズサーバの場合）、あるいは MaxConcurrentUsers（リッチクライアントサーバの場合）に応じたライセンス数が、ライセンスサーバからインスタンスに割り当てられます。

以下に、100 ユーザのリッチクライアントサーバ ライセンスを例にとって説明します。



100 ユーザのリッチクライアントライセンスを購入した場合、ライセンスサーバは最大 100 ユーザまでのライセンスを管理します。

もしリッチクライアントサーバが起動して、MaxConcurrentUsers が 40 だった場合、ライセンスサーバは 40 ユーザ分のライセンスをインスタンスに付与します。従って、ライセンスサーバには 60 ユーザのライセンスが残されており、他のインスタンスに割り当てることができます。

このとき、ユーザの有無、リクエストの有無に関わらず、インスタンスが起動しているだけで、ライセンスサーバのライセンスは消費されます。

このリッチクライアントサーバでは、40 ライセンスが利用可能となっていますが、ここでクライアントがリッチクライアントタスクを立ち上げると、そのうち1ユーザ分が消費され、残り 39 ユーザが利用可能な状態になります。

同様にユーザが次々とクライアントを立ち上げると、それに応じた数のライセンスが消費されていきます。最後に 40 クライアントが立ち上がると、このインスタンスに割り当てられたライセンスはすべて消費され、それ以上クライントを立ち上げることができなくなります。

ここで、あるユーザがリッチクライアントタスクを終了すると、消費されていたライセンスが解放され、39 ユーザが使用中、1 ユーザが利用可能な状態になります。

このように、インスタンスに割り当てられたライセンスは、クライアントの起動・終了に伴ない、消費・解放されま

す。

もし、このインスタンスが終了すると、このインスタンスに付与されていた 40 ユーザ分のライセンスが解放され、ライセンスサーバに戻されます。こうなると、ライセンスサーバは再び 100 ユーザ分のライセンスを保持することになります。

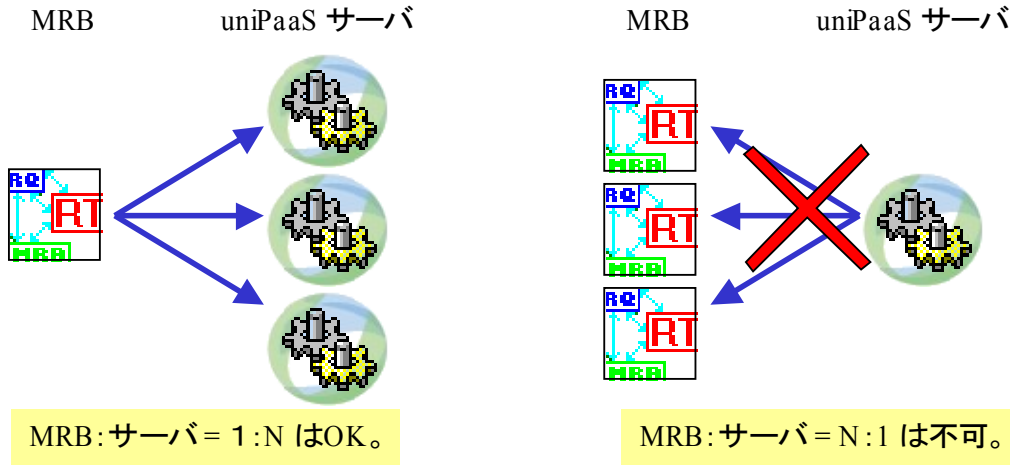
以上の例はリッチクライアントサーバの場合ですが、エンタープライズサーバの場合も全く同じです。エンタープライズサーバの場合に異なる点は、ライセンスはユーザ数ではなく同時実行可能な最大スレッド数でカウントされ、ライセンスサーバが各インスタンスに付与するライセンス数も最大スレッド数であり、各インスタンスがカウントするのも、実行中のスレッドである、という点です。

すなわち、あるリクエストが送られた時に、ライセンスが 1 スレッド消費され、そのリクエストが終了した時にそのスレッドライセンスは解放されます。

6.3 ひとつの uniPaaS サーバのインスタンスを、複数の MRB に登録することはできないのでしょうか？

できません。MRB とエンタープライズサーバとの関係は、1 対 N です。

一つの MRB のもとに、複数の uniPaaS サーバ（インスタンス）を管理するように構成することは可能です。
その反対に、一つの uniPaaS サーバインスタンスを、複数の MRB に関連づけさせる、ということはできません。
すなわち、MRB と uniPaaS サーバインスタンスとの関係は 1 対 N であり、N 対 1 とすることはできません。これは、現行の uniPaaS サーバ製品の仕様です。



MRB：サーバ＝１：N はOK。

MRB：サーバ＝ N：1 は不可。

6.4 uniPaaS サーバを自動的に起動させられますか？

MRB の自動起動機能を使って、uniPaaS サーバを自動起動させることができます。

サーバ運用の便のため、Windows サーバがブートした時に、MRB をサービスとして起動し、さらに、uniPaaS サーバインスタンスも自動起動するように設定すると便利です。

MRB を Windows のサービスとして自動起動させるのは、uniPaaS サーバ製品のインストーラがデフォルトで設定するので、管理者が特別なことをする必要はありません。

MRB が起動した後、uniPaaS サーバのインスタンスを自動起動させるには、MRB の自動起動機能を使い、以下のように設定ファイル MGRB.INI ファイルに起動すべきインスタンスについての情報を定義します。

MGRB.INI ファイルは、デフォルトで MRB モジュール（MGRQMRB.exe）と同じディレクトリにあるものが読み込まれます。

MRB の起動時に、コマンドラインパラメータで /INI=(設定ファイル名）と指定することもできます。

MRB の自動起動機能は、MGRB.INI の[APPLICATIONS_LIST]に、以下のように設定します。

MGRB.INI ファイル例

```
[APPLICATIONS_LIST]
エントリ名=<コマンド>,[<作業フォルダ>],[<ユーザ名>], [<パスワード>],[<実行回数>],[<最大数>]
```

この各パラメータの意味は、以下の通りです。

パラメータ	意味	例
エントリ名	適当な名前。半角英数字のみを使うようにしてください。 アプリケーション名と同じにすると管理上わかりやすいですが、同じである必要はありません。	MyApp1
コマンド	起動するコマンド(exe モジュール名)と、コマンドラインパラメータ。 通常は、uniPaaS サーバエンジンを指定しますが、それ以外のどんなコマンドでも指定できます。	
作業フォルダ	<コマンド>が実行される作業フォルダ。 省略可で、省略時には Windows の System32 フォルダとなります。	
ユーザ名、 パスワード	<コマンド>を実行されるユーザ名。<コマンド>の実行時のアクセス権限を決めます。 省略可で、省略時には MRB と同じ ローカルシステムアカウント SYSTEM となります。	

実行回数	MRB の起動時に、このコマンドを何回繰り返して実行するかを指定します。 uniPaaS サーバをマルチインスタンスで起動する場合には、ここに、立ち上げるインスタンス数を指定します。 0 を指定すると、MRB の起動時にこのコマンドは起動されませんが、管理者がエントリ名を指定して起動させることができます。(6.5「MGRB.INI に定義されたコマンドを手動で起動することもできますか？」参照)	2 (2 インスタンスが起動する)
最大数	0 を指定すると、無制限となります。	

6.5 MGRB.INI に定義されたコマンドを手動で起動することもできますか？

次のような方法で起動できます。

- コマンドラインリクエスタを使う
- MRB のツールバーアイコンのメニューから選択する
- ブローカモニタのメニューから起動する
- RqExe()関数を使う
- 他の MRB から起動する

MGRB.INI の [APPLICATIONS_LIST] に定義されているコマンドは、次のいずれかの方法により起動させることができます。

コマンドラインリクエスタにより

エントリ名を指定して、以下のようにコマンドラインリクエスタ MGRQCMDL.EXE を実行すると、指定されたエントリのコマンドが実行されます。(実際には 1 行で書きます)

```
Mgrqcmdl.exe -EXE=(エントリ名) HOST=(MRB ホスト名) -PORT=(ブローカポート番号)
    -PASSWORD=(パスワード)
```

ここで、各パラメータの意味は以下の通りです。

パラメータ	意味	例
エントリ名	MGRB.INI の[APPLICATION_LIST]で定義されているエントリ名	MyApp1
MRB ホスト名	MRB が動作している HW のホスト名。省略可で、省略時には localhost となります。	MRBServer
ブローカポート番号	MRB のブローカポート番号。 省略可で、省略時にはデフォルトポート番号 5215 となります。(ただし、デフォルトポート番号は、uniPaaS のバージョンごとに異なることがあります)	5215
パスワード	MGRB.INI に password で指定されている、MRB のパスワード。	Password (通常は、より強固なパスワードを推奨)

MRB のツールバーアイコンのメニューから

MRB のツールバーアイコンのメニューから、「Start an Enterprise Server」を選び、そのサブメニューからエントリ名を選択すると、コマンドが実行されます。

1. MRB が起動しているときには、ツールバーアイコンが表示されます。これを右クリックすると、MRB のポップアップメニューが表示されます。
2. その中の「Start an Enterprise Server」を選ぶと、MGRB.INI に定義されているエントリ名の一覧がサブ

メニューで表示されます。

3. この中から、実行したいエントリ名を選択すると、そのコマンドが実行されます。

ブローカモニタから

次のように、ブローカモニタから起動させることもできます。

1. ブローカモニタを起動して、メニュー「動作 → アプリケーションサーバ起動」を選択します。
2. アプリケーションサーバ名として、エントリ名を指定します。
3. 「起動」ボタンを押すと、エントリ名に対応したコマンドが実行されます。

RqExe 関数により

uniPaaS の組み込み関数 RqExe() を使って、uniPaaS のプログラムから、MGRB.INI に登録されているコマンドを実行することもできます。

RqExe 関数の構文は、次のようなものです。

RqExe(*サービス/サーバ名, 実行登録名, 引数, Supervisor パスワード*)

ここで、各パラメータの意味は以下の通りです。

パラメータ	意味	例
サービス/サーバ名	サーバテーブルに定義されているサーバ名。 あるいは、サービステーブルに登録されているサービス名	Default Server Default Service
実行登録名	MGRB.INI に登録されているエントリ名です。	MyApp1
引数	起動時に、コマンドに与える引数です。	/StartApplication=MyApplication.ecf
Supervisor パスワード	MGRB.INI に設定されている、MRB のパスワード	Password（実際にはより複雑なパスワードを推奨します）

他の MRB から

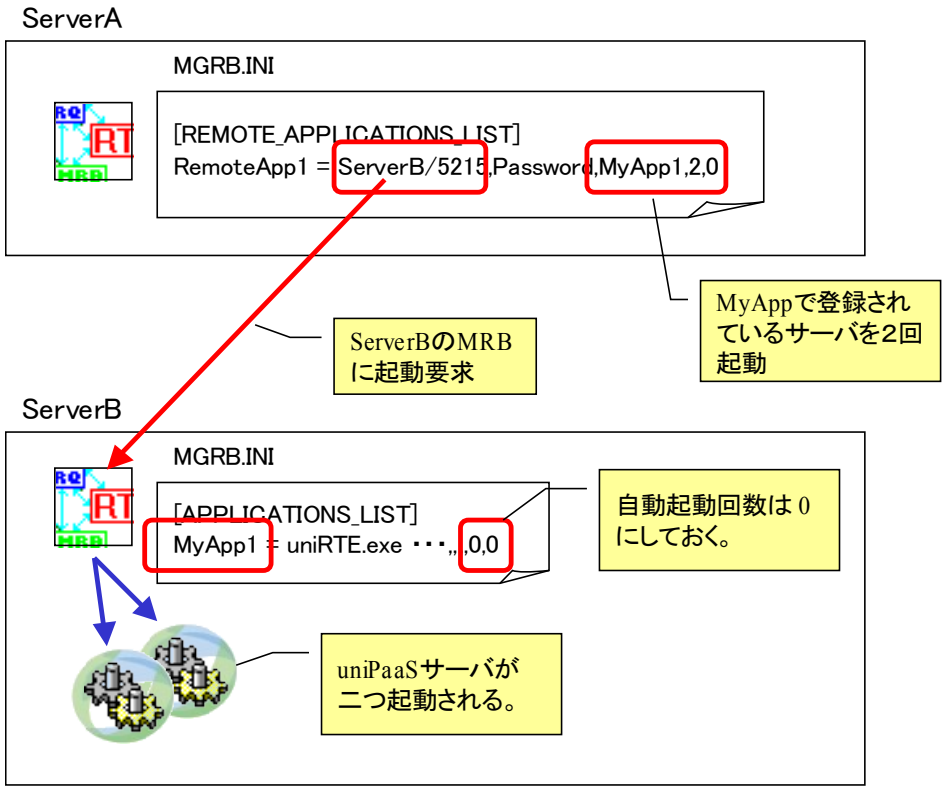
MRB が複数ある場合、ある MRB の MGRB.INI で定義されているコマンドを、別の MRB から起動させることもできます。これについては 6.6「MRB のリモート起動機能とは何ですか？」を参照してください。

6.6 MRB のリモート起動機能とは何ですか？

複数の MRB がある場合に、ある MRB の MGRB.INI に定義されているコマンドを、別の MRB から起動させることのできる機能です。[REMOTE_APPLICATIONS_LIST] を使います。

二つのサーバ HW 、サーバ A(ServerA) とサーバ B(ServerB) があって、それぞれに MRB がある場合に、サーバ B の MRB の[APPLICATIONS_LIST] に定義されているコマンドを、サーバ A の MRB から起動することができます。

この場合、サーバ A の MRB はサーバ B の MRB に起動の要求を出し、サーバ A の MRB は要求に応じて、コマンドを起動します。そのコマンドはサーバ B 上で実行されます。(下図参照)



サーバ A (起動要求する側) の設定

サーバ A (起動要求する側) の MGRB.INI には、次のように[REMOTE_APPLICATIONS_LIST] セクションを指定します。

[REMOTE_APPLICATIONS_LIST] の形式：(「エントリ」の行は、実際には1行で書きます)

```
[REMOTE_APPLICATIONS_LIST]
エントリ = (サーバ)/(ブローカポート),<パスワード>,<リモートエントリ名>,
    <実行回数>,<最大実行回数>
```

ここで、各パラメータの意味は以下の通りです。

パラメータ	意味	例
エントリ	適当な名前。半角英数字のみを使うようにしてください。 アプリケーション名と同じにすると管理上わかりやすいですが、同じである必要はありません。	RemoteApp1
サーバ	起動要求を出す MRB が実行されているサーバのホスト名、あるいは IP アドレス。	ServerB
ブローカポート	起動要求を出す MRB が待ち受けているブローカポート番号	5215
パスワード	指定した MRB の MGRB.INI で定義されているパスワード	Password（実際にはより複雑なパスワードを推奨します）
リモートエントリ名	別 MRB の[APPLICATIONS_LIST] に定義されているエントリ名	MyApp1
実行回数	MRB の起動時に、指定した別 MRB に対して、起動要求を送信する回数。	2（2 回、起動要求が出される）
最大数	0 を指定すると、無制限となります。	

このように設定しておくと、サーバ B の MRB が起動したときに、<サーバ>上で実行している MRB に対して、<リモートエントリ名> で指定されているコマンドを、<実行回数>だけ起動するよう、要求します。<実行回数> が 0 に設定されていた場合には、MRB の起動時には<サーバ>上の MRB に対して、起動要求を出しません。

例えば、サーバ A の MGRB.INI の[REMOTE_APPLICATIONS_LIST] セクションが以下のようになっていた場合：

サーバ A の MGRB.INI の[REMOTE_APPLICATIONS_LIST] セクション例：

```
[REMOTE_APPLICATIONS_LIST]
RemoteApp1 = ServerA/5215,Password,MyApp1,2,0
```

サーバ A の MRB が起動した時には、ServerB の MRB に対して、MyApp1 というエントリ名で定義されているコマンドを、2 回実行するように、要求を出します。

サーバ B（コマンドを実行させたい側）の設定

サーバ B（コマンドを実行させたい側）では、次の例のように、MGRB.INI に次のようにコマンドを定義します。

サーバ B の MGRB.INI の [APPLICATION_LIST]の例 ：

```
[APPLICATIONS_LIST]
MyApp1 = uniRTE.exe /DeploymentMode=B
   /StartApplication=d:\magic\180\Projects\MyApp\MyApp.ecf,D:\Magic\180SP1,,,0,0
```

起動手順

このような構成の場合、次のような順序で、サーバ B 上で uniPaaS サーバが起動されます。

1. サーバ B では、サーバ A に先立って、MRB を起動しておきます。
   このとき、APPLICATION_LIST の MyApp1 が読み込まれますが、自動起動回数が 0 になっているので、uniPaaS サーバは起動しません。

2. サーバ A の MRB を起動します。
   サーバ A の MRB は、REMOTE_APPLICATION_LIST のエントリを読み込み、RemoteApp1 で指定されているエントリを2回実行することを認識します。
3. サーバ A の MRB が、サーバ B の MRB に、MyApp1 を 2 回実行するよう要求を出します。
4. サーバ B の MRB は、この要求に対応して、uniPaaS サーバエンジンを二つ起動します。

6.7 サーバを停止させるにはどうしますか？

サーバを停止させるには、以下の方法があります。

- プログラムから RqRtTerm/RqRtTermEx 関数を実行する。
- MRB のタスクバーアイコンから停止させる。
- MRB を停止させる。
- コマンドラインリクエスタ Mgrqcmdl.exe を使う。
- MRB モニタから停止させる。
- タスクマネージャなどから強制終了させる（最後の手段）

uniPaaS サーバを停止させる方法はいろいろあります。

プログラムから RqRtTerm/RqRtTermEx 関数を実行する

uniPaaS のアプリケーション内で、RqRtTerm あるいは RqRtTermEx 関数を実行します。

この関数を実行すると、サーバテーブルあるいはサービステーブルに定義されている MRB に対して、インスタンス終了のリクエストが送信されます。

この関数の利用方法については、以下のような使い方をします。

1. RqRts 関数を使って、インスタンス数を取得する。
2. RqRtInf 関数を使って、各インスタンスについての情報を取得する。
3. 取得した情報をもとにして、停止すべきインスタンスの番号を決定する。
4. その番号を指定して、RqRtTerm あるいは RqRtTermEx 関数を実行する。

ステップ1において、サーバテーブルに定義されているサーバ名を指定すると、対応する MRB に登録されているサーバインスタンスの数が返されます。サービステーブルに定義されているサービス名を指定すると、指定したサーバ（MRB）において、指定されたアプリケーション名が実行されている uniPaaS サーバのインスタンス数が返されます。

各関数の使い方について、詳しくはリファレンスマニュアルを参照してください。

MRB のタスクバーアイコンから停止させる

MRB が実行されているサーバでログインすると、タスクバーに MRB のアイコンが表示されます。

ここで右クリックでメニューを表示させ、「Shutdown Enterprise Servers」を選ぶと、この MRB に登録されている uniPaaS サーバインスタンスがすべて終了させられます。

- メニューの名前は「Shutdown Enterprise Servers」ですが、エンタープライズサーバだけでなく、リッチクライアントサーバも停止します。
- この方法では、MRB に登録されているサーバインスタンスがすべて停止させられます。特定のサーバインスタンスだけを停止し、他はそのまま実行を継続させる、という指定はできません。

MRB を停止させる

MRB を停止させると、その MRB に登録されていた uniPaaS サーバインスタンスもすべて停止させられます。

コマンドラインリクエスタ Mgrqcmdl.exe を使う

コマンドラインリクエスタ mgrqcmdl.exe を使って、停止させることもできます。これには、次の手順で停止します。

1. すべてのインスタンスを停止させる場合:
   以下のコマンドを実行します。(1行で書きます)

```
mgrqcmdl.exe
   -terminate=RTS
   -host=(MRB のあるサーバ名)
   -port=(ブローカポート)
   -password=(ブローカ照会パスワード)
```

2. 特定のインスタンスを指定して停止させる場合:
   ① 以下のコマンド を実行して、MRB に登録されているサーバインスタンスの一覧を表示させます。

```
mgrqcmdl.exe
   -query=RTS
   -host=(MRB のあるサーバ名)
   -port=(ブローカポート)
   -password=(ブローカ照会パスワード)
```

   ② 表示された一覧の中から、停止させるべきインスタンスを決めます。
   ③ 以下のコマンドを実行して、ホスト名とポート番号によりインスタンスを指定して、停止させます。

```
mgrqcmdl.exe
   -TERM=(ホスト名)/(ポート番号)
   -host=(MRB のあるサーバ名)
   -port=(ブローカポート)
   -password=(ブローカ照会パスワード)
```

例:

サーバインスタンスの一覧表示

```
C:¥Program Files¥uniPaaS¥RichClient Server V1Plus>MGrqcmdl.exe -query=RTS -password=password
                Enterprise Servers of (MyRCServer/5215)
  ------------------------------------------------------------------------------
#          EnterpriseServer              Pid        Status        License              Application
================================================================================================
1|       MyRCServer/1501   192.168.75.15 2932   Avail Idle    : 0|0  0|0  0|5 , 0 , 0   | MyApp1
2|       MyRCServer/1594   192.168.75.15 2208   Avail Idle    : 0|0  0|0  0|5 , 0 , 0   | MyApp1
3|       MyRCServer/1672   192.168.75.15 2648   Avail Idle    : 0|0  0|0  0|5 , 0 , 0   | MyApp1
4|       MyRCServer/1735   192.168.75.15 2164   Avail Idle    : 0|0  0|0  0|5 , 0 , 0   | MyApp1
5|       MyRCServer/1908   192.168.75.15 2120   Avail Idle    : 0|0  0|0  0|5 , 0 , 0   | MyApp1
```

サーバを指定して、停止させる

```
C:¥Program Files¥uniPaaS¥RichClient Server V1Plus>MGrqcmdl.exe -term=MyRCServer/1501 -password=password
```

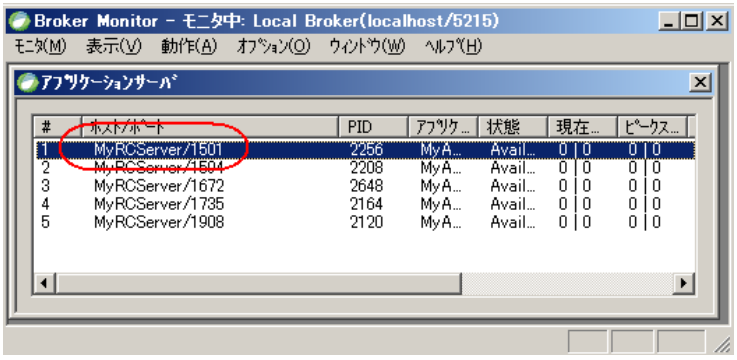
サーバー覧を再度表示させると、停止されたエンジンは表示されない。

```
C:¥Program Files¥uniPaaS¥RichClient Server V1Plus>MGrqcmdl.exe -query=RTS -password=password
                Enterprise Servers of (MyRCServer/5215)
  ------------------------------------------------------------------------------
#          EnterpriseServer              Pid        Status        License              Application
================================================================================================
1|       MyRCServer/1594   192.168.75.15 2208    Avail Idle     : 0|0  0|0  0|5 , 0 , 0   | MyApp1
2|       MyRCServer/1672   192.168.75.15 2648    Avail Idle     : 0|0  0|0  0|5 , 0 , 0   | MyApp1
3|       MyRCServer/1735   192.168.75.15 2164    Avail Idle     : 0|0  0|0  0|5 , 0 , 0   | MyApp1
4|       MyRCServer/1908   192.168.75.15 2120    Avail Idle     : 0|0  0|0  0|5 , 0 , 0   | MyApp1
```

