ものづくり・IT 融合化推進技術の研究開発

MZ Platform データ連携機能

= 導入手順書 =

Revision 2.0 [MZ Platform.2.0]

National Institute of
Advanced Industrial Science
and Technology
AIST

独立行政法人
産業技術総合研究所

＝ 目次 ＝

1. データ連携機能を使用するための初期設定 1

1.1. 環境設定（PLATFORM.INI の編集） 1
1.2. 接続先の設定（HOSTSFILE.INI の編集） 2

2. コンポーネント連携機能 3

2.1. リモートコンポーネントの設定 3

3. コンポーネント転送機能 5

3.1. PULL 型／PUSH 型 5
3.2. 様々なコンポーネント転送 6

A. 【付録】データ連携（旧バージョン）のサーバ環境設定 1

A.1. サーバ環境の準備 1
A.1.1. Sun Java System Application Server のダウンロード 1
A.1.2. 事前設定（レジストリサーバ） 1
A.1.3. 事前設定（ブローカ） 2
A.2. 管理サーバの起動と終了 3
A.3. WEB アプリケーションの配備 4
A.4. WEB アプリケーションの起動と終了 6
A.4.1. アプリケーションサーバーインスタンスの起動と終了 6

B. 【付録】データ連携（旧バージョン）の操作手順 9

B.1. 初期設定（PLATFORM.INI の編集） 9
B.2. コンポーネント連携 10
B.2.1. リモートコンポーネント選択 10
B.2.2. リモートコンポーネント入力 10
B.3. コンポーネント転送 11

1. データ連携機能を使用するための初期設定

1.1. 環境設定（Platform.ini の編集）

配布された Platform.ini を、環境に合わせて変更する。

1) データ連携を使用するかのフラグ：UseDataCooperation を true（使用する）に変更する
2) 旧バージョンのデータ連携機能を使用しないかのフラグ：UseOnlyLightDataCooperation を true（使用しない）に設定する。（もし旧バージョンと併用する場合は、このフラグを false に設定する）
3) プラットフォーム名：PlatformName に一意なプラットフォーム名を設定する
4) ローカルホストアドレス：LocalhostAddress にローカルホストアドレスを設定する
5) RMI ポート：RMIPort に RMI レジストリが使用するポート番号を設定する。この項目の記入は任意であり、未記入の場合はデフォルトのポート番号（1099）が自動的に設定される

以下はプラットフォーム名を“Platform01”、ローカルホストアドレスを“xxx.example.com”とした場合の Platform.ini の記述例である。

```
LogLevel=1
LookAndFeel=
RuntimeLocale=
ComponentListFile_ja=etc¥¥PlatformComponents_ja.ini
ComponentListFile_en=etc¥¥PlatformComponents_en.ini
ComponentListFile=etc¥¥PlatformComponents_en.ini
ComponentInformationFolder=components
BinaryDataAutoSave=true
UseDataCooperation=true
UseOnlyLightDataCooperation=true
BrokerAddress=
PlatformName=Platform01
LocalhostAddress=xxx.example.com
datamanagement.default_componentcooperation_policy=true
datamanagement.default_componentpulltransfer_policy=true
datamanagement.default_componentpushtransfer_policy=true
java.rmi.server.codebase=
java.security.policy=etc¥¥java.policy
brokermonitor.broker_access_log=
brokermonitor.broker_debug_log=
RMIPort=
CombinativeComponentsFolder=AP_DATA_COMB
```

※各プロパティについての詳細は、「詳細設定説明書」に記されているのでそちらを参照のこと。

1.2. 接続先の設定（hostsfile.ini の編集）

配布された hostsfile.ini に、データ連携機能を利用するプラットフォーム群の情報を記入する。

このファイルは、接続先となる外部プラットフォームのプラットフォーム名とホスト名、ポート番号を前もって関連付けておくための設定ファイルである。

ここに記入されるホスト名とは、各プラットフォームの Platform.ini において LocalhostAddress プロパティに設定されている値のことであり、完全に一致していなければならない。