ブローカモニタから停止させる

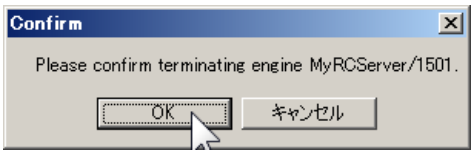
ブローカモニタからサーバインスタンスを停止させることもできます。

1. ブローカモニタを起動し、「アプリケーションサーバ」ウィンドウで、停止するサーバを選択します。



2. メニュー「動作 → アプリケーションサーバ停止」を選びます。

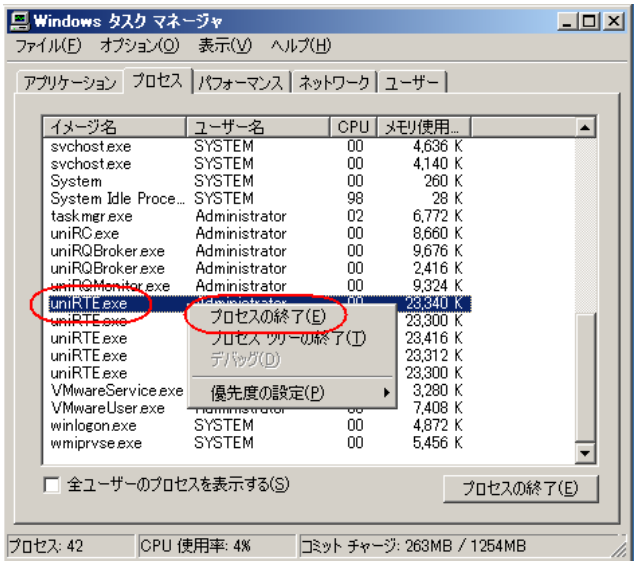
3. 確認ダイアログが出るので、「OK」を押します。



タスクマネージャなどから強制終了させる（最後の手段）

正常に動作しているアプリケーションサーバならば、上記のいずれかの方法により停止させることができますが、異常な状態になっている場合（サーバのハングアップや、無限ループなど）には、停止できないこともあります。

このような場合に強制的に停止させるには、最後の手



段として、タスクマネージャなどから、サーバのプロセスを強制的に終了させることがあります。

1. タスクマネージャを開きます。(タスクバーから右クリックでメニューを表示させて、「タスクマネージャ」を選択します)
2. 「プロセス」タブを開き、停止したいプロセスを選択します。uniPaaS サーバのインスタンスは、「イメージ名」が uniRTE.exe です。
3. 「プロセスを終了」ボタンを押します。
4. 確認ダイアログが出たら、「はい(Y)」を押します。

この方法は、プロセスを強制的に終了させるため、サーバの終了化処理などが一切行われません。通常、未コミットのデータは DBMS がロールバックしますから、トランザクション設定が正しくされていれば DBMS のデータの不整合などは発生しないはずですが、その他のファイル出力などは中途半端に終了していることもありえますので、この方法で強制的に停止させるのはできるだけ避け、あくまで最後の手段として利用してください。

マルチインスタンスの場合には、サーバプロセス uniRTE.exe が複数タスクマネージャに表示されます。その中で、どのプロセスを終了させるかは、プロセスの挙動を見て推測・判断する必要があります。
例えば、ハングアップしているプロセスであれば、CPU 利用率がいつまでも 0 % であるし、逆に、無限ループに陥っている場合には、100% 近い状態がいつまでも続きます。ただし、マルチ CPU の場合には、CPU 2個の場合 50%、CPU 4個の場合 25%などになります。また、メモリ使用量を見てみて、異常に大きな値になっている場合には、何か問題が起きている可能性が高いと推測できます。

6.8 スケーラビリティをどう実現しますか？

エンタープライズサーバの複数起動や、HW の追加により実現できます。

uniPaaS サーバは、スケーラブルな設計となっていますので、システムの負荷が高くなってきた場合にも対応することが可能です。

パフォーマンス向上を図るためには、まず最初にボトルネックがどこにあるかを見極めることが重要です。

ボトルネックを見つける第一歩は、各サーバ HW での CPU 利用率、メモリ使用量、IO(ディスク)回数およびバイト数、ネットワーク使用率などを、リソースモニタなどで確認します。

全般的にサーバ側のリソースに余裕があるのに、クライアント側でのレスポンスが良くない場合には、uniPaaS サーバのインスタンスを増やすことで改善されることがありますので、マルチインスタンス構成とすることを検討してください（3.16「マルチインスタンス構成の設定は？」参照）。これにより、サーバ HW 資源を有効に活用することができるようになります。

ただし、同一サーバ HW 上でのマルチインスタンス化による性能向上には限界があります。例えば、CPU を非常に多く利用するようなアプリケーションを実行している場合には、CPU が 100%になってしまったら、それ以上いくらインスタンスを増やしても性能向上は見込めません。逆に、プロセスが多くなることによるオーバーヘッドが増大して性能が低下してしまう可能性もあります。

同一サーバ HW 上でのマルチインスタンス化は、あくまで、サーバの HW 資源を有効に活用することに目的があります。

uniPaaS サーバを実行している HW で CPU やメモリ使用量が高い場合には、サーバの HW 性能に限界が来ていることを意味しますから、次に検討することは、サーバ HW を増設して、マルチサーバ構成とすることです（3.17「マルチサーバ構成の設定は？」参照）。サーバ HW が N 台になれば、単純計算で、N 倍の性能向上が見込めます。

システムの使い方によっては、Web サーバがネックになってしまう可能性もあります。この場合には、IIS のプロセスの CPU 利用率が高くなっているはずです。このような状態になったら、Web サーバを別 HW に分けることを検討すべきです。また、セキュリティの面からも、Web サーバは別 HW に分けることは望ましいことです（3.15「Web サーバのみ別 HW にしたい場合には？」参照）。

それでも性能が不足するようになったら、ロードバランサを導入することを検討してください（3.20「ロードバランサによる多重化」参照）。ロードバランサをクライアントに対しての前面に立てて、その背後に Web サーバ→MRB→アプリケーションサーバ→DBMS という一連の系統を複数立ち上げます。これにより、単純計算では、N 系統で N 倍の性能向上を見込めます。

ロードバランサを使って、複数系統を用意しておくことは、可用性向上の点からも好ましい構成であると言えます。

これはあくまでもサーバの構成という観点から見た、一般的な指針として理解してください。パフォーマンスの問題は多くの要因がからむ複雑な問題であり、実際のアプリケーション・環境・利用方法などを勘案して総合的に検討する必要があります。

サーバの構成変更の他には、プログラムの改善、データベースのスキーマの変更、DBMS サーバのチューニングなども、パフォーマンスに大きな影響を与えます。

6.9 セキュリティをどう高めますか？

インターネットからの攻撃に対しては、Web サーバを DMZ に配置することによりセキュリティを高めることができます。

また、通信データの傍受を防ぐには、次のような方法を利用します。

- SSL を利用する。
- （Web アプリケーションの場合）なりすましや乗っ取りを防ぐため、セッションの情報を工夫する。
- （リッチクライアントの場合）データの暗号化機能を用いる

インターネット環境では悪意ある攻撃、通信データの傍受やなりすまし、乗っ取りなどの危険を十分に考慮してシステムを設計する必要があります。

悪意ある攻撃からシステムを守る第一歩は、Web サーバを DMZ に置いて、アプリケーションサーバや DBMS を直接インターネットにさらさないようにすることが基本です。

このときには、Web サーバとインターネットの間は、HTTP ポート(80) および、必要ならば HTTPS ポート(443) のみを開きます。Web サーバと MRB およびアプリケーションサーバの間は、必要最小限のポートだけを開くようにします。具体的な設定については、3.15「Web サーバのみ別 HW にしたい場合には？」を参照してください。

次には、通信データを傍受されたり、なりすまし、乗っ取りを防ぐ必要があります。

一番簡単で確実なのは、SSL(HTTPS)を利用することです。これにより、データが暗号化されるので、第 3 者はデータを見ても内容を知ることができなくなります。

SSL を利用しない場合には、基本的にデータはすべて見えてしまいます。乗っ取りなどを防ぐためには、セッション情報を工夫して、なりすましの第 3 者からのリクエストでないかをよく確認するようにアプリケーションを設計する必要があります。

このへんの話題は、非常に多岐・詳細に渡る話になってしまい、また uniPaaS 固有の問題ではなく、Web アプリケーション一般の問題であるために、関連図書も多く出版されていますので、そちらの専門書籍を参照してください。

リッチクライアントシステムの場合には、uniPaaS サーバに通信データの暗号化機能があるので、セキュリティが必要な環境ではこれを利用することができます。暗号化機能はデフォルトでオフになっています。暗号化機能は、MAGIC.INI の[MAGIC_SPECIALS] セクションの SecureBrowserClient パラメータにより有効・無効が決まります。このパラメータが Y ならば暗号化され、N ならば暗号化されません。

```
[MAGIC_SPECIALS]
SpecialClientSecureMessages=Y
```

6.10 可用性をどう高めますか？

多重化(マルチインスタンス構成、代替ブローカ、ロードバランサの導入)により可用性を高めることができます。

可用性を高める基本は、SPOF (Single Point of Failure) を極力なくすことにあります。そのために、ハードウェア・ソフトウェア共に多重化を行います。uniPaaS サーバでは、次のような多重化の機能を備えています。

マルチインスタンス構成: uniPaaS サーバ (エンタープライズサーバ、あるいはリッチクライアントサーバ)を複数立ち上げて置くことにより、一つのサーバインスタンスが動作停止しても、他のサーバインスタンスが稼働を続けることにより、システム全体として動作を続けられます。

この方法だけでは、Web サーバ、MRB、ライセンスサーバが SPOF となっています。これらのコンポーネントは比較的安定しているソフトウェアコンポーネントですので、最低限の可用性が必要な場合には、マルチインスタンス構成で行うのが良いでしょう。

マルチサーバ構成: マルチインスタンス構成の延長として、サーバインスタンスを複数のサーバ HW 上で実行させることができます。これにより、サーバ HW(あるいはオペレーティングシステム)が動作停止しても、動いているサーバにより動作を継続できます。

ただし、Web サーバ、MRB、ライセンスサーバが SPOF であることには変りなく、これらのソフトウェアモジュールが動作しているサーバ HW が動作停止すると、システムとして停止してしまいます。

このことから、マルチサーバ構成は、可用性の向上というよりは、負荷分散の目的のために用いられることが多いです。

代替ブローカ構成: この構成では、MRB → uniPaaS サーバ (マルチインスタンス) → ライセンスサーバ というひとまとまりのセットを二つ用意しておいて、ひとつは通常系、他方は待機系とします。正常に稼働しているときには通常系を使いますが、MRB が稼動停止してしまった場合には、待機系に処理を移して、実行を継続します。

この構成では、MRB 以下が二重化されるので、Web サーバのみが SPOF となります。

ロードバランサを使う構成: この構成はもっと多重化を進めたもので、Web サーバ以下、MRB、uniPaaS サーバ、ライセンスサーバすべてを多重化し、Web サーバの前にロードバランサを置くことにより、ソフトウェア的には SPOF を究極的に低減させることができます。

この構成での SPOF はロードバランサのみとなりますが、ロードバランサは通常専用ハードウェアを使った信頼性の高いものですので、可用性は極めて高くなります。

ソフトウェアの SPOF としては、この他に DBMS がありますが、DBMS はそれ自体にクラスタリングなどの高度な可用性向上の仕組みが備わっていますので、必要な可用性のレベルによって、適当なものを選んでください。

6.11 uniPaaS サーバの異常終了時、自動的に再起動させるようにできますか？

MRB の自動起動機能を使って、異常終了時に再起動させることができます。

uniPaaS サーバが何らかの理由により異常終了してしまった場合、自動的に再起動されるように設定できれば便利です。これは、MRB の自動起動機能を使って実現することができます。

MRB の自動起動機能では、MGRB.INI ファイルの [APPLICATIONS_LIST] に登録されたプロセスを、MRB 起動時に同時に起動させることができますが、MGRB.INI に ReLoad=Y という記述があった場合には、MRB は自分が起動したプロセスを監視し、もし終了した場合にはそれを検出し、自動的に再起動させます。

MGRB.INI

```
ReLoad = Y
[APPLICATIONS_LIST]
MyApp1 = uniRTE.exe /DeploymentMode=B /StartApplication=[ecf file name and path],D:¥Magic¥150SP1,,,1,0
```

この自動再起動機能の対象となるのは、次の条件をすべて満たすプロセスです。

- [APPLICATIONS_LIST] に登録されている。
- MRB の自動起動機能により、起動される。すなわち、[APPLICATIONS_LIST] に定義されたエントリの、最後から二番目のパラメータが正数であるもの。

この自動再起動は便利ですが、起動直後に異常終了するような状況では、起動→異常終了→起動→異常終了→・・・ というサイクルがいつまでも続くことになってしまいます。これを防ぐために、自動再起動の回数には上限を設定することができます。最初の自動起動も含めた自動(再)起動回数の上限は、[APPLICATIONS_LIST] のエントリの最後のパラメータとして指定します。これが 0 の場合には、無制限となります。

6.12 スタンバイ構成の場合のライセンスはどうなりますか？

ホットスタンバイとコールドスタンバイとがあり、ホットスタンバイはスタンバイライセンスが必要となります。
コールドスタンバイでも、フェイルオーバー構成で使用する場合は、スタンバイライセンスが必要です。

可用性を向上させるための基本は冗長構成であり、この具体的な方式として「スタンバイ構成」という方法が採用されることがあります。

スタンバイ構成は、MRB→uniPaaS サーバ という系統を二つ用意しておき、一方は通常系として、通常動作させておく系統であり、他方は待機系として、通常は使われないが、通常系に異常があって動作停止してしまった場合に、通常系に交代して処理を継続するものです。

構成によっては、Web サーバや DBMS も含めて2系統準備する構成も考えられます。これには大きく分けて、次の二つの方式があります。

ホットスタンバイ: 待機系と標準系が両方共起動していて、常に動作可能になっているもの。通常はリクエストが標準系にだけ行くので、待機系は処理を行わないが、通常系が動作を停止してしまったと判明した場合に、瞬時にリクエストを待機系に送る方法。ユーザから見ると、システムの動作停止時間が非常に短いのが特徴。

コールドスタンバイ: 待機系は普通は停止しています。これはハードウェア的に電源がオフになっているか、あるいは OS は起動しているが uniPaaS サーバが起動していない状態になっています。標準系に異常があったことが検出されたら、待機系の系統を起動します。ホットスタンバイ方式に比べ、起動のための時間がかかり（OS から起動する場合には数分かかる）、その間システムが停止してしまうのが欠点です。

uniPaaS サーバでは、このいずれの形式も対応できるようになっています。構成の方法は、マルチインスタンス・マルチサーバ、あるいは代替ブローカ構成の場合と同じですが、スタンバイ構成の場合に異なるのは、次の2点です。

- 運用方法: 運用時に、通常は待機系は休止しており、通常系に異常があったときに待機系を利用するように、運用管理を適切に行う必要があること。
- ライセンス: ライセンスについては、2系統あることから2倍のライセンス数が必要になるのですが、スタンバイの場合には価格が単純に2倍ではなく、特別の価格が用意されています。具体的な価格については、ケースバイケースの契約により決まります。

スタンバイ・ライセンスについて: サーバクラスタ、仮想化技術等を使用して高可用性のシステムを構成し、フェイルオーバー（システム障害が発生したとき、自動的に冗長な待機系システムに切り換える機能）構成で使用する場合は、別途スタンバイライセンスが必要です。

昨今の仮想化技術の進歩によりホットスタンバイとコールドスタンバイの区別が曖昧になってきています。利用する技術や運用方式によりいろいろなバリエーションが考えられますので、ライセンスについて詳しくはその都度、ご相談ください。

6.13 ファイアーウォールの設定方法は？

以下のポートを開ける必要があります。

- クライアント → Web サーバ間は、80 (HTTP)/443 (HTTPS)
- インターネットリクエスタ → MRB 間はブローカポート
- インターネットリクエスタ → エンタープライズサーバ間は サーバポート
- エンタープライズサーバ → MRB 間はブローカポートおよびサーバポート

インターネットの各種脅威からシステムを守るために、Web サーバを DMZ に置いて、インターネットとアプリケーションサーバ・DBMS を隔離する構成が一般にとられています。この場合、システムが正常に動作するためには、ファイアーウォールを適切に設定する必要があります。

uniPaaS サーバシステムでは、各モジュール間の通信のために、以下のような TCP/IP のポートを利用します。

クライアント → Web サーバ間: クライアント → Web サーバ間は、HTTP あるいは HTTPS プロトコルを用いて通信します。従って、外側の（外界と Web サーバの間の）ファイアーウォールでは、ポート 80 (HTTP)およびポート 443 (HTTPS)を開ける必要があります。

Web サーバの構成によってはこのような標準的な設定以外の設定も可能ですが、その場合には、Web サーバの設定に合わせたポート番号を開く必要があります。

インターネットリクエスタ → MRB 間: インターネットリクエスタ → MRB 間は、ブローカポートを通じて通信します。従って、内側の（Web サーバと、MRB および uniPaaS サーバの間）ファイアーウォールでは、ブローカ ポート（V1Plus のデフォルト設定では 5215）を開く必要があります。

MRB→サーバ間: インターネットリクエスタは、MRB だけでなく、uniPaaS サーバとも通信を行います。この通信は、サーバポートを使って行われます。サーバポートは、uniPaaS サーバの「通信ゲートウェイ」テーブルの「TCP/IP」エントリに設定されているパラメータで、ポート番号の範囲が指定されます。

この設定は、MAGIC.INI ファイルの [MAGIC_COMMS] セクションの「TCP/IP」エントリの３番目のパラメータに相当します。V1Plus のデフォルトでは、1500-2000 です。

```
[MAGIC_COMMS]
NONE = 1, 0, NO Parameters needed
TCP/IP = 2, 30, 1500-2000
```

サーバポートは、uniPaaS サーバが MRB およびインターネットリクエスタと通信を行う際に利用するポートで、インスタンスごとに異なるポートを使います。uniPaaS サーバは起動時に、指定された範囲の中から自動的に、使われていないポート番号を探し出して、利用します。

サーバポート番号はインスタンスごとに異なり、各インスタンスが自動的に決定するものですので、ファイアーウォールでは MAGIC.INI で設定したポート番号の範囲のポートを開けておく必要があります。ここでのファイアーウォールは、インターネットリクエスタのある Web サーバと、uniPaaS サーバの間の、内側のファイアーウォールです。

エンタープライズサーバと MRB は、通常は DMZ より内側の同一 LAN セグメント内に配置されるものなので、

ファイアーウォールはないと思いますが、使われるポートを挙げておきますと、ブローカポートおよびサーバポートとなります。

ライセンスサーバ: ライセンスサーバについては、ポート 744 を使いますが、uniPaaS サーバとライセンスサーバの間にファイアーウォールが入るような構成は通常ないので、これについては説明を省略します。

注意: バージョンによる違い

Magic V10 および uniPaaS V1 (Ver1.5) でリッチクライアントシステムを構成する場合には、リッチクライアントキャッシュディレクトリも外部に公開しておく必要がありました。このディレクトリは、uniPaaS サーバと Web サーバの両方から共有できるファイルシステムでなければならないので、どうしても、内側のファイルシステムで、ネットワークファイル共有のためのポートを開いておく必要がありました。しかし、ファイル共有のネットワークプロトコルは攻撃されやすいものであるので、セキュリティの厳しい組織ではこれを開けることが許可されないこともありました。

uniPaaS V1Plus (Ver1.8)では、リッチクライアントキャッシュディレクトリは uniPaaS サーバからだけアクセスできれば良いものとなっており、Web サーバと共有する必要がなくなったので、内側のファイアーウォールで、ネットワークファイル共有のためのポートを開く必要がなくなり、安全性が高まりました。

6.14 MRB やライセンスサーバはどこに置くのがよいですか？

理想は安定した独立のサーバ。

MRB やライセンスサーバの性質として、次のようなことが挙げられます。

- 多重化した構成でない限り、MRB やライセンスサーバは単一障害点（SPOF）になる。
- 機能が特化されているので、安定性は高い。
- HW にかける負荷（CPU、メモリ、ディスク IO など）は通常かなり低い。
- 個々の処理は小さいが、迅速に応答する必要がある。
- MRB の場合、他のモジュールとのやりとり回数が多い。

MRB が他のモジュールとやり取りする回数については、クライアントがリクエストを Web サーバに出して、インターネットリクエスタ経由でアプリケーションサーバが処理をし、レスポンスを返すまでに、MRB はインターネットリクエスタやアプリケーションサーバと 10 回以上の通信を行います。このため、例えば 1 秒に 10 リクエストが来るようなシステムでは、MRB は 1 秒に 100 回以上各モジュールと通信を行うことになります。

このような性質を持つモジュールであるので、MRB やライセンスサーバは、できるだけ他の処理の影響を受けない安定した HW 上に置いておきたいものです。

例えば、大量のデータの集計や印刷のバッチタスクを行うエンタープライズサーバでは、バッチタスクの実行中 CPU、メモリ、ディスク IO、ネットワークなどに高い負荷がかかります。このような HW と同一 HW 上に MRB やライセンスサーバを置くと、その動作に悪い影響を与える可能性があります。

Web サーバも、リクエスト回数や、データ量が多い場合には、かなりの負荷になります。また、セキュリティの観点からは、Web サーバが DMZ にあることが多く、この場合には MRB やライセンスサーバが同じ HW 上にあるのは好ましくありません。

以上のようなことから、MRB やライセンスサーバは、理想的には独立した安定なサーバに置くことが望ましいです。HW にかける負荷は低いので、大きな HW スペックは必要ありません。実用的な点からこれが難しい場合には、なるべく負荷の小さなサーバに置くようにしてください。

仮想環境の場合 HOST-ID も仮想化されるため、特別なライセンスを発行する必要がある可能性もあります。ライセンスサーバは仮想環境ではなく、実サーバにインストールすることをお勧めします。 仮想環境での実行については、6.21「仮想環境で uniPaaS を実行できますか？」も参照してください。

6.15 エンタープライズサーバには、「コンテキスト」はないのですか？

ありますが、通常はリクエスト処理終了とともに消去します。タスク特性でコンテキストを残しておくように指定することも可能です。

リッチクライアントサーバでは「コンテキスト」を使い（5.2 「リッチクライアントサーバでの「コンテキスト」とは何ですか？」参照）、設計段階ではコンテキストに対する考慮（インスタンスの分割方法、タイムアウトの考慮）が必須です。

一方、エンタープライズサーバでは、通常、1リクエストが独立した処理単位であり、コンテキストがリクエストにまたがって使われるということがないので、コンテキストに対する考慮はあまり必要ありません。

この場合でも、コンテキストが全くないわけではなく、内部的には、エンタープライズサーバがリクエストを受け付けた時にコンテキストが作成され、処理が終了するとコンテキストが破棄されます。

ただし、タスクの設定によって、処理終了時にコンテキストを破棄せずに、そのまま保持させておくことが可能です。コンテキストを保持させておくには、「タスク特性 → 拡張→作成コンテキストの保持」を Yes にします。

エンタープライズサーバでのコンテキストの利用例としては、Web アプリケーションシステムにおいて、メモリテーブルの 一時テーブルをリクエストにまたがって利用したいとか、グローバル変数にログイン名などの認証情報を残しておくなどが考えられます。

具体的な使い方については、本書の範囲外となるので省略します。

6.16 タイムアウトにはどのような種類があり、どこで指定しますか？

各モジュール間の通信にそれぞれタイムアウトがあり、各種 INI ファイルで設定します。

タイムアウトの種類

uniPaaS サーバでは、次のようなタイムアウトの種類があります。

タイムアウトの種類	パラメータ名	意味
通信タイムアウト	CommTimeout	TCP/IP レベルでの、通信障害等に起因する遅延に対するタイムアウト。
ブローカタイムアウト	BrokerTimeout	リクエストが MRB に送られた後、MRB から使用可能なエンジンを割当てることができたことが報告されるまで待機する時のタイムアウト。
リクエスタタイムアウト	Requeste r Timeout	アプリケーションサーバがリクエスタ処理を終わらるまでリクエスタが待つ最大時間を指定します。
サーバタイムアウト	ServerTimeout	アプリケーションサーバから MRB への「I-AM-ALIVE」メッセージの送信間隔。

タイムアウトの設定は、MGREQ.INI、MGRB.INI、MAGIC.INI などに指定しますが、モジュールによって指定する場所が決まっていますので、正しい INI ファイルに指定する必要があります。
以下に、タイムアウトの詳細と、有効なモジュール、指定する INI ファイルなどについて説明します。

上記のタイムアウトについては、リファレンスマニュアル で「検索」を行うと出てきますので、そちらを参照してください。また、上記の他に、シャットダウンタイムアウト (ShutdownTimeout)、終了待ちタイムアウト (CloseWaitTimeout) がありますが、本書では説明を省略します

6.16.1 通信タイムアウト

通信タイムアウトは、TCP/IP（ソケット）レベルでのタイムアウトです。uniPaaS サーバは、通信に TCP/IP を使っていますので、このタイムアウトはすべてのモジュールで有効となります。

このタイムアウトが発生する理由は、主に指定の誤り、モジュールの起動・終了タイミングの誤りや、ネットワーク障害に起因するものです。

例えば、次のようなときに、TCP/IP のエラーが起こり、通信タイムアウトとなります。

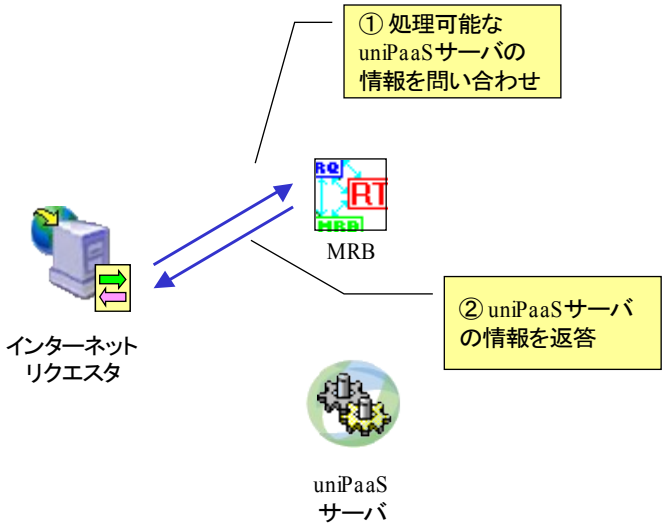
- uniPaaS サーバエンジンを起動しようとしたときに、MRB,が起動していなかった。
- MRB として指定したサーバのホスト名やポート番号が違っていた。
- ネットワーク障害が起きたため、モジュール間の通信ができなくなった。

このタイムアウトに対するパラメータは、CommTimeout で、次の INI ファイルに指定します。指定されていなかった場合には、デフォルトで 10 秒となります。

モジュール	INI ファイル	INI ファイルの場所	備考
リクエスタ	MGREQ.INI	それぞれのモジュールがインストールされているディレクトリ	
uniPaaS サーバ	MGREQ.INI	uniPaaS ディレクトリ	MAGIC.INI に指定しても無視されます
MRB	MGRB.INI	MRB がインストールされているディレクトリ	MGREQ.INI ファイルでの CommTimeout 指定は無視されます。

6.16.2 ブローカタイムアウト

このタイムアウトは、リクエスタ、及び、uniPaaS サーバでのリモートコールコマンド（同期）実行時に有効です。MRB にはこの設定は関係ありません。



リクエスタ（uniPaaS サーバでのリモートコマンド実行時もを含む）は、最初に MRB にリクエストを発行すると、MRB はそのリクエストに応じて、処理可能なアプリケーションサーバを選んで、その情報（IP アドレス、サーバポート番号等）をリクエスタに返します。このときの、リクエスタが MRB にリクエストを発行してから、アプリケーションサーバの情報を受け取るまでのタイムアウトが、ブローカタイムアウトです。

通常は、アプリケーションサーバの情報はすぐにリクエスタに返されますが、次のような場合にブローカタイムアウトが発生します。

- すべてのアプリケーションサーバのすべてのスレッドが処理中（ビジー）であり、処理可能なアプリケーションサーバが見つからなかったとき、MRB はアプリケーションサーバが空きになるまで待機しますが、ブローカタイムアウトの時間が経過しても空きのアプリケーションサーバが見つからなかった場合。
  この場合には、APP-IN-USE のステータスが MRB からリクエスタに返されます。
- リクエストに指定したアプリケーションを実行可能なアプリケーションサーバが MRB に登録されていなかった場合、同様に、MRB は指定されたアプリケーションサーバが登録されるのを待機しますが、ブローカタイムアウトの時間が経過してもそのようなアプリケーションサーバが登録されなかった場合。
  この場合には、APP-NOT-FOUND のステータスが MRB からリクエスタに返されます。

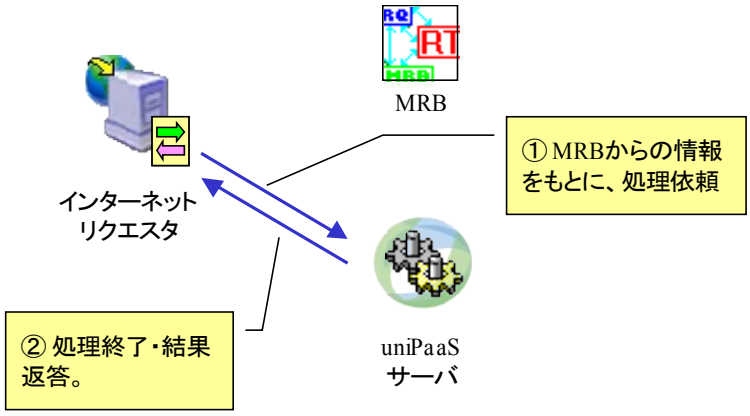
ブローカタイムアウトのデフォルト値は10秒です。

このタイムアウトのパラメータ名は BrokerTimeout で、各リクエスタと同じディレクトリにある MGREQ.INI ファイルに指定したものが有効となります。

6.16.3 リクエスタタイムアウト

このタイムアウトも、リクエスタ（uniPaaS サーバが同期リモートコールリクエストを実行する場合を含む）に関してだけ有効で、MRB には関係ありません。

リクエスタは、MRB から、リクエスト処理可能なアプリケーションサーバの情報を受け取ると、その情報に基づいて、直接アプリケーションサーバに接続し、やりとりを開始します。アプリケーションサーバはリクエスタから与えられた公開プログラム名およびパラメータ名に基づいて処理を行います。この処理は、プログラムの作り方によって、すぐ終わるものもあれば長時間かかるものもあります。処理が終了したら、結果のデータと共に、リクエスタに終了を通知します。



リクエスタタイムアウトは、リクエスタがアプリケーションサーバにリクエストを発行してから、結果を受け取るまでのタイムアウトです。アプリケーションサーバで処理に長時間がかかり、リクエスタタイムアウト時間になっても終了が通知されなかった場合には、タイムアウトが発生し、リクエスタは処理の終了を待たずに、「110 ERR REQUEST TIMEOUT」ステータスとなります。

なお、このタイムアウトが発生した場合でも、アプリケーションサーバ内部での処理はそのまま継続され、中断されることはありません。

リクエスタタイムアウトのデフォルト値は 0 であり、これは無制限（タイムアウトは発生しない）を意味します。

リクエスタタイムアウトのパラメータのキーワードは RequesterTimeout であり、次の INI ファイルで指定します。

- リクエスタを実行する場合には、リクエスタと同じディレクトリにある MGREQ.INI ファイル。
- uniPaaS サーバが同期リモートコマンドを実行する場合には、MAGIC.INI ファイルで指定します。同じディレクトリに MGREQ.INI があって、RequesterTimeout が指定されていても、uniPaaS サーバからは無視されます。
- MRB には、このパラメータは関係ありません。

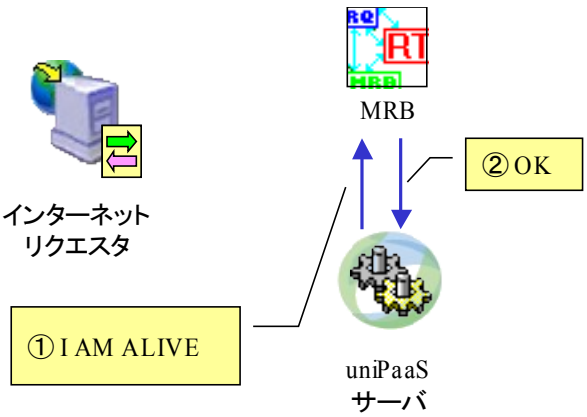
6.16.4 サーバタイムアウト

サーバタイムアウトは、MRB でだけ有効です。リクエスタおよび uniPaaS サーバには関係ありません。

MRB と uniPaaS サーバの間では、一定時間ごとに I-AM-ALIVE メッセージを交換して、お互いのプロセスが正常に動作していることを確認します。この I-AM-ALIVE メッセージを出す間隔がサーバタイムアウトです。

I-AM-ALIVE メッセージは、uniPaaS サーバが出して、MRB がそれに対して返事を返します。

もし、MRB が正常動作していない場合、uniPaaS サーバが I-AM-ALIVE を出しても、MRB が返事をしないようになります。この時には、uniPaaS サーバは「MRB は異常状態にある」と判断し、動作を停止します。



反対に、MRB は正常動作しているけれども、登録されている uniPaaS サーバが正常動作していない場合には、タイムアウト時間が過ぎても MRB は I-AM-ALIVE メッセージを受けない事になります。このような事態を MRB が検出したら、MRB は「uniPaaS サーバが異常状態にある」と判断し、このサーバの登録を解除し、以後のリクエストに対してこのサーバを割り当てないようにします。

サーバタイムアウトのパラメータ名は ServerTimeout であり、秒単位で指定します。サーバタイムアウトのデフォルト値は、60 秒です。このパラメータは、MRB と同じディレクトリにある MGRB.INI ファイルに指定します。

6.17 MRB が異常終了した場合、何が起こりますか？

MRB に登録されている uniPaaS サーバがすべて終了します。MRB は自動的に再起動されます。

MRB は uniPaaS のサーバシステム全体が円滑に実行するための要になるモジュールなので、非常に安定して動作することが必要であり、実際にも安定して動作しますが、それでも異常終了するケースが皆無ではありません。

MRB が異常終了すると、その MRB に登録されている uniPaaS サーバ（エンタープライズサーバ、リッチクライアントサーバ）はすべて、リクエストを送ってくれる元がなくなってしまいます。

uniPaaS サーバと MRB とは TCP/IP のコネクションを通して通信していますが、MRB が異常終了するとコネクションが切断されるので、uniPaaS サーバインスタンスはこれを検出すると自動的に動作停止（強制終了）します。これに伴ない、このインスタンスがライセンスサーバから割り当てられていたライセンスが解放されます。

サーバインスタンスが強制終了することによる影響は、下記の項目を参照してください。

モジュール	参照箇所
エンタープライズサーバ	3.21「エンタープライズサーバが異常終了した場合、何が起こりますか？」
リッチクライアントサーバ	5.17「クライアントが動作停止・切断した場合、コンテキストはどうなりますか？」

基本的に、すべてのサーバが終了するので、実行中の処理やコンテキストはすべて破棄されます。

その後、MRB は自動的に再起動します。これはどういう仕組になっているかというと、MRB は実は二つのプロセスからなっていて、一つは監視用のプロセスであり、もう一つは実際にリクエストの交通整理を行うプロセスです。タスクマネージャの「プロセス」タブを開いてみると、二つ MRB のモジュール名 uniRQBroker.exe があるのはこのためです。

監視用のプロセスは、通常は何もしませんが、もう一方のプロセス（実際にリクエストを捌いているプロセス）が異常終了すると、これを検知して、適当な時間の後に再起動します。

再起動された uniRQBroker.exe のプロセスは、自分自身の初期化を行ってリクエストを受け付けられる状態になると共に、MGRB.INI の [APPLICATIONS_LIST] や [REMOTE_APPLICATIONS] にエントリが定義されていれば、それに従って、uniPaaS サーバのインスタンスを起動させます。

このようにして、万一 MRB が異常終了してしまった場合には、その時点でのリクエストやコンテキストはなくなってしまいますが、システムとしては自動再起動して、運用を継続できるようになっています。

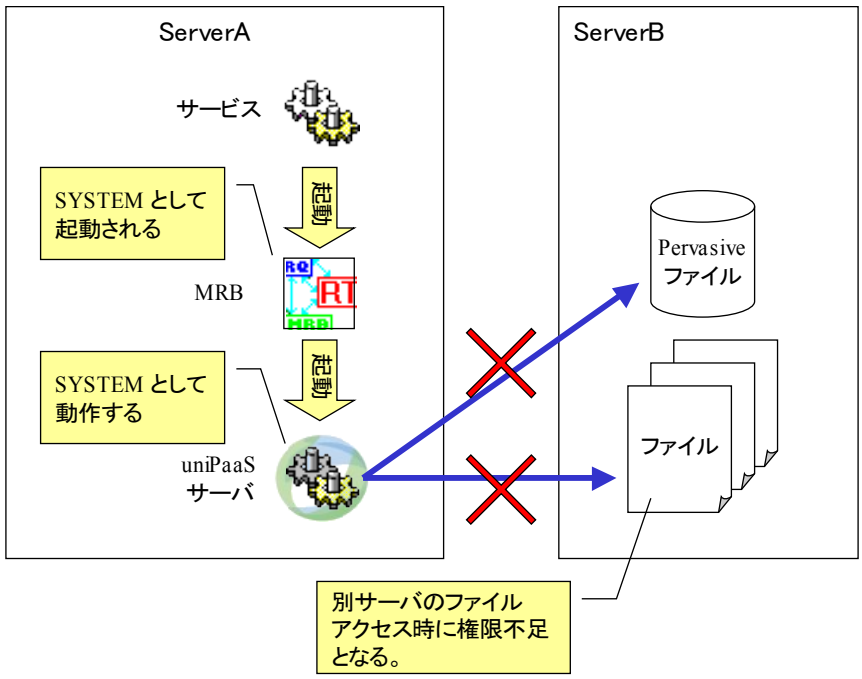
6.18 uniPaaS サーバから、ネットワークファイルをアクセスできますか？

サービスのアカウントの変更が必要です。

uniPaaS サーバからファイル（Pervasive ファイル、あるいは、タスクの IO テーブルに定義されている入出力ファイルとか、IO 関数により直接操作するファイル）をアクセスする場合には、アクセス権限についての注意が必要で、別サーバ HW 上にあるネットワークファイルは特に気を付ける必要があります。

uniPaaS のサーバ製品をインストールすると、デフォルトの設定では、MRB はサービスとしてインストールされます。これは、Windows 起動時に自動的に MRB も起動され、MRB の自動起動機能を用いて uniPaaS サーバも自動起動される、という運用が普通だからです。

この際、MRB のサービスの実行アカウントは「SYSTEM」となります。このアカウントは、ローカルの資源（ファイルなど）に対しては大きな権限を持っているのですが、別 HW 上のリモートの資源に対してはほとんど権限を持っていません。



MRB の自動起動機能により起動される uniPaaS サーバのインスタンスは、MRB の権限を継承して、「SYSTEM」として実行されます。すなわち、uniPaaS サーバのインスタンスは、リモートの資源に対して、ほとんど権限がないアカウントで実行されていることになります。このため、uniPaaS サーバのインスタンスが、Windows のファイル共有機能を使って、別サーバ HW 上のファイルを参照しようとすると、権限がないので、エラーになってしまいます。

通常は、uniPaaS サーバがリモートのネットワークファイルを参照するようなことはあまりないと思いますが、アプリケーションの仕様によりどうしてもネットワークファイルを参照しなければならない場合には、次のいずれかの方法によります。

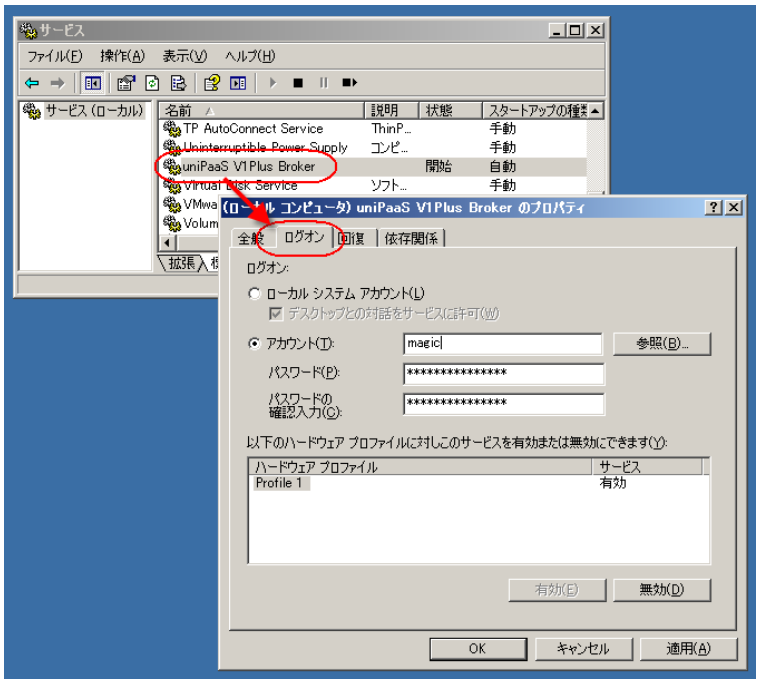
MRB サービスの実行アカウントを変える

この方法は、MRB が実行する資格情報を変更するもので、デフォルトの SYSTEM から、別のアカウント（リモートの資源に対して、必要なアクセス権限があるアカウント）を指定して実行させるようにします。

MRB の資格情報が変われば、それから起動される uniPaaS サーバの資格情報も継承されますので、指定されたアカウントで動作するようになります。

次のように設定します。

1. コントロールパネルから「管理ツール → サービス」を開きます。
2. MRB のサービス「uniPaaS V1Plus Broker」を選択します。
3. プロパティを開きます（右クリックでコンテキストメニューから選ぶ、あるいはダブルクリック）
4. 「ログオン」タブを開き、アカウントとそのパスワードを指定します。



自動起動のエントリでユーザ ID/パスワードを指定する

MRB の自動起動機能を使うとき、MGRB.INI の[APPLICATIONS_LIST] のエントリで、ユーザ名とパスワードとを指定すると、その資格情報で uniPaaS サーバが実行します。これを利用して、ネットワークファイルに対してアクセス権限のあるアカウントを指定してやれば、uniPaaS サーバからネットワークファイルをアクセスできるようになります。

下記は、ユーザ ID (magic) とパスワード (magicpass) を指定した例です。(MyApp1 の行は、実際には 1 行で書きます)

```
[APPLICATIONS_LIST]
MyApp1 = uniRTE.exe /DeploymentMode=B /StartApplication=*Projects¥MyApp1¥MyApp1.ecf,
   C:¥Program Files¥uniPaaS¥RichClient Server V1Plus,magic,magicpass,0,0
```

この場合、MRB のログイン情報は変更せず、ローカルシステムアカウント(SYSTEM)のままにしておきます。MRB のログイン情報をローカルシステムアカウント以外に設定すると、エントリに指定されたユーザ名で起動しようとした時に OS から権限違反のエラーが返ってきて、起動することができません。

[APPLICATIONS_LIST] では、エントリごとにユーザ ID とパスワードを設定できますので、uniPaaS サーバの用途に応じて、異なった資格情報で実行させることができます。セキュリティのためには、アプリケーションを動作させる上で必要最小限の権限を持ったアカウントで動作させるべきです。

6.19 １台のサーバあたりのユーザ数の目安はどれくらいですか？

アプリケーションやサーバ HW 等に完全に依存します。

システム構成を設計する際には、どのようなハードウェアを何台くらい用意する必要があるのか、ということが非常に重要です。処理にハードウェアが追いつかなければレスポンスが非常に悪いシステムになってしまいますし、逆に必要以上のハードウェアを用意するとシステムのコストが上がってしまいます。

このため、「uniPaaS でシステムを構築する際、１台のサーバで何ユーザくらいをサポートできますか？」という質問が良く寄せられます。

これに対する答えは、公式的には、アプリケーションやハードウェアに完全に依存するものなので、ケースバイケースで評価する必要がある、ということになります。uniPaaS はあくまでアプリケーションを作成するためのツールであるので、開発者はいかようにでもプログラムを作成することができます。極端な話、1ユーザだけでサーバの資源(CPU やメモリ)を食いつぶしてしまうようなアプリケーションも作成することも可能です。

ただ、これではあまりにも漠然としすぎているので、昨今のハードウェア事情と、一般的なアプリケーションの作りを前提として、次のようなところを目安として考えることができましょう。

まず、uniPaaS サーバ製品の基本性能として、データ入力や少数のレコードの参照のような軽い処理ばかりであれば、１サーバハードウェアあたり、

- エンタープライズサーバ: 30 リクエスト/秒
- リッチクライアントサーバ: 100 ユーザ/サーバ

くらいは十分に処理が可能です。

エンタープライズサーバについては、リクエスト/秒が単位となっていますが、ユーザ数を割り出すには簡単な計算を行えば算出できます。Web アプリケーションであれば、ごく大ざっぱに言って、「１画面表示 ＝ 1リクエスト」と考えられます。もちろん、フレームなどを用いて、１回の画面表示更新に複数のリクエストが出る場合には、それで割る必要があります。そして、１ユーザあたりの平均的な画面表示更新頻度で割れば、利用ユーザ数が計算できます。