以下に、関連付けの書式を示す。

```
<Platform Name>__<Host Name(or IP Address)>:Port Number
```

プラットフォーム名とホスト名の間は、タブで区切る。

ポート番号は、ホスト名の後ろに「:」（コロン）を加えて記入する。

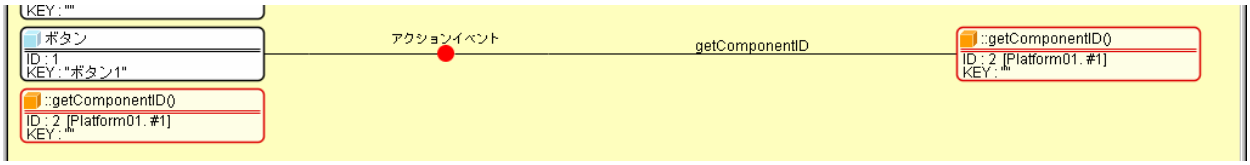
（ポート番号を省略する場合には、コロンも省略すること）

記入例：

PlatformA	xxx.example.com:3000
PlatformB	127.0.0.1:3001
PlatformC	xxx.example.com　（←ポート番号が未記入＝デフォルトポート(1099)）

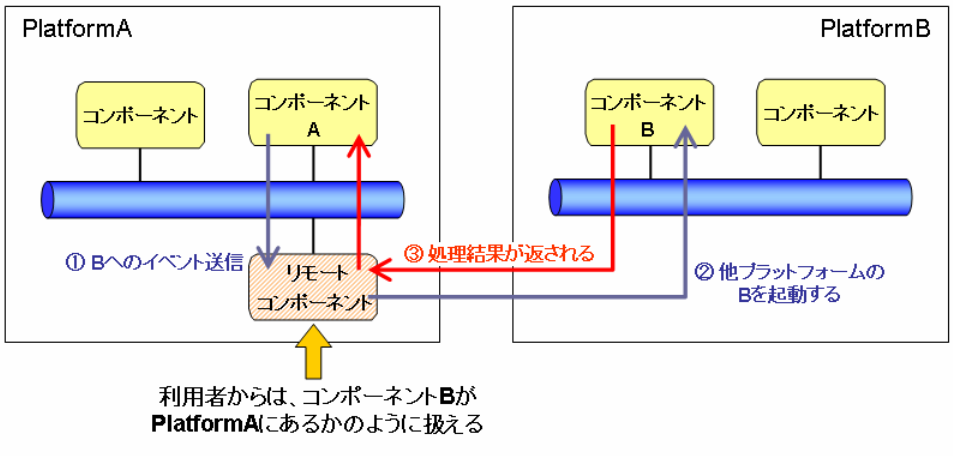
2. コンポーネント連携機能

コンポーネント連携機能は、起動メソッドの接続コンポーネントにビルダー上のコンポーネントを選択する代わりに、リモートコンポーネントを設定して使用する。



リモートコンポーネントは実体としてビルダー上に配置されるコンポーネントではなく、コンポーネント連携機能で呼び出すメソッドの情報を保持する仮想コンポーネントである。

ビルダー上でリモートコンポーネントを示しているオレンジ色の四角は、コンポーネント連携機能を用いることで、外部のプラットフォームにあるコンポーネントが、あたかも手元のプラットフォームに存在するかのような振る舞いをすることを表現している。