実際の運用ではピーク時の性能が確保される必要がありますので、アプリケーションで様々な処理が入ってオーバーヘッドが加わることを考慮すれば、上記の数値の半分くらいが余裕のあるところではないかと思われます。すなわち、エンタープライズサーバは 15 リクエスト/秒、リッチクライアントサーバは 50 ユーザ程度です。

それを超えるユーザ数をサポートする場合には、それに比例してサーバハードウェアを追加し、マルチサーバ構成とします。

各サーバでは、マルチインスタンス構成とします。各サーバあたり何インスタンスを立ち上げればよいかについては、あまり極端なことはせず、3 ～ 10 インスタンス程度が普通でしょう。可用性向上のためにはインスタンスが多い方が良いのですが、各インスタンスは、何もしていなくとも数 10～100MB 程度のメモリを消費しますから、何十もインスタンスを立ち上げるのは推奨できません。

以上はあくまで大雑把で一般的な概算ですので、個々のアプリケーションにおいては、タスクの複雑さ、対象レコード数の大小(特に、検索などで対象レコード数が多い場合には時間がかかる)、処理内容、サーバハードウェアの性能、他のソフトウェアとの干渉などにより、大きく性能は変わりますので、実機について十分なストレステストを行って確認することが重要です。

6.20 サービスの依存関係

TCP/IP → ライセンスサーバ → MRB → エンタープライズサーバ の順になります。

uniPaaS サーバ製品は、デフォルトのインストールで、サービスとしてセットアップされます。これにより、サーバ OS が起動したときに、同時に、uniPaaS サーバも自動的に起動されるようになります。

具体的には、次のような二つのサービスがインストールされます。(カスタムインストールにより、実際にインストールされるサービスを変更することもできます。)

- FlexLM License Server (ライセンスサーバ)
- uniPaaS V1Plus Broker (MRB)

このほかに、次のようなサービスも、インストールされている必要があります。

- DBMS のサービス (DBMS をローカルで利用する場合)
- Web サーバ (World Wide Web Publishing Service、パーティショニングサーバの場合には不要)

また、uniPaaS サーバ製品の各モジュールは、TCP/IP を利用して相互に接続されているので、Windows の TCP/IP が利用可能になっている必要があります。

それぞれのサービスはそれぞれ独立して設定されており、通常はそれで問題は起こらないのですが、まれに、タイミングの問題で、依存するサービスが利用可能な状態になっていないときに、それを利用する別のサービスが起動してしまい、エラーを起こすことがあります。

このような場合には、サービスの依存関係を明示的にサービスの定義に設定してやる必要があります。

各サービスの依存関係 (開始すべき順序)は、以下の通りです。

1. TCP/IP (すべてのモジュール間の通信プロトコルなので、最初に利用可能になっていることが必須)。
2. ライセンスサーバ (FlexLM)
   Web サーバ (World Wide Web Publishing Service)
   DBMS
3. MRB
4. uniPaaS サーバエンジン (これは、MRB から自動起動されるのが普通なので、サービスに依存関係を定義する必要はない)

(2 番目の 3 つのサービスについては、お互いに独立であり、依存関係はないので、同一の番号内に入れてあります。)

サービスの依存関係を定義するには、レジストリを設定する必要があります。具体的な操作方法については、マイクロソフト社のナレッジベース
http://support.microsoft.com/kb/193888/ja や、あるいは、インターネットで「Windows サービス依存関係」などをキーワードとして検索すると出てきますので、そちらを参照してください。

6.21 仮想環境で uniPaaS を実行できますか？

動作保証外ですが実績はあります。仮想環境に固有な注意事項もあります。

近年のハードウェア性能の向上を背景とし、サーバ運用管理コストの低減と、リソースの有効利用を目的として、サーバの仮想化が一般的になってきました。Hyper-V、VMWare、XenServer などが代表的な商用仮想化ソフトウェアです。

このような仮想環境上での uniPaaS 製品の実行については、公式的にはサポートしておらず、動作保証はなしとなります。正しく動作するかどうかは、仮想化ソフトウェアが実ハードウェアをどれだけ忠実にエミュレートしているかという、仮想化ソフトウェアの方の問題となります。実際には、通常の物理サーバと互換性のある範囲内において、利用可能であり、動作実績もあります。

仮想化環境に固有な問題になる可能性があるものとしては、以下のようなものが考えられます：

- サーバ製品でライセンスサーバも動作させる場合、HOST-ID も仮想化されるため、特別なライセンスを発行する必要がある可能性もあります。ライセンスサーバは仮想環境ではなく、実サーバにインストールすることを強くお勧めします。
- 仮想化ソフトウェアの高可用性・負荷分散機能等を利用する場合、実ハードウェアでは想定されないような環境的変化がある場合があり、uniPaaS の動作に影響を与える可能性があります。
- 仮想化ソフトウェアでフェイルオーバーを使う場合、特別なスタンバイライセンスが必要になります。

仮想化技術は急速に進歩しており、新しい技術の台頭に伴って新たな問題が出てくることもあるかもしれませんので、実際の利用に先立って、十分にテストを行うようにしてください。

また uniPaaS 製品は、他社の DBMS 製品を通してデータにアクセスしているので、ご利用になる DBMS 製品が仮想環境で動作保証されていることが前提となります。仮想環境における動作保証については、各 DBMS ベンダー様にご確認ください。

6.22 uniPaaS サーバをフォアグランドで実行できますか？

環境設定やデバッグ等でどうしても必要な場合を除いては、バックグラウンドで実行してください。

バックグラウンドモード

uniPaaS サーバは、通常、バックグラウンドモードで実行します。このモードでは、次のような特性があります:

- Web ブラウザ（Internet Explorer）
- クライアント実行製品の場合と異なり、uniPaaS サーバプロセスが、直接ユーザと対話を行うことはありません。
- uniPaaS サーバプロセスは、画面（ウィンドウ）を表示しません。
- マルチスレッド、マルチユーザ（リッチクライアントサーバの場合）で動作します。
- MRB がサービスとして設定されていて、uniPaaS サーバ Web ブラウザ（Internet Explorer)が MRB から自動起動されるようになっていたら、Windows がブートした時に、誰がログインせずとも人手の介入なしで、自動的に起動されます。

バックグラウンドモードで実行させるには、MAGIC.INI に以下の設定をします。

```
[MAGIC_ENV]
DeploymentMode = B
```

フォアグラウンドモード

uniPaaS サーバは、特殊な用途においては、フォアグラウンドモードで実行させることも可能です。この場合には、次のような特性があります。

- uniPaaS サーバプロセスは、クライアント実行製品と同じように、デスクトップに画面（ウィンドウ）を表示します。
- エンジンは、ライセンスや MaxConcurrentRequests/MaxConcurrentUsers の設定に関わらず、1スレッドのみで動作し、リッチクライアントサーバでは1ユーザのみをサポートします。
- ウィンドウがデスクトップに表示されるので、ログインして手作業で uniPaaS サーバプロセスを起動する必要があります。

フォアグラウンドモードで実行させるには、MAGIC.INI に以下の設定をします。

```
[MAGIC_ENV]
DeploymentMode = R
```

いつフォアグラウンドモードにするか？

フォアグラウンドモードでは、1インスタンスあたり1スレッド、1ユーザだけしかサポートされないので、実際の運用には実用的でありません。従って、uniPaaS サーバをデフォルトでインストールすると、バックグラウンドモードで動作するようにセットアップされます。

uniPaaS サーバをあえてフォアグラウンドで動作させるのは、主に、環境設定、およびテスト・デバッグの局面に限られるでしょう。

リッチクライアントサーバの場合 (リッチクライアントの MGRIA1P1 ライセンスで動作している場合)には、バックグラウンドモードでのみ動作します。DeploymentMode=R としてフォアグランドモードで起動しようとすると、エラーになります。

環境設定のためにフォアグラウンドモードにする

uniPaaS サーバの主要な環境設定は、MAGIC.INI ファイルに定義されます。これには、「動作環境」のさまざまなパラメータ、DBMS やデータベース設定、サービスやサーバの設定、プリンタの設定、ユーザやグループの設定などがあります。

環境設定を変更するには、直接 MAGIC.INI ファイルをテキストエディッタで修正することも可能ですが、パラメータ値が F、R、B 等といった、一見してどういう意味かわかりにくい記号であったり、データベース定義のように、多くのパラメータがカンマで区切られて定義されている場合には、カンマの位置を間違えてしまい、意図したような設定として解釈されない場合もあります。

このようなことから、環境設定は、MAGIC.INI ファイルを直接テキストエディッタで編集するのではなく、uniPaaS の設定画面から行う方がはるかに簡単です。このために、uniPaaS サーバエンジンをあえてフォアグラウンドで起動し、「設定」メニューから、設定したい画面(「動作環境」「データベース」など)を開き、編集します。

デバッグ・動作確認のためにフォアグラウンドモードにする

動作確認やデバッグの際には、特定のタスクやハンドラが実行されているかを確認するために、エラーコマンドを入れてみたり、アクティビティモニタを表示させてみたりすることがよくありますが、これはバックグラウンドモードの場合には画面に表示されないので、フォアグラウンドモードで起動し、画面を表示させてやる必要があります。

アクティビティモニタの場合には、出力する内容やゲートウェイのログレベルを必要に応じて「ロギング」メニューから変更することもできます。

バックグラウンドモードで実行している場合でも、MAGIC.INI の[MAGIC_LOGGING] の指定により、ログを記録させることができます。また、リモートデバッガの機能を用いて、アクティビティモニタをリアルタイムで表示させることが可能です。リモートデバッガについては、リファレンスヘルプで、「リモートデバッガ」をキーワードにして検索してください。

サービスの「ユーザとの対話を許可する」の設定

uniPaaS サーバ製品のインストーラは、デフォルトで、MRB をサービスとしてインストールします。

Windows のサービスには、ユーザとの対話を許可するかしないかを、サービスのプロパティ画面で設定することができますが、uniPaaS サーバのデフォルトのインストールでは、ユーザとの対話は許可されないようになっています。

MRB がユーザとの対話を許可しないサービスとして起動されると、そこから起動される uniPaaS サーバプロセスもまた、ユーザとの対話が許可されません。

ユーザとの対話が許可されない場合には、ウィンドウを表示させることができません。従って、このように構成されている場合には、必ずバックグラウンドモードでなければなりません。

万一、フォアグラウンドモードの設定 (MAGIC.INI で DeploymentMode=R) で、ユーザとの対話が許可されないプロセスとして起動してしまうと、uniPaaS サーバプロセスが起動時にハングアップしてしまうこともありますので、十分に注意してください。ハングアップすると、プロセスを強制終了させるしか回復方法はありません。

6.23 Studio の実行エンジンをサーバエンジンとして使うことはできますか？

技術的制限と、ライセンス上の制限から、運用環境でのサーバエンジンとしては利用できません。

Studio 製品に付属の実行エンジンは、uniPaaS サーバエンジンと同じモジュールであり、同じ機能を有しています。また、Studio ライセンス（MGCSTK1P)を使えば、エンタープライズサーバ（5 スレッド）としてもリッチクライアントサーバ（5 ユーザ）としても利用可能です。また、実行エンジンは、フォアグラウンド・バックグラウンドいずれでも実行可能です。

しかし、Studio ライセンスは、開発時のテスト実行を目的としているものですので、以下の制限があります。

- ライセンス上、テスト目的以外の、実運用のためには利用できない。
- 連続リクエストの上限値が 2000 までとなっている。

このような制限があるため、Studio に付属の実行エンジンを実運環境での実行エンジンとして利用することはできません。

なお、「連続リクエストの上限が 2000」というのは、1 回のデバッグセッションでの上限であり、デバッグセッションが改まれば、リクエスト数はリセットされます。

一回のデバッグセッションというのは、次のような一連の流れです。

- Studio から F7 キーでリッチクライアントプログラムを起動したときに開始され、そのプログラムが終了して Studio に戻ってきたときに終了します。
- Studio で「プロジェクトの実行」によって実行モードになったときに開始され、実行モードから Studio に戻ってきたときに終了します。

このようなセッションが終了した場合には、リクエスト数はリセットされ、次のデバッグセッションでは、また 0 からリクエストがカウントされます。

6.24 サーバ側で Office 製品を使うことができますか？

Microsofot 社によりサポートされていません。

アプリケーションサーバに Microsofot Office 製品（Excel、Word など）をインストールし、uniPaaS の COM 機能を使って、サーバ側で Excel シートを作成してクライアントにダウンロードさせたり、Word 文書を印刷したりしたい、という要望を時々聞きます。

しかし、このような使い方は Microsoft 社によってサポートされておらず、推奨もされていません。特に、Vista、Windows Server 2008、Windows 7 など、最近の Windows OS では、サービスは「セッション 0 分離」というセキュリティ機能の強化が施されており、Office のようにユーザインターフェースを持つアプリケーションの実行には大きな制限がかかっています。

以上のことから、アプリケーションサーバから Office 製品を直接利用することはできないと考えた方が良いと思われます。

Excel などの利用がどうしても必要になるのであれば、以下のような方法で行うことになります。

- サーバ側では CSV（カンマ区切りファイル）を作成し、クライアントにダウンロードして、クライアントにインストールされている Excel を起動する。
- サーバ側で、Excel プログラムを起動する代わりに、XML ファイル（DOCX ファイル）を作成・操作する。

「セッション 0 分離」に関する情報は、マイクロソフト社の HP で、「セッション 0 分離」をキーワードにして検索してください。

6.25 同一サーバに独立した複数のアプリケーション環境を実装することは可能ですか？

推奨はしないが、系統ごとに設定を分離して、混在させることは可能です。

同一サーバに独立した複数のアプリケーション環境を配置したくなる場合が時々あります。例えば、開発環境、テスト環境、教育用環境、実運用環境を実行することが必要だが、サーバハードウェアがひとつしかない、というような場合です。

すべて同一バージョンであることを前提として、このような運用は可能ですが、しっかりと管理しないと混乱しがちなので、管理上の問題として、推奨できません。このような用途には、仮想環境を作成して、uniPaaS の環境を分離するのがベストです。

どうしても混在させなければならない場合には、次の設定をそれぞれの系統ごとに別々に分けて設定するようにしてください。

- 製品種別が異なる場合 には、uniPaaS 製品のディレクトリ。
- ブローカ ポート番号。uniPaaS 製品のインストーラでは、最大4つまで、MRB をサービスとして登録することができます。それぞれの MRB は、異なるブローカーポートで動作します。
- IIS のエイリアス（Scripts ディレクトリ、RIAModules ディレクトリ、PublishtdApplications ディレクトリ）。
- MAGIC.INI などの設定ファイル。

異なるバージョンの uniPaaS 製品を、同一のサーバにインストールして、混在して利用することはサポートされていません。同一ファイル名だがバージョンが異なるファイルが干渉する可能性があり、トラブルの元になりやすいので行わないでください。

6.26 サーバの処理されるリクエストを監視する方法は？

コマンドラインリクエスタを使って監視します。

運用時には uniPaaS サーバの状態や、リクエストの処理状況を監視したくなることがよくあります。このような場合には、コマンドラインリクエスタ（Mgrqcmdl.exe）を利用するのが一番簡単です。

コマンドラインリクエスタを使った監視方法で、よく使うものは以下のようなものがあります。

- 現在起動中のエンジンの一覧

```
MGrqcmdl.exe -query=rt
```

- 現在実行中（Executing）あるいは待機中（Pending）のリクエストの一覧

```
MGrqcmdl.exe -query=log
```

- MRB の統計情報

```
MGrqcmdl.exe -query=load
```

他にも Mgrqcmdl.exe には多くの機能・オプションパラメータがありますので、リファレンスマニュアルで利用法を見て、活用してください。

コマンドラインリクエスタの詳しい利用方法は、リファレンスヘルプの「分散アプリケーション > アプリケーションパーティショニング > コマンドラインリクエスタ」を参照してください。

コマンドラインリクエスタの他に、uniPaaS には Broker モニタ があります。これは GUI インターフェースを持った、MRB の監視ツールで、一定間隔で表示を更新します。

便利なツールではありますが、定期的に多くの情報を取得するため、MRB にかかる負担は大きく、運用に支障を来たすことが間々あります。基本的に開発環境やテスト環境で利用し、運用環境では常時使わないようにしてください。

Broker モニタを常時利用している際に、サーバで不定期に問題が起こるようなことがあったら、まずは Broker モニタの利用を停止して様子を見てください。

Broker モニタをどうしても使いたい場合には、次の設定により MRB にかかる負担を軽減することができます。

- メニュー「オプション → 再表示設定」から、「再表示間隔」を長くとる（例：300 秒）。また、「表示リクエスト数」を少なめにする（例：30）
- 「コンテキスト」画面を閉じる。

7 Web サーバの設定の詳細

以下の設定が必要になります。IIS のバージョンにより具体的な設定方法が異なります。

- 仮想ディレクトリ
- ISAPI/CGI の設定
- MIME 設定

Web サーバの設定は、uniPaaS 製品のインストーラが行います。オールインワン構成で行う場合にはこれでよいのですが、本書で解説したようなさまざまなシステム構成を利用する場合には、手作業で Web サーバの設定を行う場合もあります。

例えば、Web サーバだけを DMZ 上の別サーバに載せて運用する場合には、Web サーバ HW には uniPaaS のインターネットリクエスタだけを置いて、uniPaaS サーバ本体は DMZ より内側の別のサーバ HW 上にインストールすることになります。このような場合には、Web サーバを手作業で設定する必要があります。

また、オールインワンで構成した場合でも、万一、正しく動作しなくなってしまった場合に、原因の追求のために、Web サーバがどのように設定されていなければならないのかを知っておくことは重要です。

7.1 サーバの形態による違い

サーバの形態により、最低限必要な Web サーバの設定が異なります。

- エンタープライズサーバでは、インターネットリクエスタのある Scripts ディレクトリの設定が必要です。
- パーティショニングサーバでは、Web サーバを使わないので、Web サーバの設定もありません。
- リッチクライアントサーバでは、Scripts 仮想ディレクトリの他に、RIAModules、PublishedApplications ディレクトリの設定が必要です。また、MIME の設定が必要な場合もあります。

以下に、それぞれについて詳しく説明します。

7.2 Scripts ディレクトリ

7.2.1 物理ディレクトリ名

uniPaaS サーバのインストールディレクトリの下の Scripts という名前のサブディレクトリです。

7.2.2 内容

ここには、インターネットリクエスタや、それの必要モジュール・ファイルなどが格納されます。

7.2.3 仮想ディレクトリ名

このディレクトリには、uni18Scripts という仮想ディレクトリ名が割り当てられます。

ここで、「18」というのは、uniPaaS サーバのバージョン番号で、製品のバージョンにより異なります。

この名前はデフォルトのもので、インストーラで「カスタム」インストールを選ぶことにより、異なる名前を指定することもできます。

7.2.4 仮想ディレクトリのプロパティ

この仮想ディレクトリでは、インターネットリクエスタ MGrqispi018.dll (ISAPI API を利用)、あるいは Mgrqcgi018.exe (CGI API を利用) を実行しなければならないので、実行権限がなければなりません。またセキュリティのためには、読み込み、ディレクトリの参照、書き込みなどの権限をオフにしておいた方がよいでしょう。

また、Windows 2003 以降では、セキュリティの強化のため、ISAPI、CGI などはデフォルトで無効となっています。有効にするには、ISAPI/CGI 拡張を設定する必要があります。

7.3 RIAModules ディレクトリ

RIAModules ディレクトリは uniPaaS リッチクライアント製品をインストールしたときに作成されます。エンタープライズサーバ製品にはありません。

7.3.1 物理ディレクトリ

uniPaaS サーバのインストールディレクトリの下の RIAModules という名前のサブディレクトリです。

7.3.2 内容

ここには、リッチクライアントシステムでのクライアントモジュールが格納されています。

詳しくは、次のようなディレクトリ構造になっています。

- Mobile サブディレクトリには、Windows Mobile 用のクライアントモジュールが格納されています。
- uniRC_1_8_1_410 サブディレクトリには、PC 用のクライアントモジュールが格納されています。ここで「1_8_1_410」というのは、uniPaaS 製品のバージョン（1.8SP1x を表す）と、クライアントモジュールのビルドバージョン（410）とからなります。これらの数値は、uniPaaS のバージョンやクライアントモジュールにより異なります。

uniPaaS 製品のバージョンが同じでも、クライアントモジュールには不具合修正などのために、異なるビルドバージョンのものが提供されることがあります。この場合には、そのビルドバージョンに合ったサブディレクトリ名のところに格納しておく必要があります。また、RIAModules ディレクトリの直下にある uniRC.application ファイルもそのビルドバージョンに合ったものにしておく必要があります。

通常、パッチなどとして提供されるモジュールは、必要なディレクトリ構造を保って提供されます。

7.3.3 仮想ディレクトリ名

このディレクトリには、uni18RIAModules という仮想ディレクトリ名が割り当てられます。ここで、「18」というのは、uniPaaS サーバのバージョン番号で、製品のバージョンにより異なります。

この名前はデフォルトのもので、インストーラで「カスタム」インストールを選ぶことにより、異なる名前を指定することもできます。

7.3.4 仮想ディレクトリのプロパティ

このディレクトリには、exe や dll モジュールもありますが、これらはクライアント PC にダウンロードして、クライアント PC 上で実行されるべきもので、Web サーバ上で実行されるものはありません。従って、この仮想ディレクトリには、「読み取り」権限のみを付与します。実行権限を付与すると、exe や dll のモジュールが Web サーバ上で実行され、エラーとなってしまうので、実行権限を設定してはいけません。

また、セキュリティのために、ディレクトリー覧、書き込みなどの権限は与えないことをお奨めします。

7.3.5 MIME 設定

このディレクトリからは、リッチクライアントのクライアント側モジュール uniRC.exe がダウンロードされますが、ClickOnce のセキュリティの要求を満たすため、uniRC.exe のマニフェストファイル uniRC.exe.manifest もダウンロードされます。このファイルが正しくダウンロードされるためには、.manifest という拡張子に対する MIME 設定が application/x-ms-manifest に設定されていなければなりません。

Windows XP、Windows Server 2003 には設定されていませんので、手作業で追加します。Vista 以降にははじめから設定されているので、追加設定する必要はありません。

7.4 PublishedApplications ディレクトリ

このディレクトリも、uniPaaS リッチクライアントサーバにだけあります。エンタープライズサーバにはありません。

7.4.1 物理ディレクトリ

uniPaaS サーバのインストールディレクトリの下の PublishedApplications という名前のサブディレクトリです。

7.4.2 内容

ここには、リッチクライアントアプリケーションで使うマニフェストファイルの他、アプリケーションで参照する HTML ファイル、画像ファイル、その他のファイルなどを格納します。

開発環境では、Studio のリッチクライアント インターフェス ビルダが自動的にここにサブディレクトリを作成します。サブディレクトリの名前はデフォルトでプロジェクト名で（異なる名前を設定することも可能です）、そこにマニフェストファイルと .publish.html ファイルを作成・格納します。（5.5「マニフェストファイルとは何ですか？」参照）