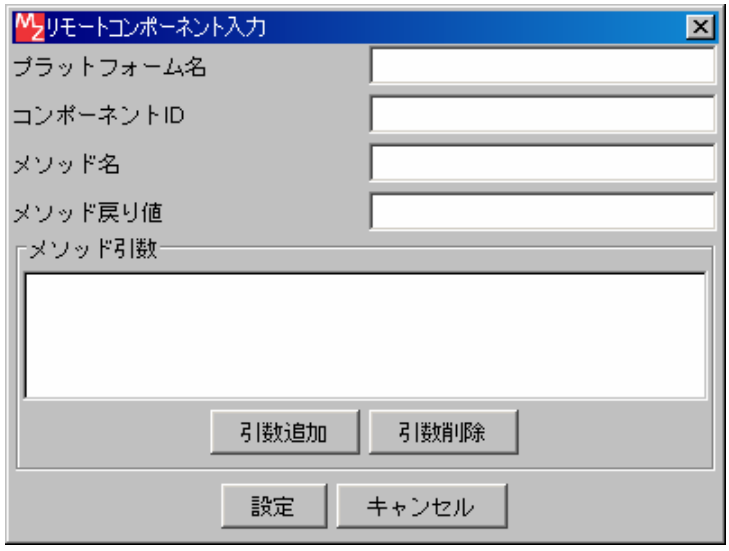


2.1. リモートコンポーネントの設定

接続コンポーネントを示す四角の上で右クリックをしてポップアップメニューを表示させる。

メニューの中から「リモートコンポーネント選択」－「データ連携」－「リモートコンポーネント入力」を選択し、クリックする。

以下のようなダイアログが表示されるので、各項目に呼び出し先の情報を記入する。



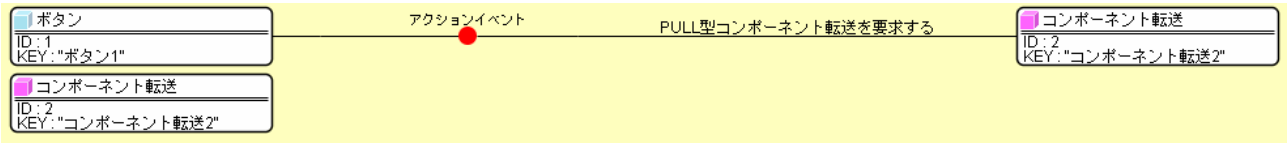
全ての項目について記入が必須である。(引数のないメソッドを呼び出す場合に限り、メソッド引数項目の記入を省略できる)

プラットフォーム名は呼び出すコンポーネントが配置されているプラットフォームの Platform.ini に記載されている名前であり、コンポーネント ID はその呼び出し先であるプラットフォーム上におけるコンポーネント ID である。

メソッド戻り値には、戻り値の型を記入する。(例 : int、java.lang.Long、java.lang.String、etc.)

3. コンポーネント転送機能

コンポーネント転送機能は、ビルダー上にコンポーネント転送コンポーネントを配置して、そのコンポーネント転送要求メソッドを起動することで使用する。



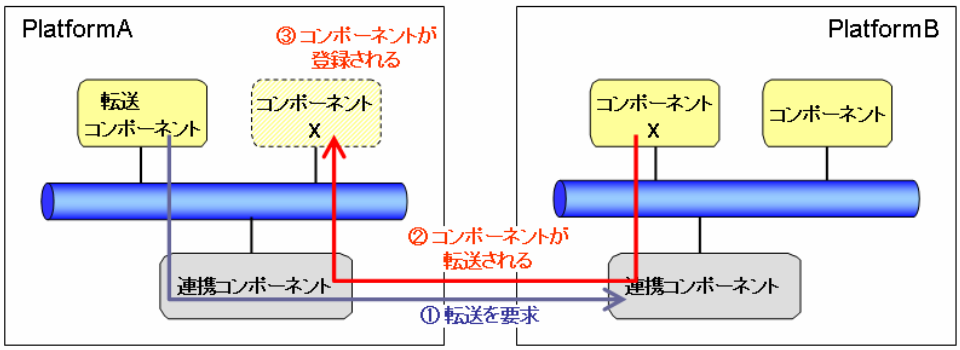
コンポーネント転送コンポーネントは、実体を持った処理コンポーネントであるため、ビルダー上での扱いは一般のコンポーネントと変わりはない。

3.1. PULL 型／PUSH 型

コンポーネント転送には、転送要求に対してどちら向きにコンポーネントが転送されるかによって、PULL 型と PUSH 型の 2 種類がある。

3.1.1. PULL 型コンポーネント転送

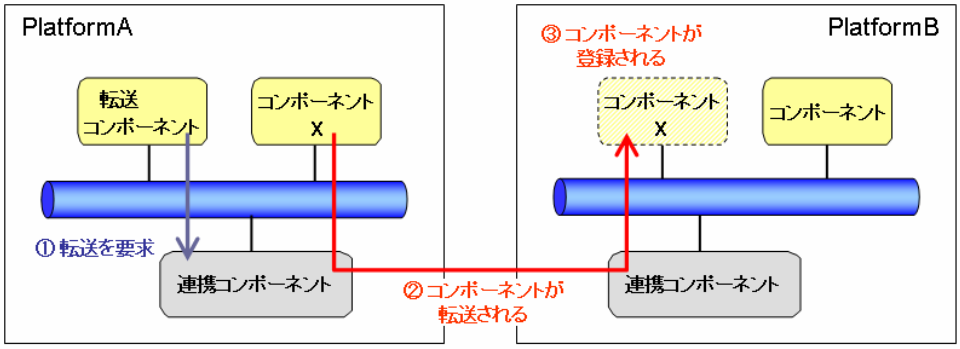
コンポーネント転送コンポーネントからの転送要求に、他プラットフォームからコンポーネントを取り寄せて、手元のプラットフォームに登録する転送を PULL 型コンポーネント転送という。



※ここでの「連携コンポーネント」とは、プラットフォーム側のデータ連携に関する一機能を表わす

3.1.2. PUSH 型コンポーネント転送

コンポーネント転送コンポーネントからの転送要求に、手元のプラットフォームにあるコンポーネントを送り出して、他プラットフォームに登録する転送を PUSH 型コンポーネント転送という。



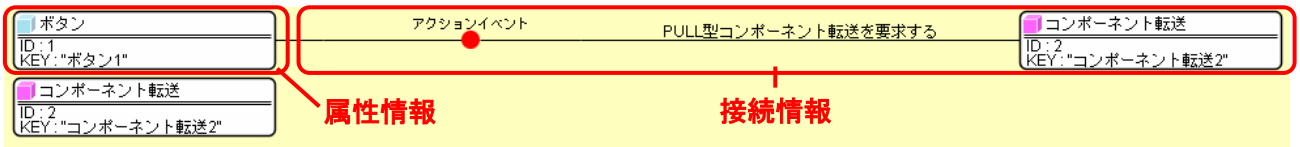
3.2. 様々なコンポーネント転送

コンポーネント転送コンポーネントには、様々な種類の転送要求メソッドが存在している。

これらは、転送されるコンポーネント情報の範囲や、転送に伴う各プラットフォームでの挙動の違いによって区別されている。

■ 転送されるコンポーネント情報の範囲

コンポーネント情報には、コンポーネントそのもののプロパティを示す「属性情報」と、そのコンポーネントから発生するイベントや起動メソッドの設定を示す「接続情報」がある。



■ 転送に伴う挙動のパターン

現在用意されている５種のコンポーネント転送について、転送に伴って起こる挙動をまとめる。

転送の名称	属性情報の転送	接続情報の転送	コンポーネント連携生成		転送対象の削除
			転送先	転送元	
コンポーネント転送	○	×	×	×	×
コンポーネントコピー転送	○	○	×	×	×
コンポーネント連携コピー転送	○	○	○	×	×
コンポーネント連携置換転送	○	○	○	○	×
コンポーネント連携移動転送	○	○	○	○	○