実行環境では、リッチクライアントサーバ製品をインストールすると、初期状態ではこのディレクトリは空なので、開発環境と同じディレクトリ構造をそのままコピーします。

7.4.3 仮想ディレクトリ名

このディレクトリには uni18RIAApplications という仮想ディレクトリ名を設定します。

ここで「18」というのは uniPaaS 製品のバージョン番号で、バージョンにより異なります。

また、この仮想ディレクトリ名はデフォルトのもので、インストーラで「カスタムインストール」を選択することにより、異なる名前を指定することもできます。

7.4.4 仮想ディレクトリのプロパティ

ここにあるファイルは、すべて PC にダウンロードするものばかりなので、「読み取り」権限のみを付与します。

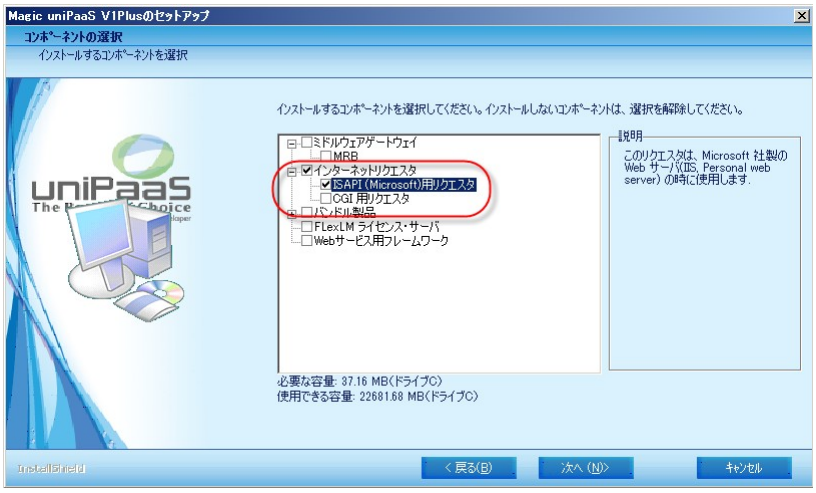
セキュリティのために、ディレクトリ参照、実行、書き込み権限などは与えないことをお奨めします。

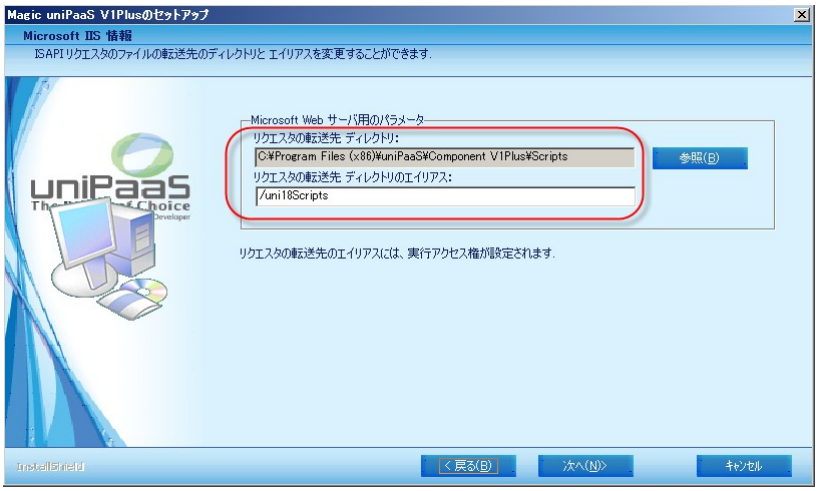
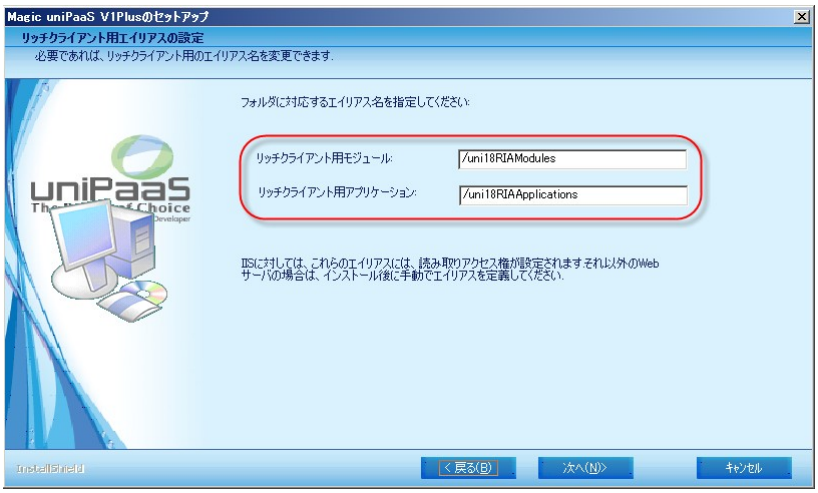
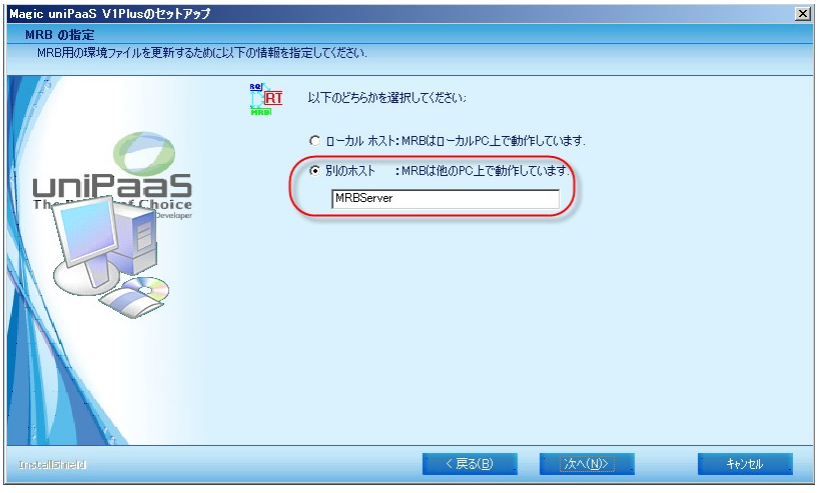
また、このディレクトリにはマニフェストファイルがあるので、拡張子 .application に対する MIME 設定が必要です。

7.5 uniPaaS インストーラでのインストール

uniPaaS サーバそのものはインストールせず、Web サーバの設定だけをおこないたい場合、uniPaaS の「Component」のインストーラで行うことができます。

7.5.1 インストール手順

インストーラのトップ画面で、「Component」を選択します。	
ウィザードに従ってインストールを進めます。 「コンポーネントの選択」画面で、「インターネットリクエスタ」を選択します。	

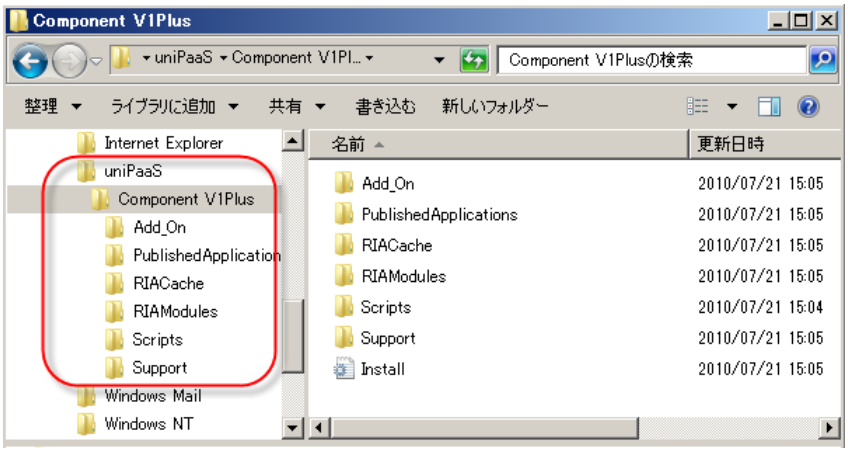
リクエスタの物理ディレクトリとエイリアスとを確認します。通常はデフォルトのままにしてください。	
リッチクライアント用モジュールとリッチクライアント用アプリケーションのエイリアスも確認してください。 これも通常デフォルトのままにします。	
MRB のあるサーバを指定します。	
続けてウィザードを進めると、インストールが完了します。	

x64 サーバの場合には、アプリケーションプールの 32 ビットアプリケーションの実行許可の設定が必要ですが、上記の手順によるインストールでは行われません。7.11「X64 サーバでの設定」に従って、手作業で設定を行って下さい。

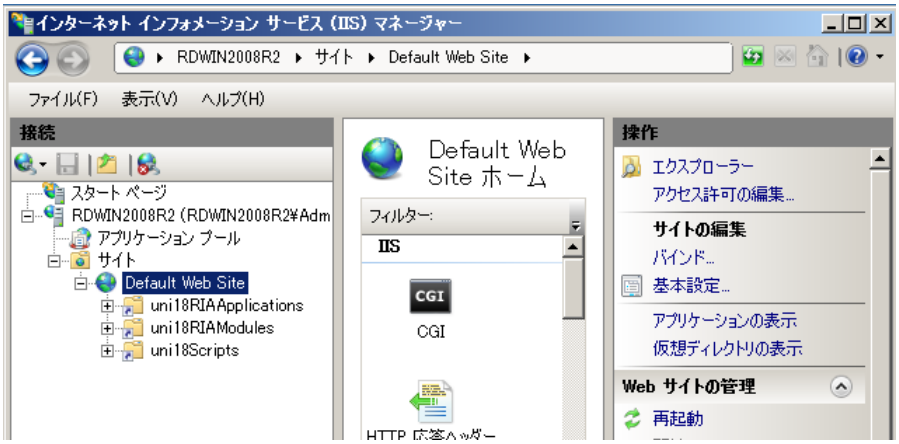
7.5.2 インストールの結果

上記のインストールにより、次のように設定されます。

- uniPaaS ディレクトリ C:¥Program Files¥uniPaaS¥Component V1Plus が作成され、Scripts ディレクトリ、RIAModules、PublishedApplication その他のディレクトリが作成され、必要なファイルがコピーされます。



- IIS のエイリアスが設定されます。



RIAModules (エイリアスは uni18RIAModules) および PublishedApplications (エイリアスは uni18RIAApplications) は、uniPaaS リッチクライアントサーバでは必要ですが、uniPaaS エンタープライズサーバでは必要ありません。Component でインストールした場合には、これらのディレクトリは無条件に作成されますので、不要な場合には手作業で削除してください。

7.6 手作業での IIS 設定

通常、Web サーバの設定は uniPaaS のインストーラを使って行うのが便利ですが、インストーラを使わずに、すべて手作業で行うこともできます。本節以下では、手作業により IIS の設定を行う方法を説明します。

実際の環境設定において手作業で行うことはあまりないと思いますが、IIS の環境周りに何か問題があるように思われる場合には、トラブルシューティングの手段として知っておくと役に立つこともあると思います。

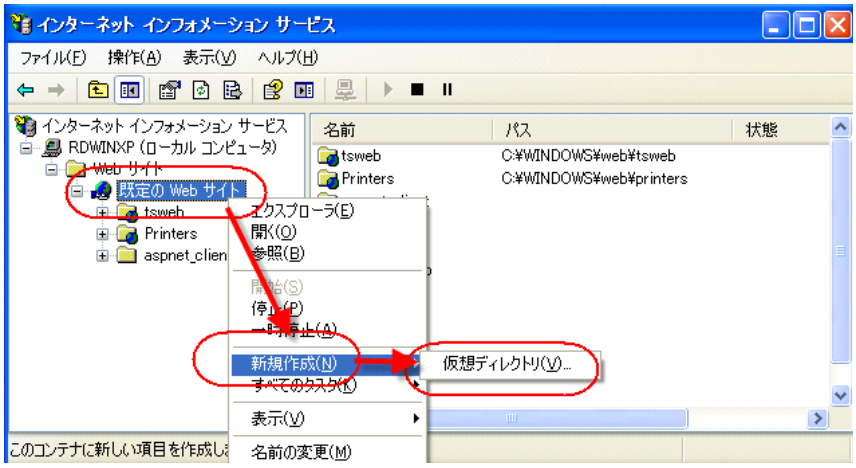
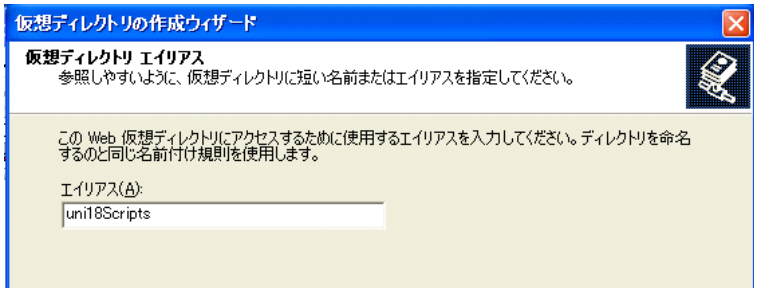
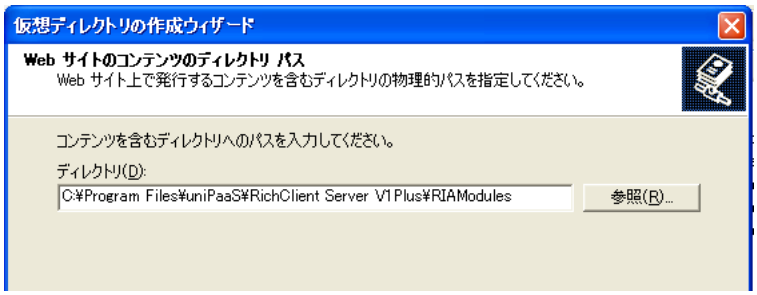
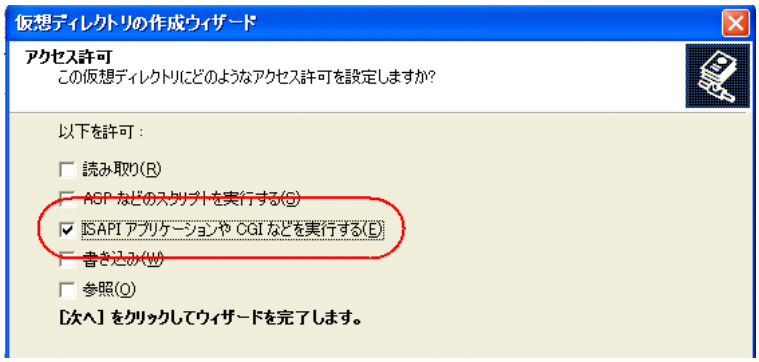
インターネット インフォメーション サービス (IIS) の設定は、「インターネット インフォメーション サービス (IIS) マネージャー」で行ないます。これは、次のいずれかの方法により起動できます。

1. Windows の「スタート」メニューから、「コントロールパネル → 管理ツール → インターネット インフォメーション サービス 」を開きます。
2. コマンドプロンプトから、「%windir%¥system32¥inetsrv¥InetMgr.exe」を実行します。
3. Windows Server 2003 の場合には、「スタートメ」ニューから、「サーバの役割管理 → アプリケーションサーバ → このアプリケーションサーバを管理する」を開きます。
4. Windows Server 2008 の場合には、「サーバー マネージャ」 を開き、「役割 → Web サーバ (IIS) → インターネット インフォメーション サービス (IIS) マネージャ」を開きます。

具体的なユーザインターフェースと手順は、Windows のバージョンにより異なります。以下の節では、Windows のバージョンごとに、インターネット インフォメーション サービス マネージャー での設定手順を説明していきます。

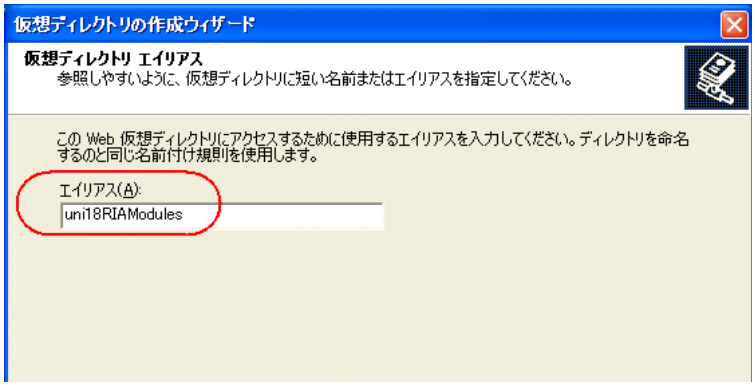
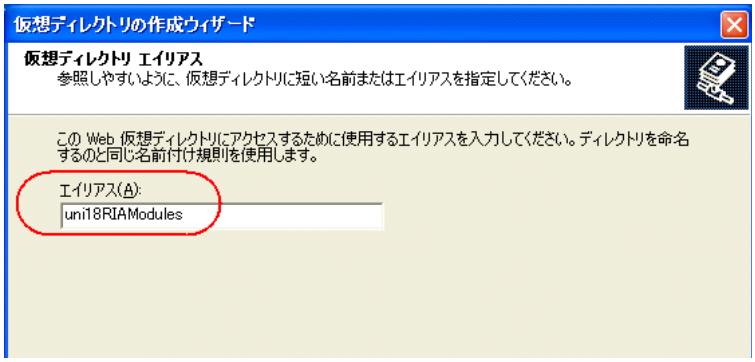
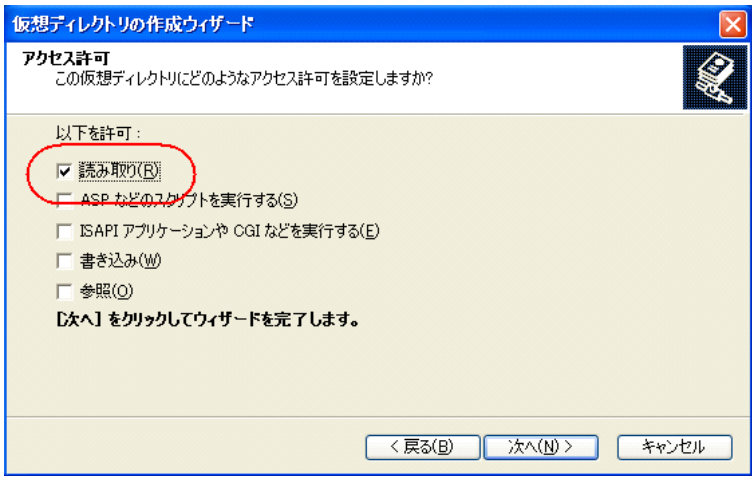
7.7 Windows XP での設定方法

7.7.1 Scripts ディレクトリの設定

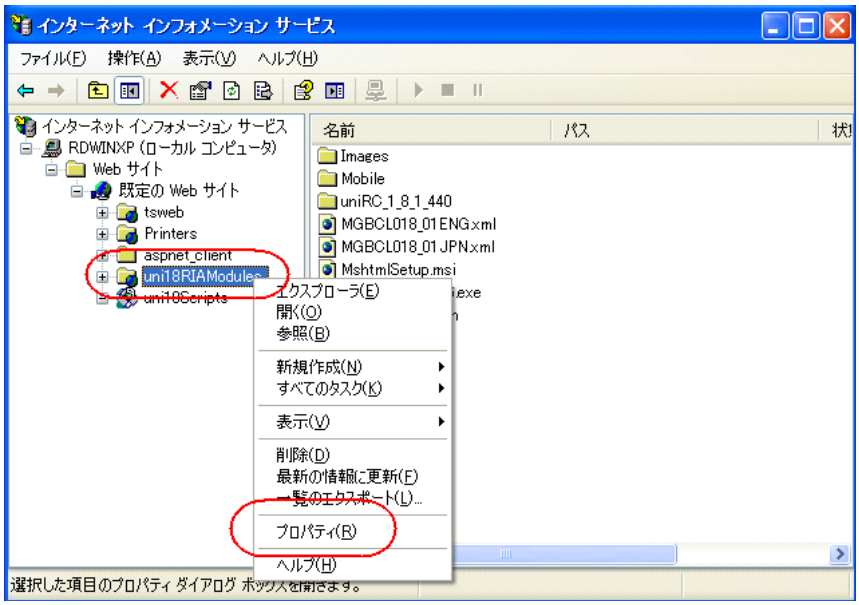
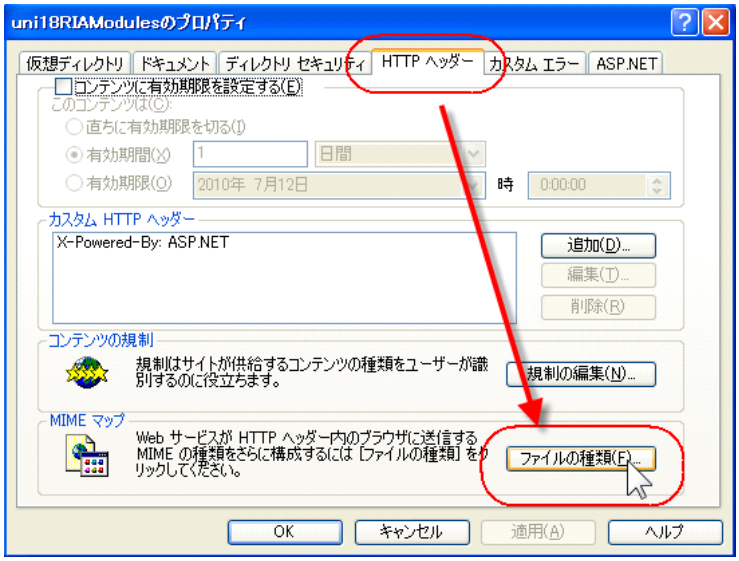
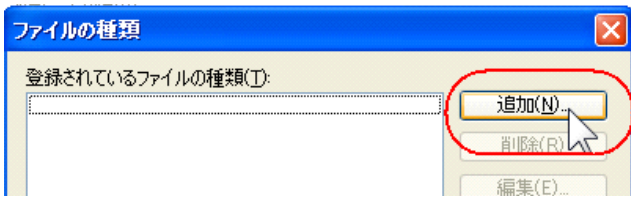
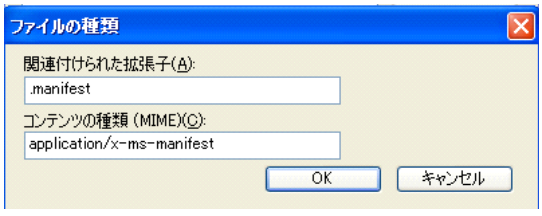
「既定の Web サイト → 新規作成 → 仮想ディレクトリ」により、仮想ディレクトリを作成します。	
エイリアスを指定します。	
物理ディレクトリを指定します。	
アクセス許可を指定します。	

7.7.2 RIAModules ディレクトリ

仮想ディレクトリの作成

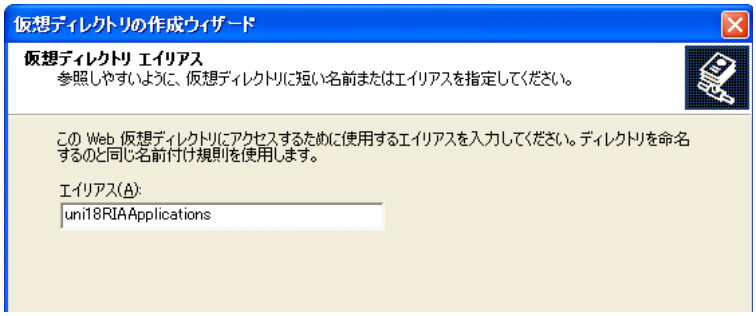
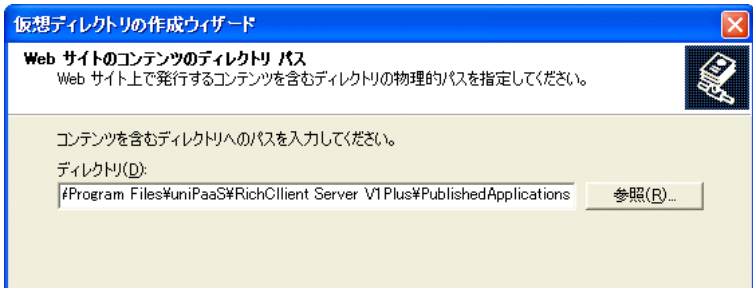
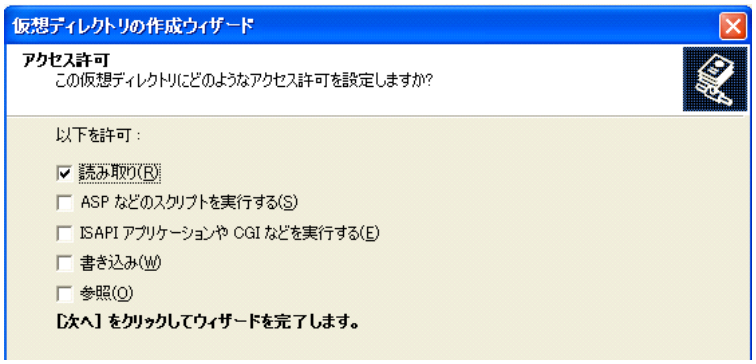
Scripts ディレクトリの場合と同様に、「既定の Web サイト → 新規作成 → 仮想ディレクトリ」により、仮想ディレクトリを作成します。 ます、エイリアスを指定します。	
物理ディレクトリを指定します。	
アクセス許可で、「読み取り」のみを許可します。	

MIME 設定

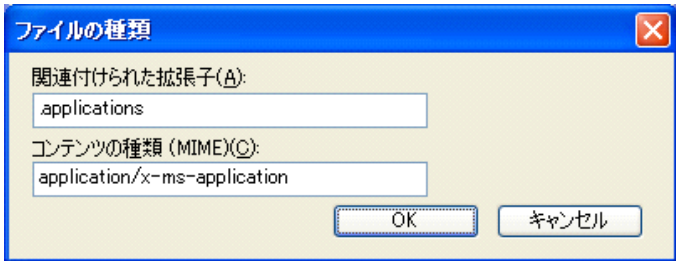
作成した仮想ディレクトリのプロパティを開きます。	
「HTTP ヘッダー タブ → ファイルの種類」を開きます。	
「追加」をクリックします。	
拡張子は.manifest、 コンテンツ情報は application/x-ms-manifest とします。	

7.7.3 PublishedApplications ディレクトリ

仮想ディレクトリ作成

RIAModules の場合と同様に、仮想ディレクトリを作成します。 まず、エイリアスを指定します。	
物理ディレクトリを指定します。	
アクセス許可は、「読み取り」だけを設定します。	

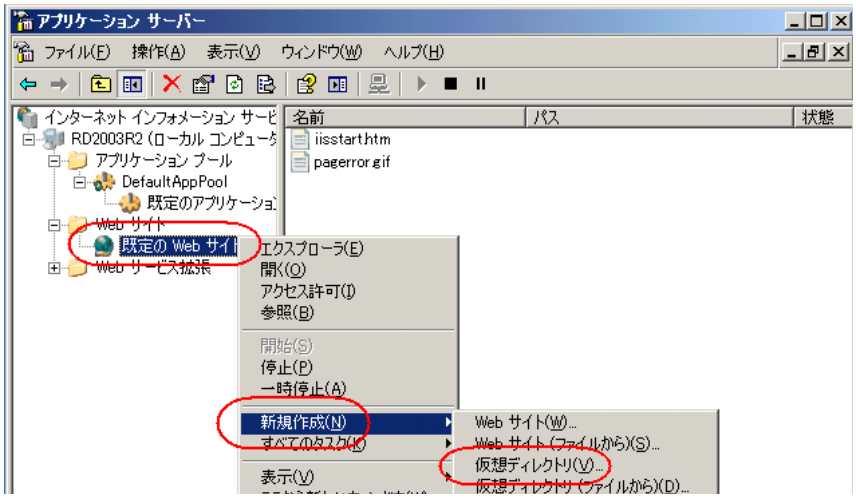
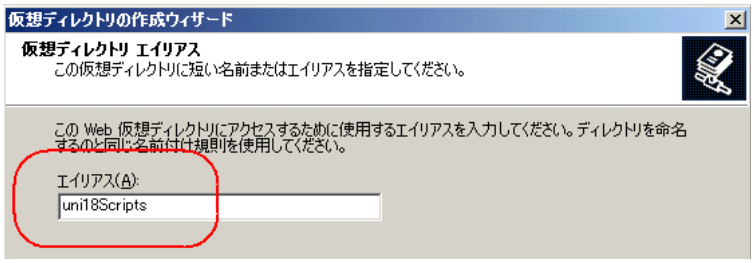
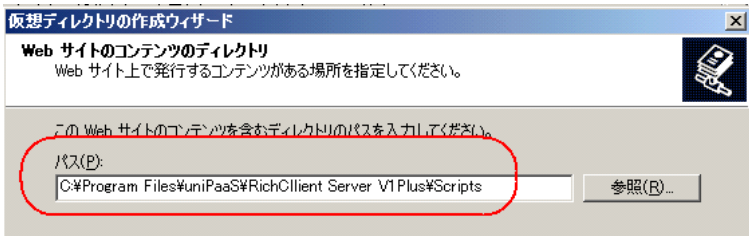
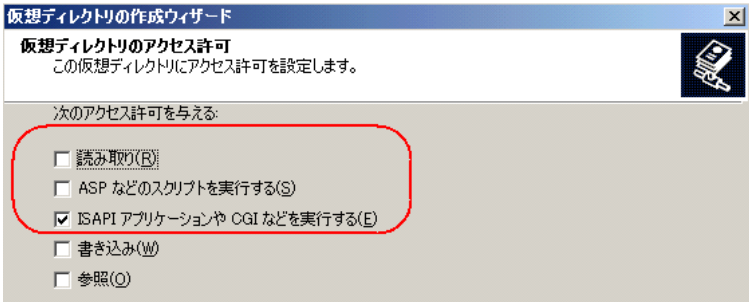
MIME 設定

RIAModules のときと同様に、仮想ディレクトリのプロパティを開き、「HTTP ヘッダー タブ → ファイルの種類 → 追加」で、.applications を追加します。	

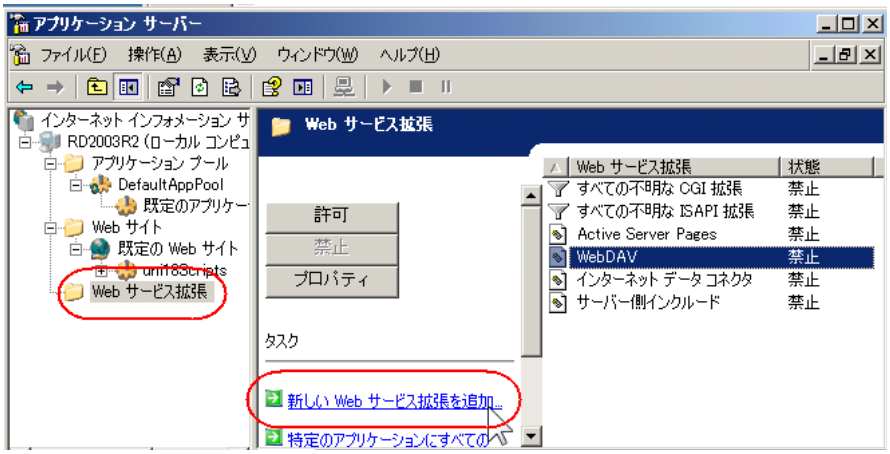
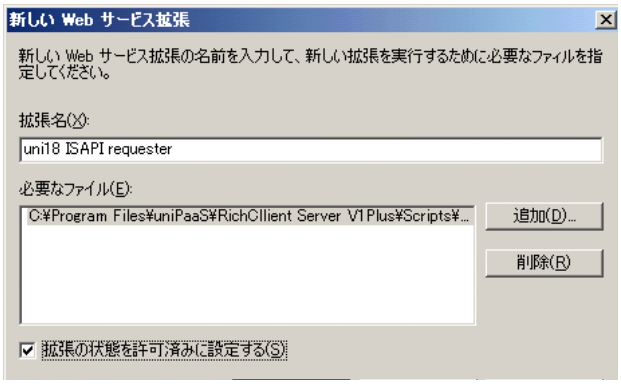
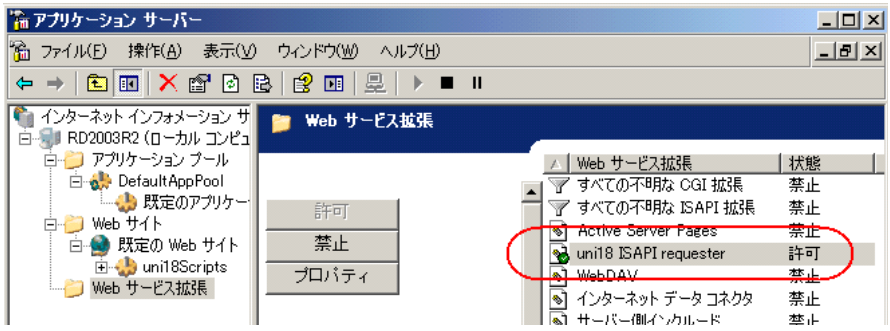
7.8 Windows Server 2003 の場合

7.8.1 Scripts ディレクトリの設定

仮想ディレクトリの作成

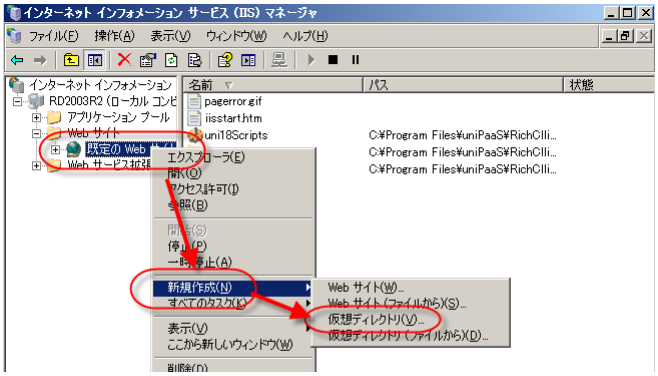
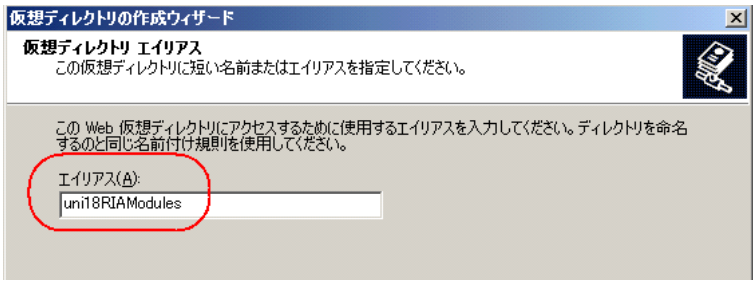
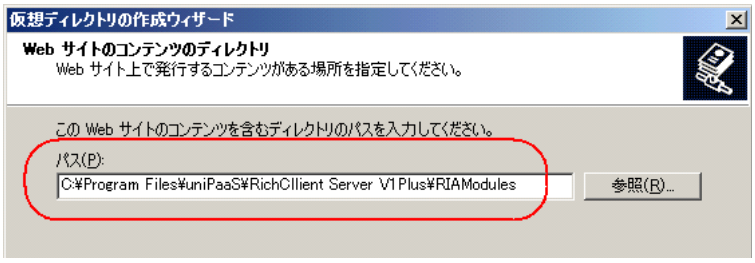
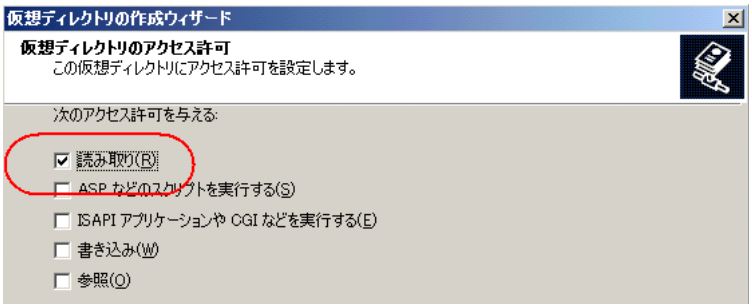
「既定の Web サイト → 新規作成 → 仮想ディレクトリ」 で仮想ディレクトリを作成します。	
エイリアスを指定します。	
物理ディレクトリを指定します	
アクセス許可を設定します。	

ISAPI の有効化

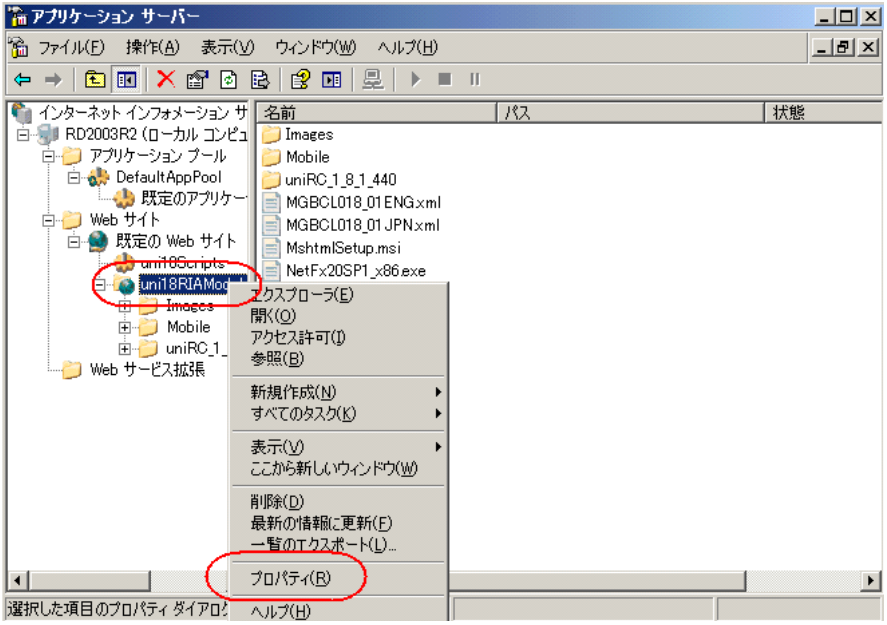
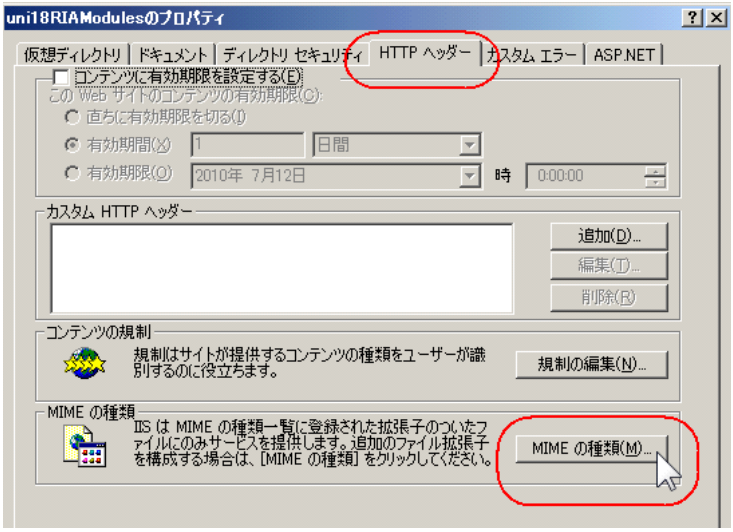
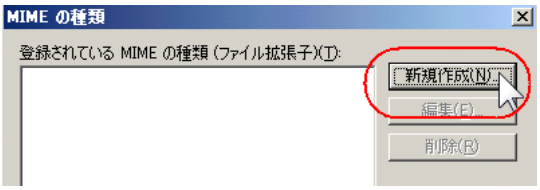
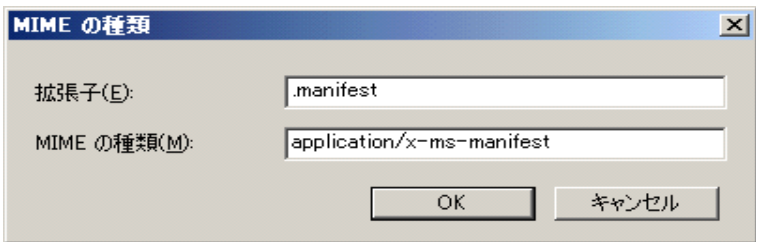
「Web サービス拡張 → 新しい Web サービス拡張を追加」 から、Web サービスの拡張許可を設定します。	
インターネットリクエスタを登録します。 「拡張名」は適当な名前を指定します。 「必要なファイル」 には、ISAPI リクエスタ DLL 名 (mgrqispi018.dll) をフルパスで指定します。 「拡張の状態を許可済みに設定する」 は オンにします。	
追加されたことを確認します。	

7.8.2 RIAModules ディレクトリ

仮想ディレクトリの作成

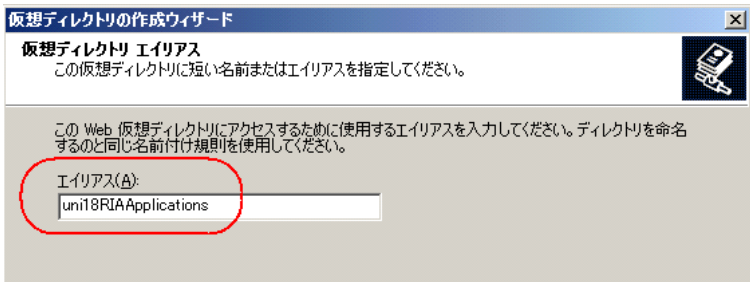
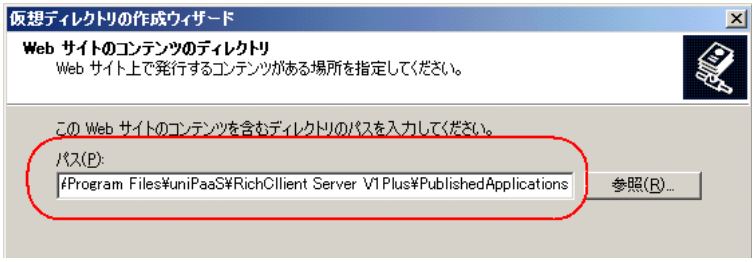
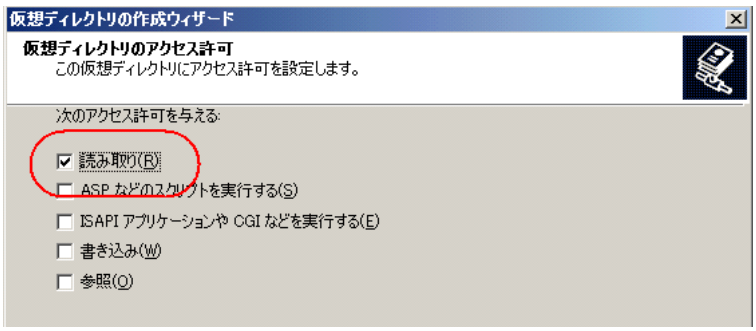
「規定の Web サイト」から、「新規作成 → 仮想ディレクトリ」を選択します。	
エイリアスを指定します。	
物理パスを指定します。	
アクセス許可は「読み取り」のみを設定します。	

MIME 設定

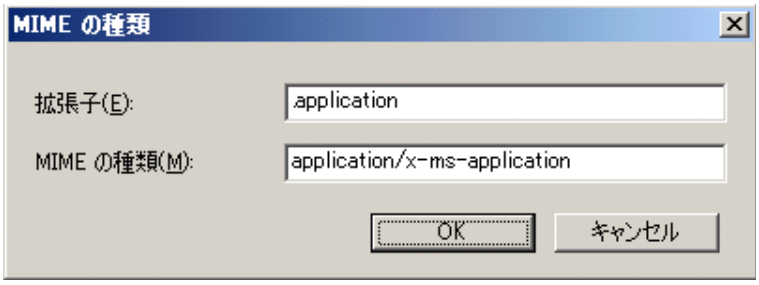
「既定の Web サイト → uni18RIAModules → プロパティ」を開きます。	
「HTTP ヘッダー タブ → MIME の種類」を開きます。	
「新規作成」を選びます。	
拡張子は .manifest、MIME の種類は application/x-ms-manifest とします。	

7.8.3 PublishedApplications ディレクトリ

仮想ディレクトリの作成

RIAModules の場合と同様に、仮想ディレクトリを作成します。 まず、エイリアスを指定します。	
続いて、物理パスを指定します。	
アクセス許可には、「読み取り」だけを許可します。	

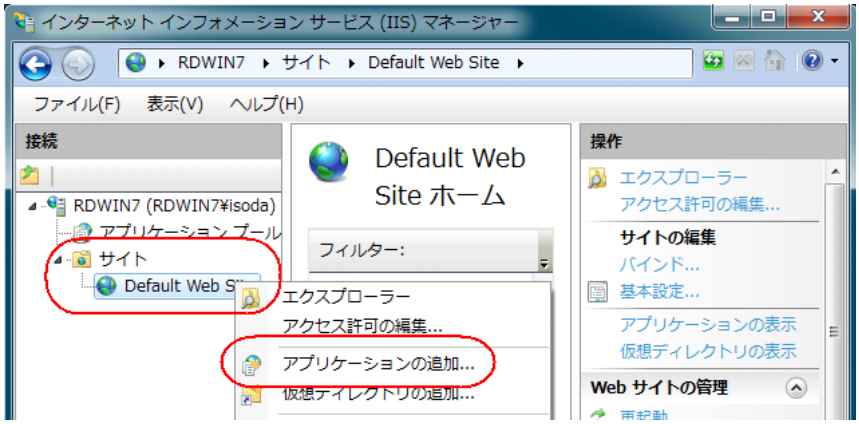
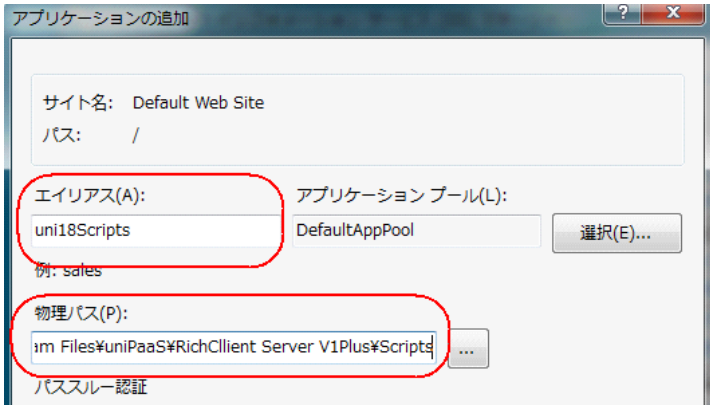
MIME 設定

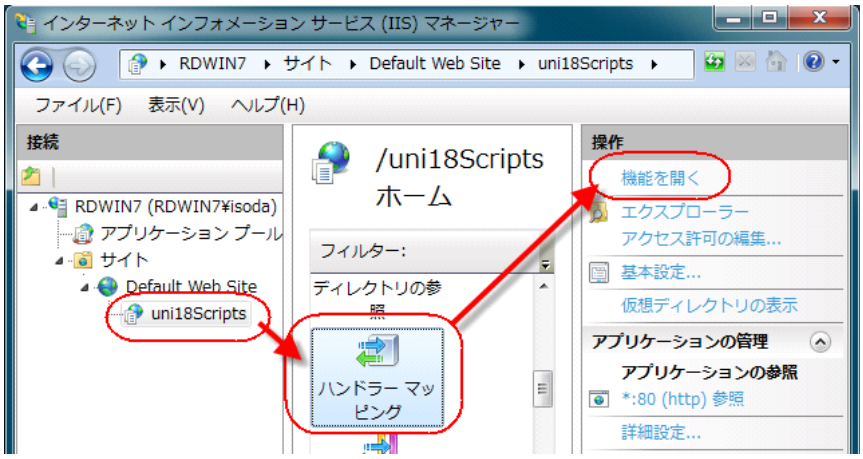
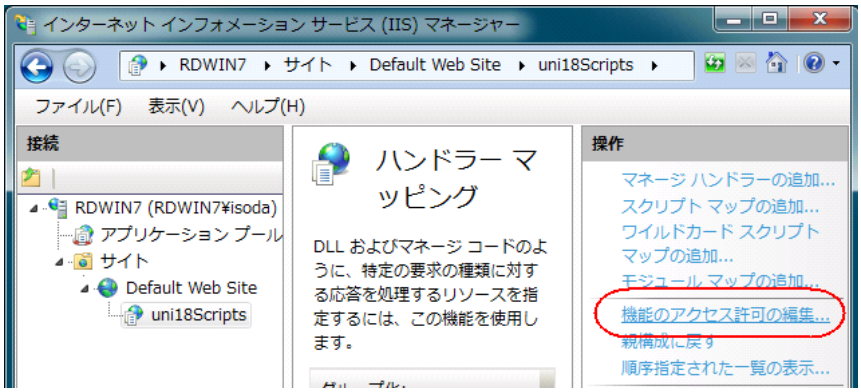
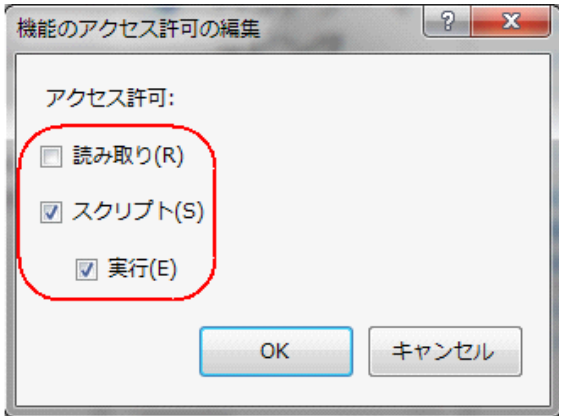
RIAModules と同様の手順で、MIME の種類を指定します。 拡張子は .application MIME の種類は application/x-ms-application	

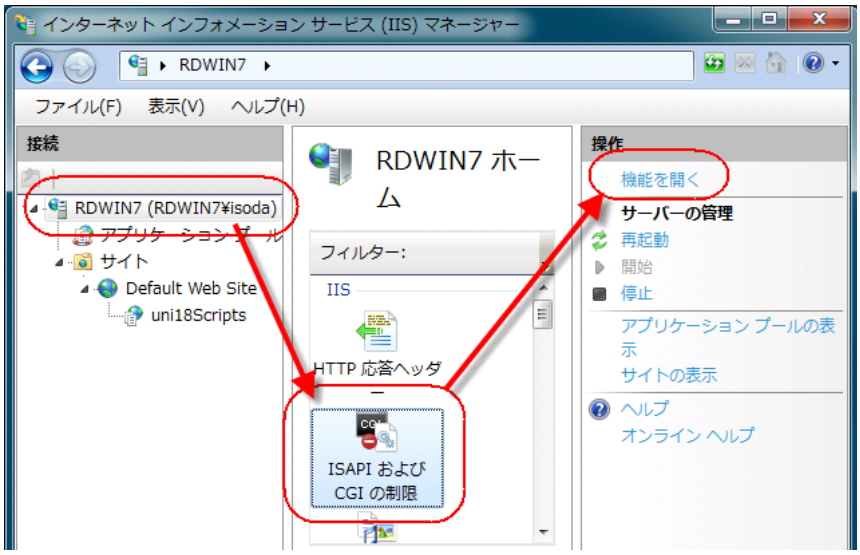
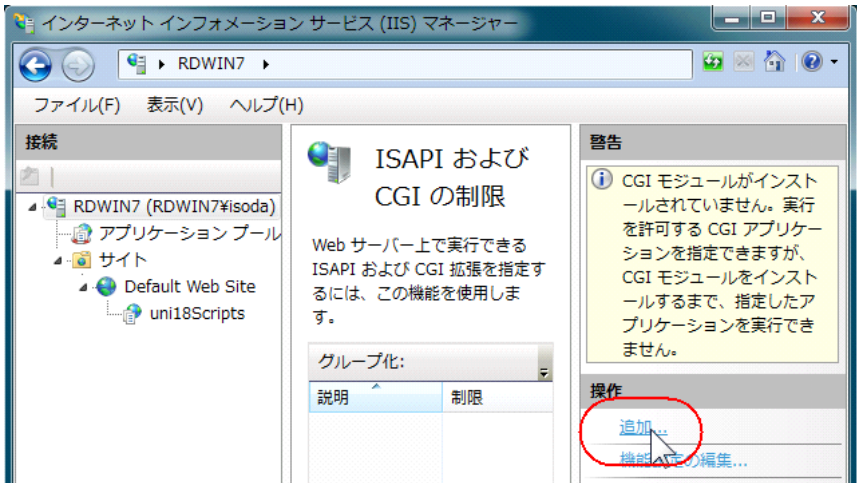
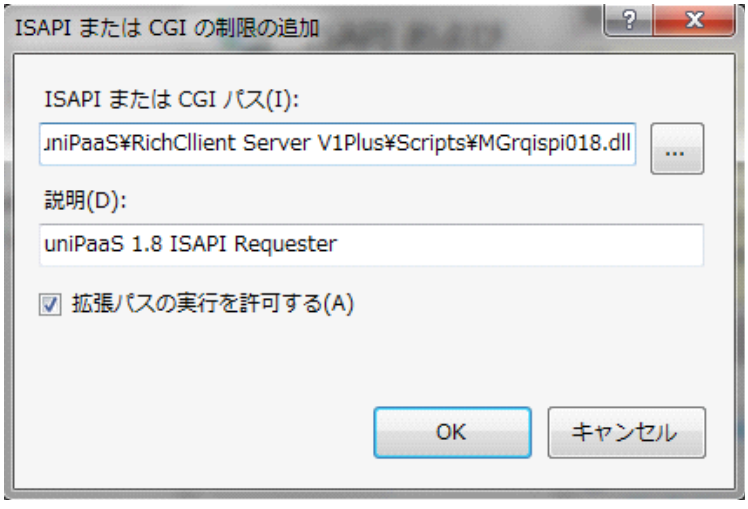
7.9 Windows Vista、Windows 7 の場合

Windows Vista、Windows 7 では、拡張子 .manifest および .application についての MIME の設定はすでになされているので、RIAModules および RIAApplication での「MIME の種類」の追加定義の必要はありません。

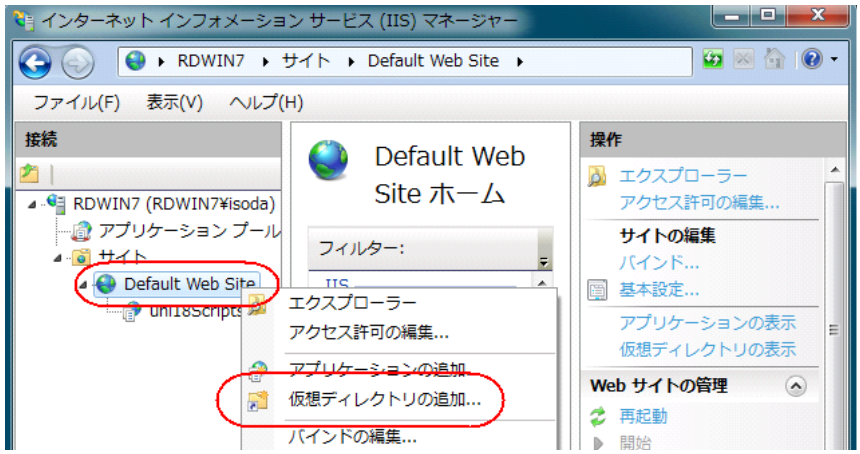
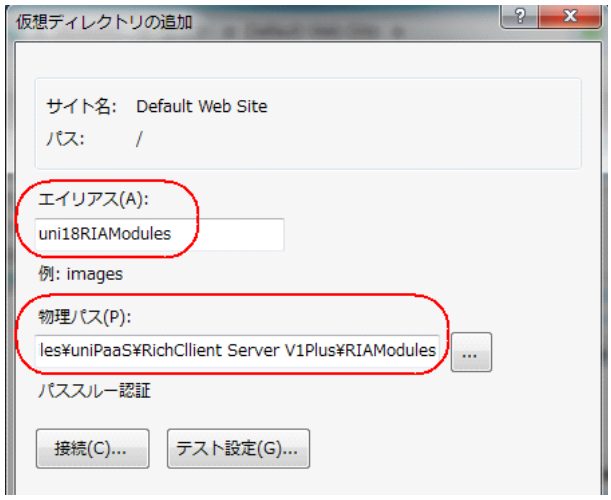
7.9.1 Scripts ディレクトリの設定

「サイト → Default Web Site → アプリケーションの追加」で、アプリケーション（実行可能な仮想ディレクトリ）を追加します。	
エイリアスと物理パスの情報を設定します。	

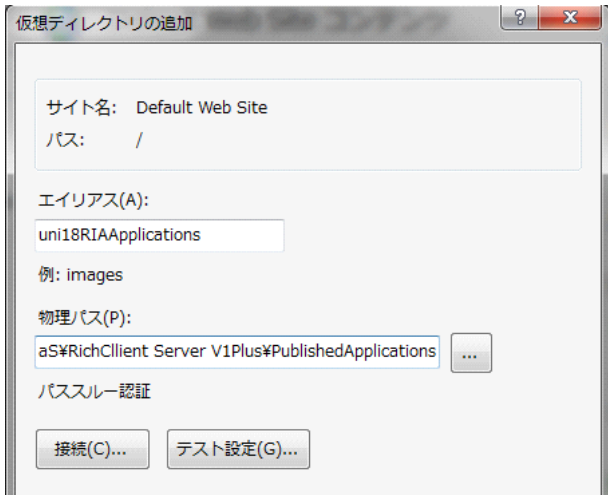
「ハンドラーマッピング」により、 アクセス許可を設定します。	
	
アクセス許可として、 読み取りはオフ スクリプトと実行はオン に設定します。	

ISAPI および CGI の制限を設定します。 「(トップ) → ISAPI および CGI の制限 → 機能を開く」を開きます。	
「追加」を選択します。	
リクエスタのパスを指定します。 また、「拡張パスの実行を許可します」にチェックを入れます。	

7.9.2 RIAModules ディレクトリ

Default Web Site から、「仮想ディレクトリの追加」を選びます。	
エイリアス、物理パスを指定します。	

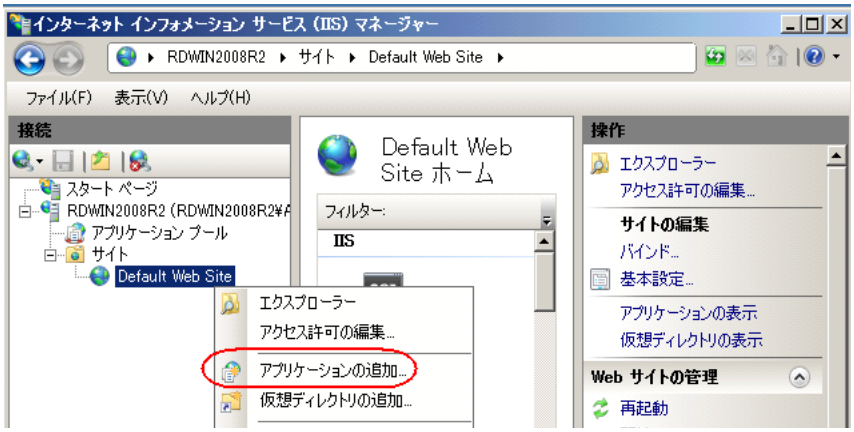
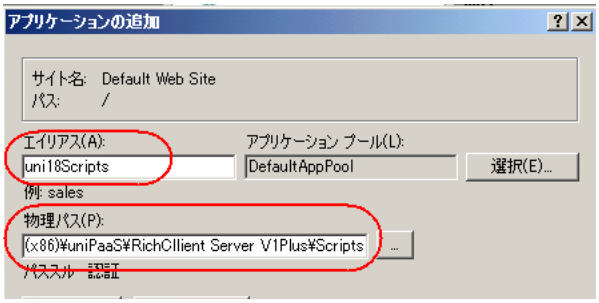
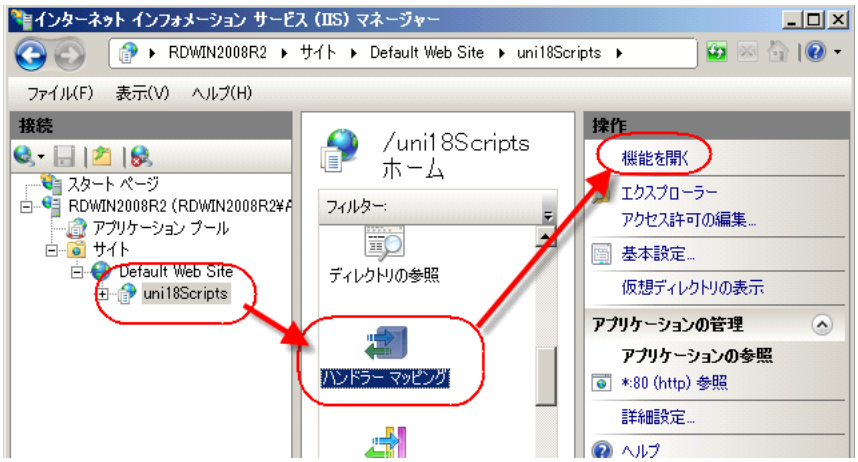
7.9.3 PublishedApplications ディレクトリ

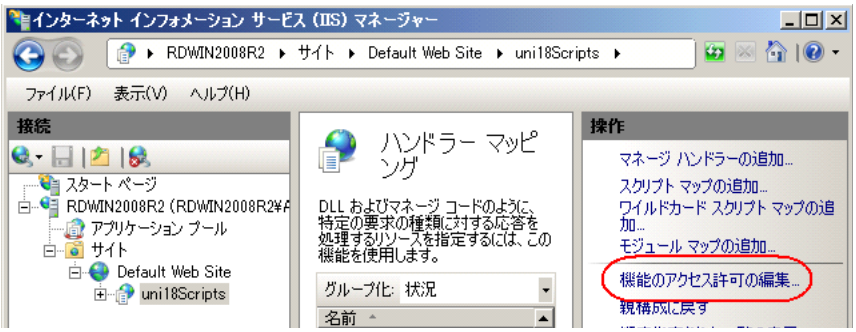
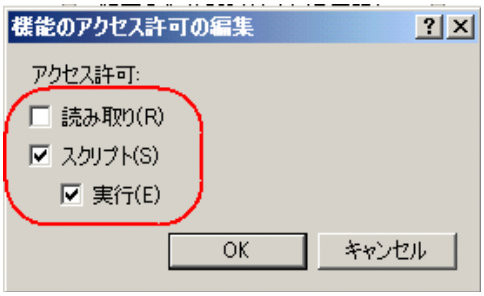
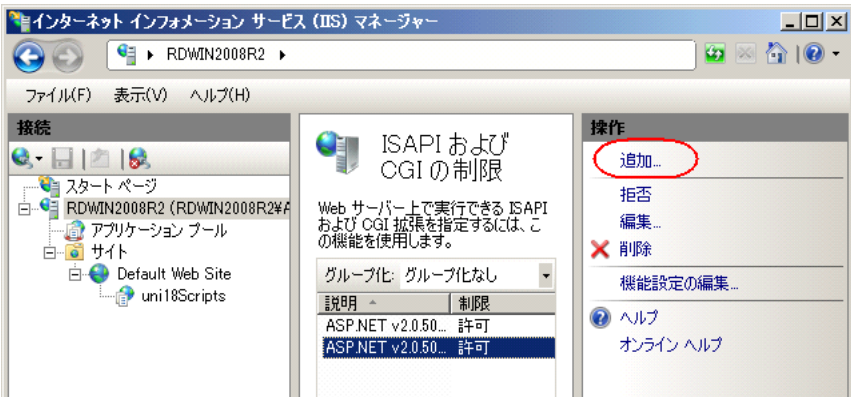
RIAModules の場合と同様の手順で、仮想ディレクトリを定義します。	

7.10 Windows Server 2008 の場合

Windows Server 2008 では、拡張子 .manifest および .application についての MIME の設定はすでになされているので、RIAModules および PublishedApplications での「MIME の種類」の追加定義の必要はありません。

7.10.1 Scripts ディレクトリの設定

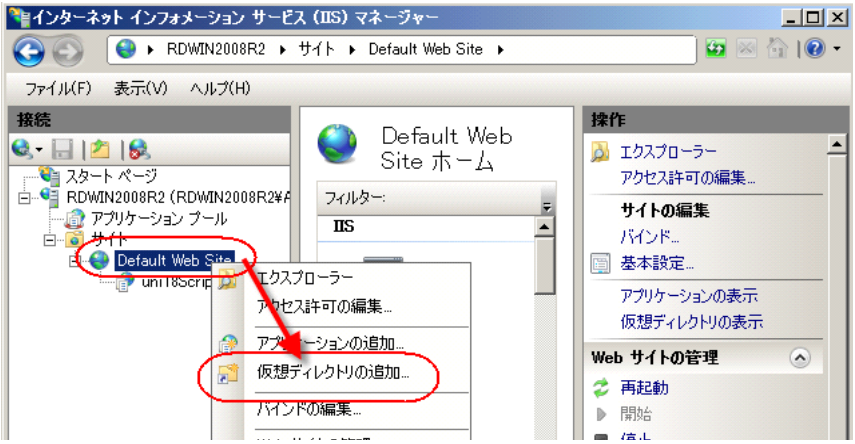
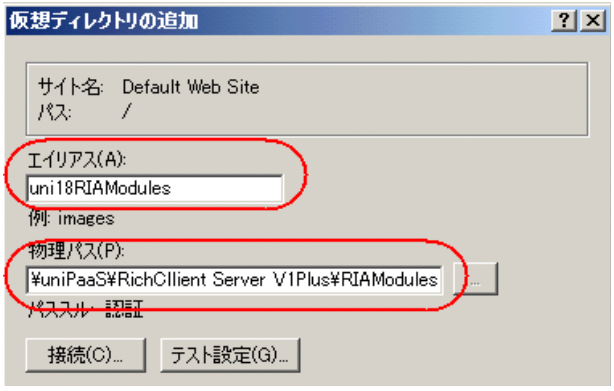
アプリケーション（実行可能な仮想ディレクトリ）を追加します。	
エイリアスと実行パスを指定します。	
ハンドラマッピングを指定します。	

「機能のアクセス許可の編集」を選びます。	
読み取りはオフ、 スクリプトと実行はオン にします。	
ISAPI および CGI の制限の設定をします。	
「追加」を選びます。	

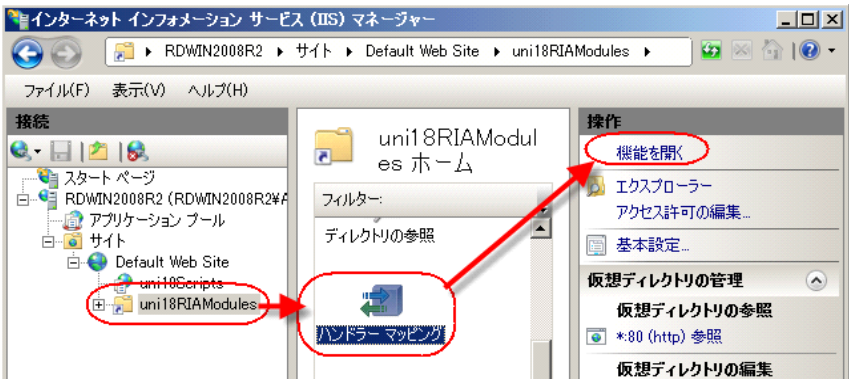
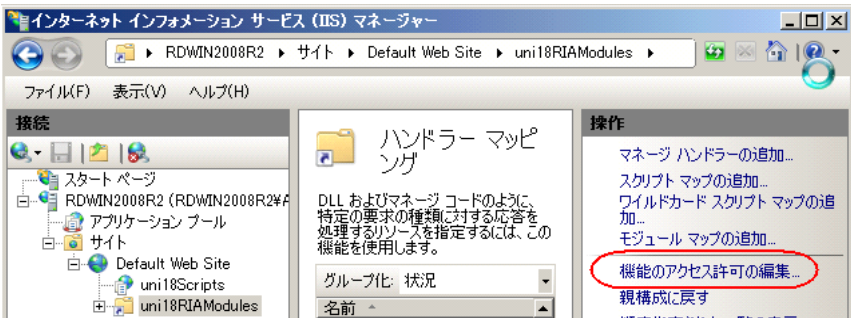
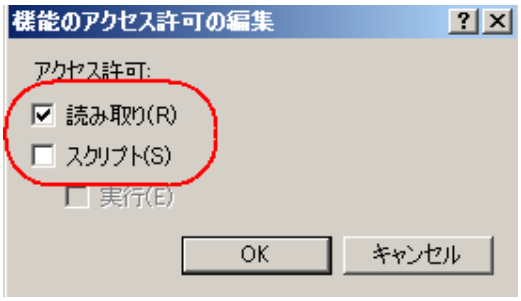
インターネットリクエスタのパスを指定します。 また、「拡張パスの実行を許可する」にチェックを入れます。	

7.10.2 RIAModules ディレクトリ

仮想ディレクトリの作成

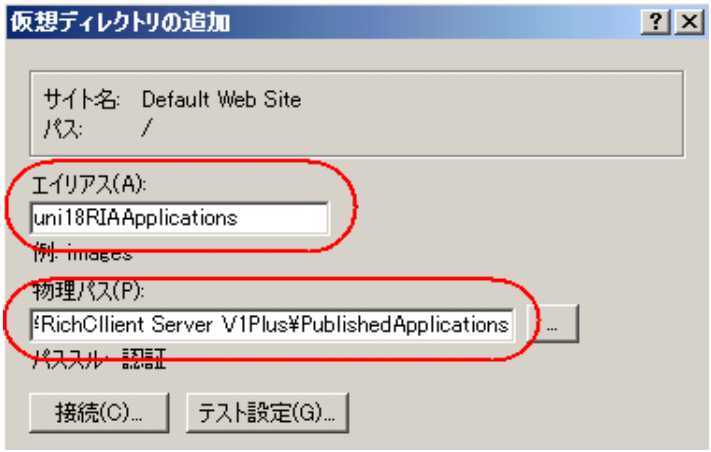
Default Web Site から、「仮想ディレクトリの追加」を選びます。	
エイリアスと物理パスを指定します。	

ハンドラー マッピング

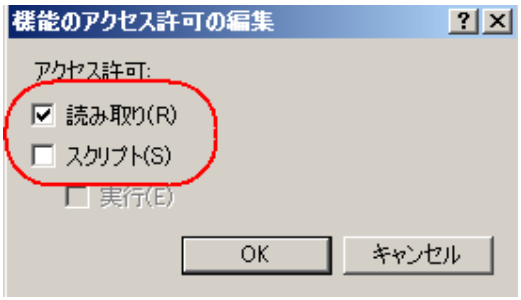
作成した仮想ディレクトリ uni18RIAModules から、「ハンドラー マッピング → 機能を開く」を選びます。	
「機能のアクセス許可の編集」を選びます。	
アクセス許可は「読み取り」のみを許可します。	

7.10.3 PublishedApplications ディレクトリ

仮想ディレクトリの作成

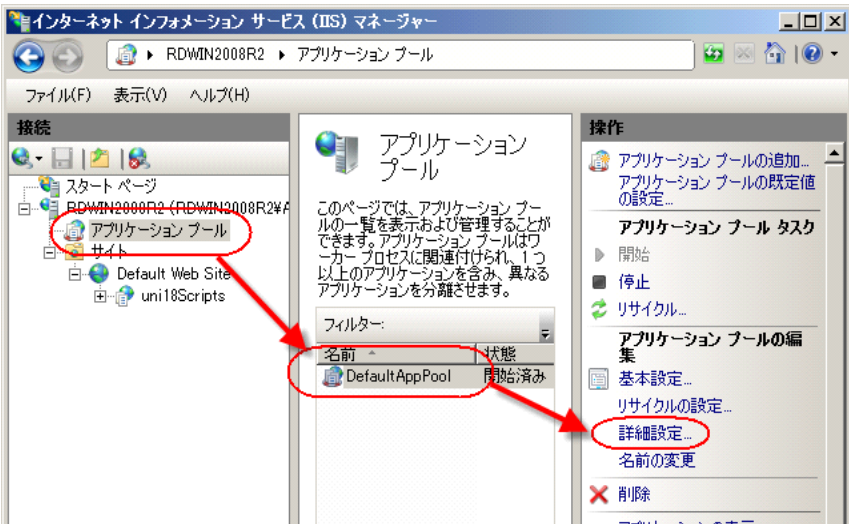
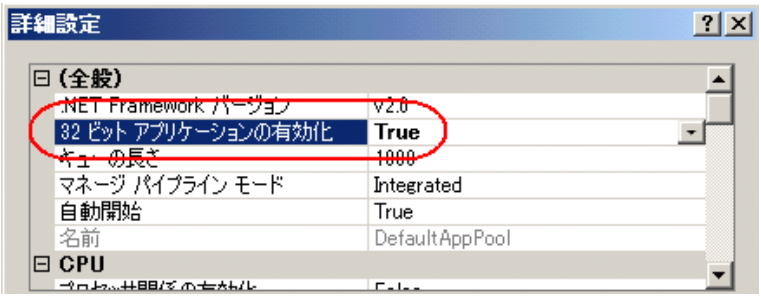
RIAModules の場合と同様の手順で、仮想ディレクトリを定義します。	

ハンドラー マッピング

RIAModules の場合と同様に、「ハンドラーマッピング」から「アクセス許可」を設定します。ここでも「読み取り」のみを許可します。	

7.11 X64 サーバでの設定

64 ビット版の Windows Server 2008 では、IIS も 64 ビット版なので、uniPaaS の 32 ビットリクエスタをそのまま使えませんので、上記設定に加え、ISAPI に 32 ビット DLL を利用できるように設定する必要があります。

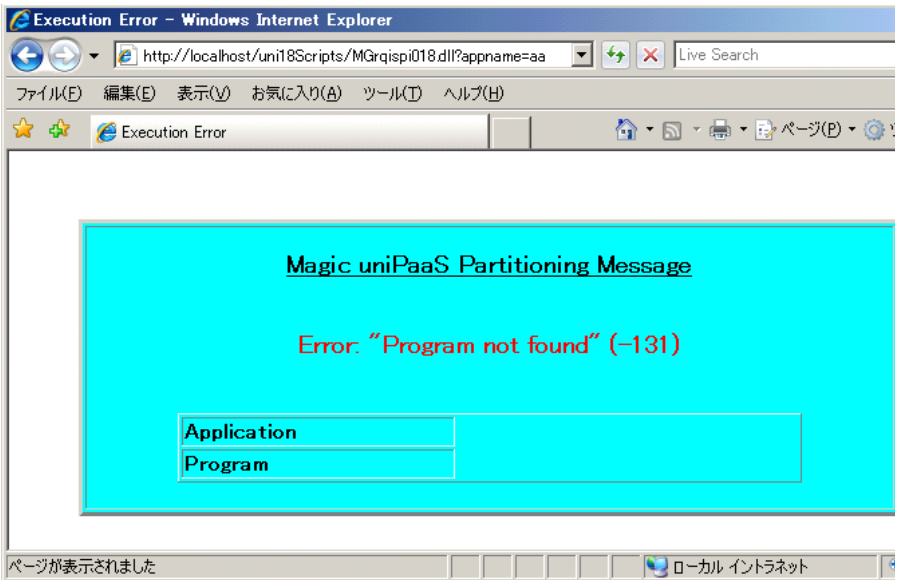
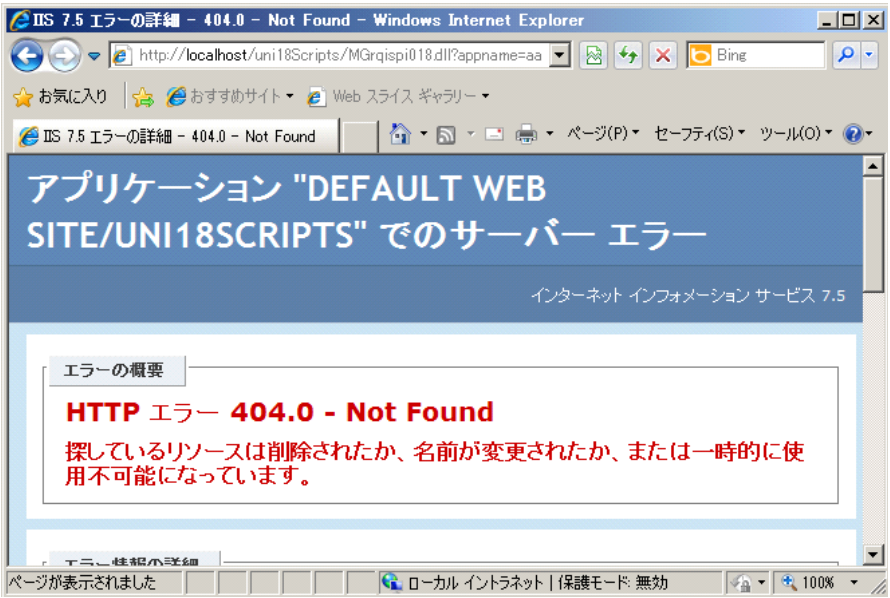
「アプリケーションプール → DefaultAppPool → 詳細設定」を選びます。	
「32 ビットアプリケーションの有効化」を True にします。	

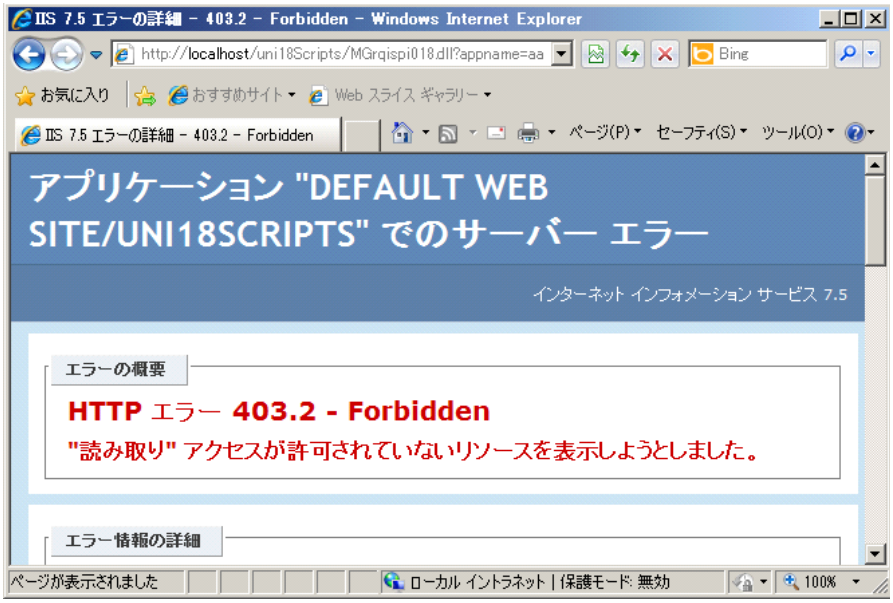
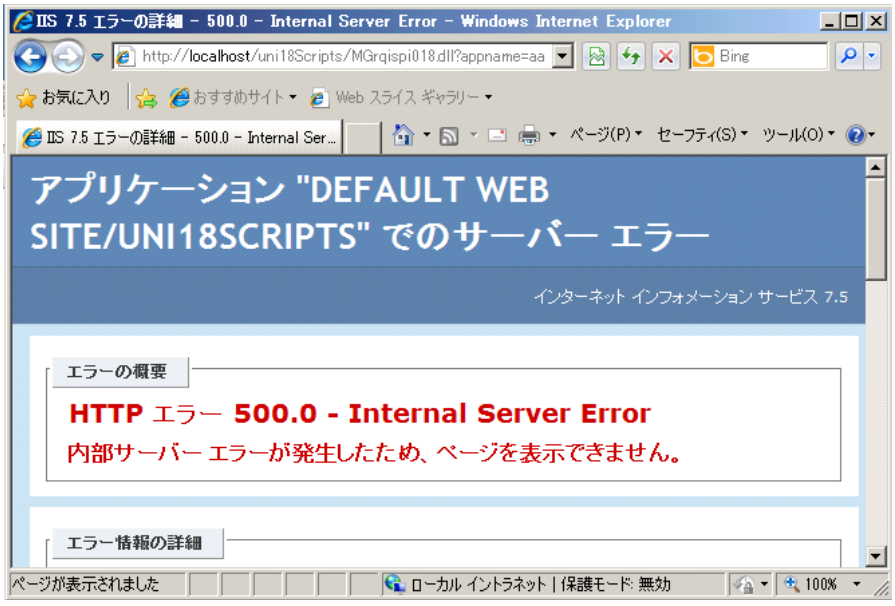
7.12 確認方法

7.12.1 Scripts ディレクトリの確認

Scripts ディレクトリが正しく設定されたかは、インターネットリクエスタを呼び出してみればわかります。ここでは、意図的にエラーが発生するようなパラメータを渡してみて、インターネットリクエスタがエラーメッセージを出すかを確認してみます。

Web ブラウザを起動してみて、URL として、http://localhost/uni18Scripts/MGrqispi018.dll?appname=aa を入力してみます。この URL は、インターネットリクエスタ Mgrqispi018.dll を呼び出しているのですが、パラメータとして、appname のみが指定されており、prgname が指定されていないため、インターネットリクエスタが正しく動作していれば、インターネットリクエスタがエラーメッセージを表示します。

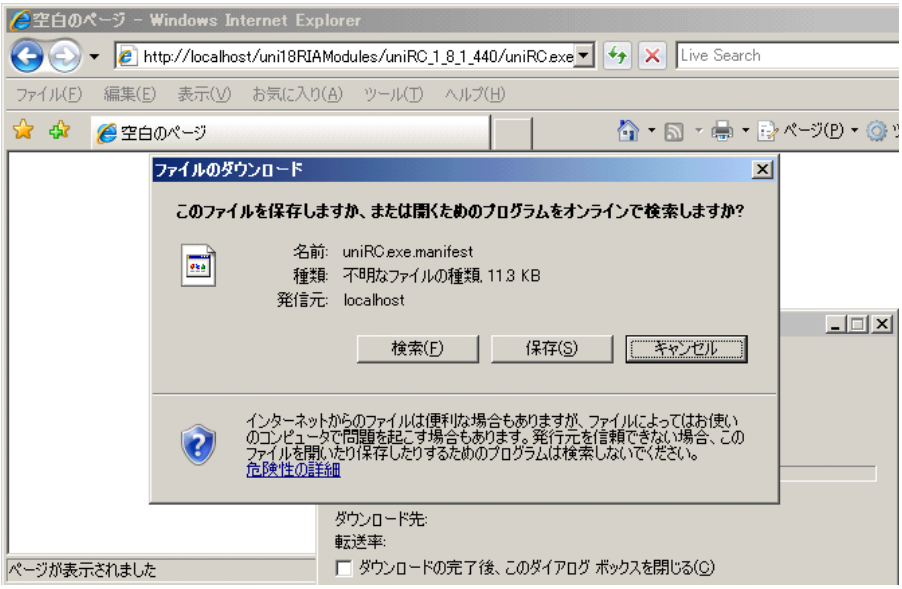
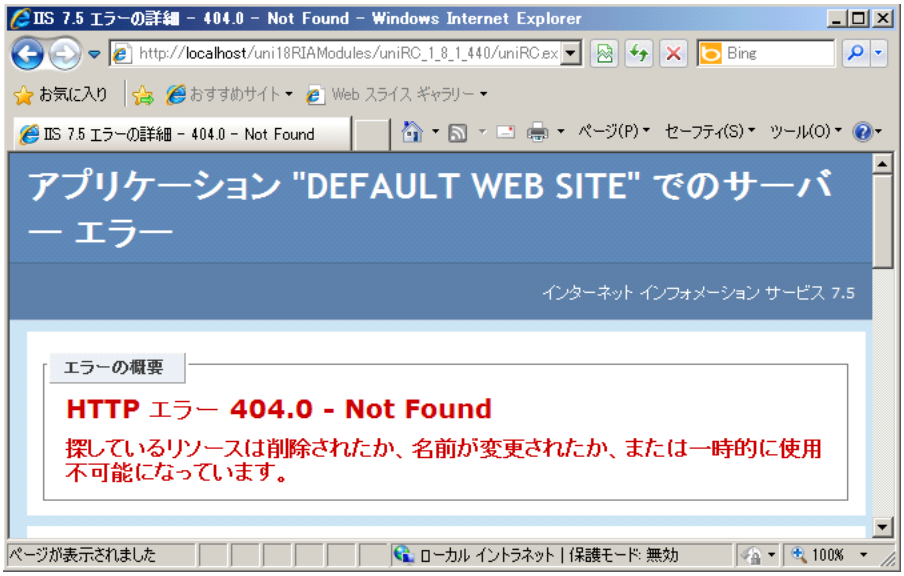
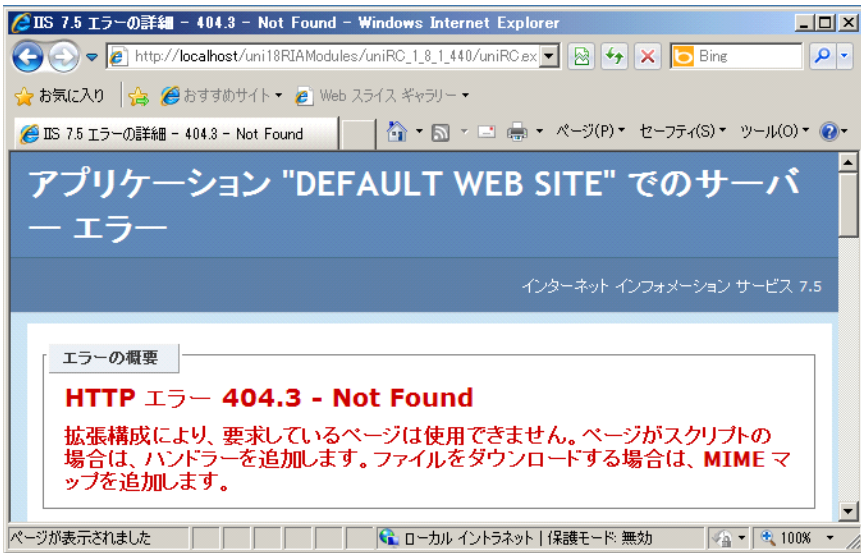
正しく動作していれば、右図のようなエラー画面が出るはずです。	
もし、仮想ディレクトリの名前や、物理パスの設定が間違っていると、次のような「Not Found」のエラーとなります。	

もし、ここでの ISAPI 実行が許可されていなければ、右図のような「Forbidden」のエラーが出ます。	
Windows Server 2008 の x64 Edition の場合、「32 ビットアプリケーションの有効化」が True になっていないと、右図のような「Internal Server Error」のエラーが出ます。	

7.12.2 RIAModules ディレクトリの確認

RIAModules が正しく設定されているかは、このディレクトリからリッチクライアントモジュールのマニフェストファイルをダウンロードしてみることにより確認できます。

Web ブラウザを開いて、URL として http://localhost/uni18RIAModules/uniRC_1_8_1_440/uniRC.exe.manifest と入力してみてください。(「 uniRC_1_8_1_440」の部分は、uniPaaS のバージョンにより異なります。RIAModules に実際に存在するディレクトリ名に合わせて変更してください。)

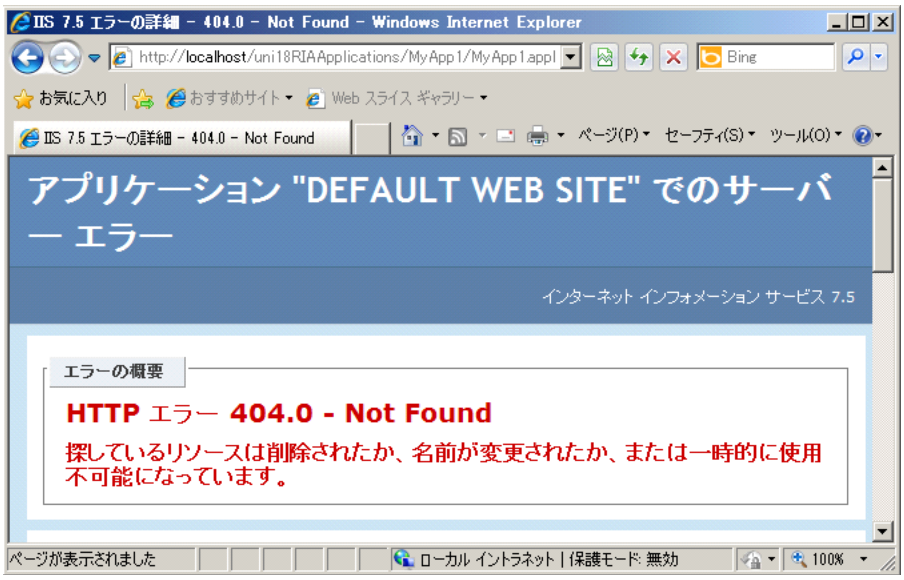
右図のようなダウンロードダイアログが出れば OK です。 確認後、ダウンロードはキャンセルしてください。	
エイリアスや物理パスが間違っていると、右図のような「Not Found」のエラーが出ます。	
MIME の種類として .manifest が登録されていないと、右図のようなエラーが出ます。 上のエラーと似ていますが、エラーコードが 404.3 となっています。	

7.12.3 PublishedApplications ディレクトリの確認

PublishedApplications ディレクトリが正しく設定されているかは、デプロイメントマニフェストファイル(MyApp1.applications など)をダウンロードしてみれば、確認できます。

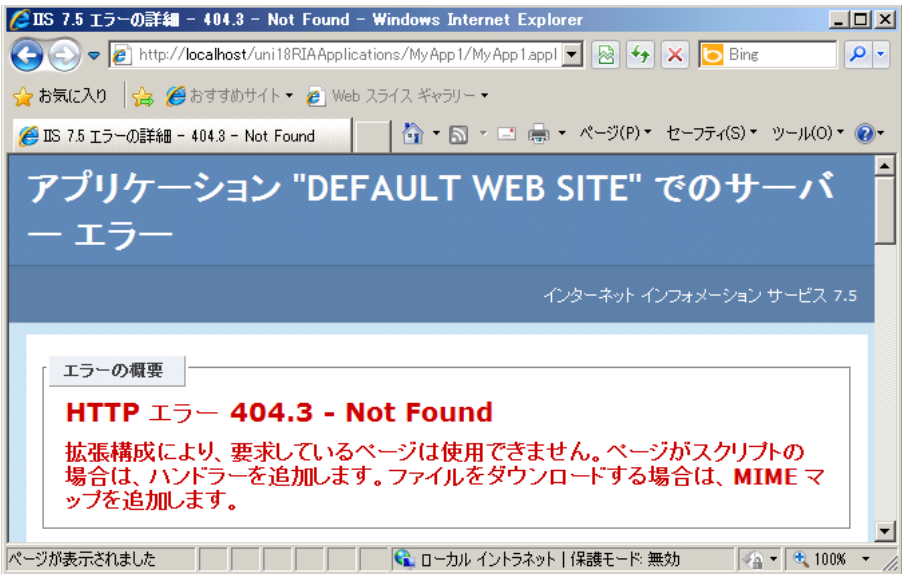
ここでは、Studio で作成されたデプロイメントマニフェストファイル MyApp1.application と、myapp1.publish.html ファイルとが、PublishedApplications¥MyApp1 にコピーされているものと仮定します。

Web ブラウザを開いて、URL として http://localhost/uni18RIAApplications/MyApp1/MyApp1.application を指定してください。

正しく設定されていると、右図のように「アプリケーションの起動中」というダイアログが出ます。これは、デプロイメントマニフェストファイルが正しくダウンロードされて、ClickOnce の機能が起動されたことを示しています。 ここではダウンロードが正しく行われたことを確認するだけなので、キャンセルして構いません。	
エイリアスや物理パスの名前が間違っていると、右図のような「Not Found」のエラーが出ます。	

MIME の種類として .application が登録されていないと、右図のようなエラーが出ます。

上のエラーと似ていますが、エラーコードが 404.3 となっています。



Magic uniPaaS V1Plus

サーバ構成

第 2 版　2010 年 8 月 25 日

発行　〒151-0053 東京都渋谷区代々木三丁目二十五番地三号
あいおい損保新宿ビル 14 階
マジック ソフトウェア・ジャパン (株)
http://www.magicsoftware.co.jp/