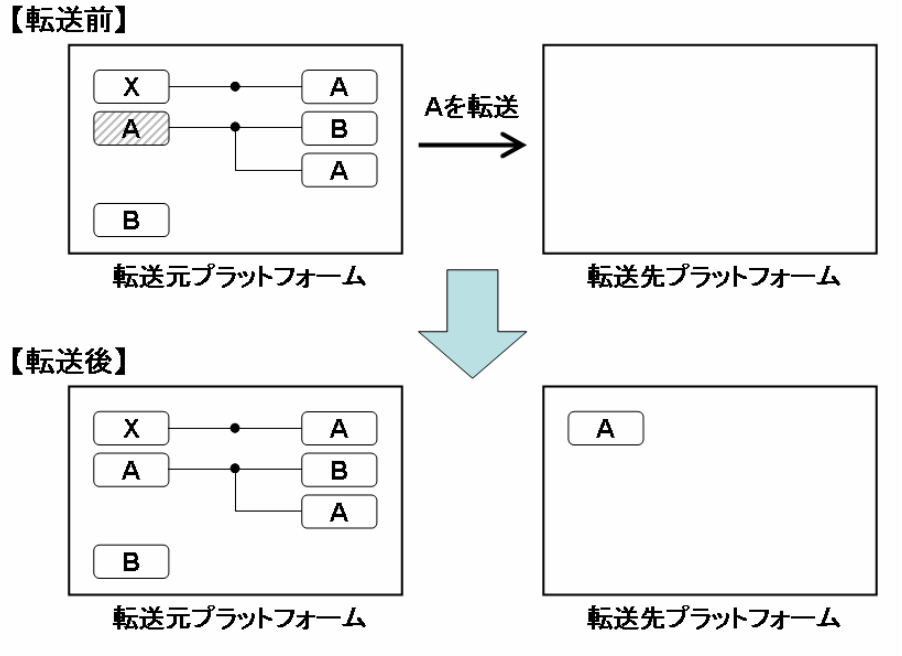
※転送先 ＝ 転送されてきたコンポーネントを受け取るプラットフォーム

※転送元 ＝ コンポーネントを送り出すプラットフォーム

各コンポーネント転送が実際にどのような振る舞いをするかを、以下に説明する。

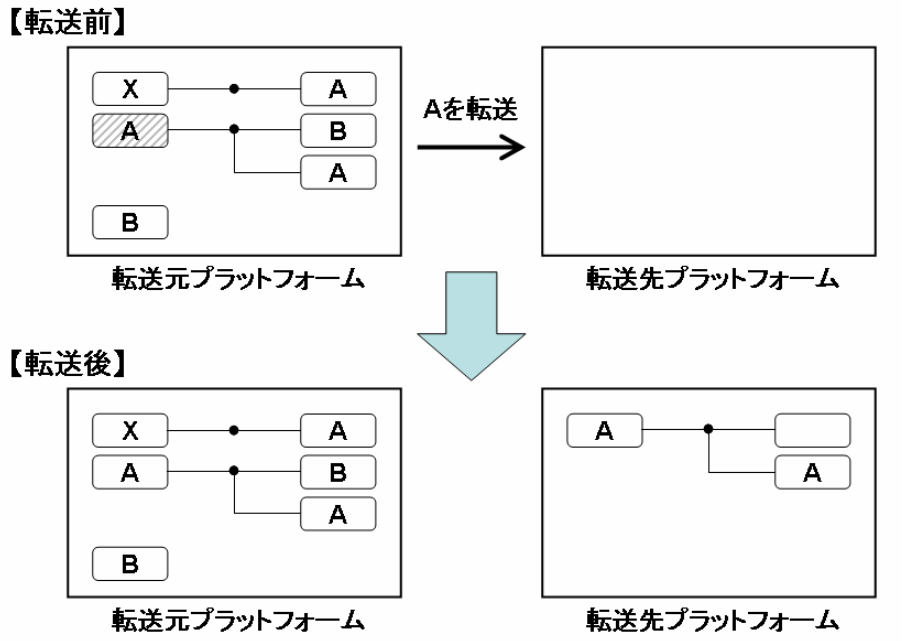
3.2.1. コンポーネント転送

最も基本的な転送で、コンポーネントの属性情報のみが転送される。



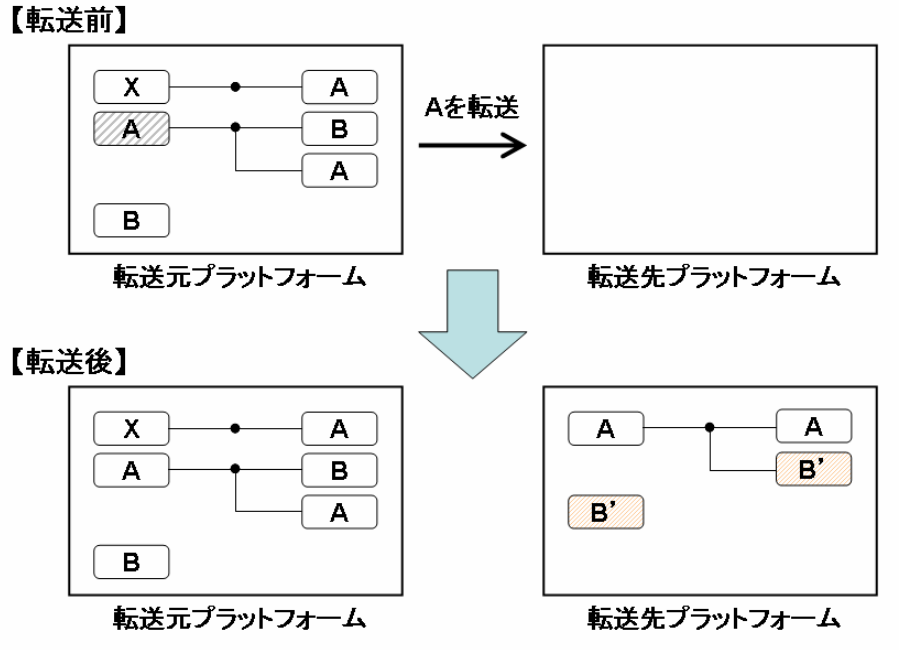
3.2.2. コンポーネントコピー転送

コンポーネントの接続情報も転送される。転送先プラットフォームにおいて、接続コンポーネントが転送対象ではない起動メソッドが空白に置き換わる。



3.2.3. コンポーネント連携コピー転送

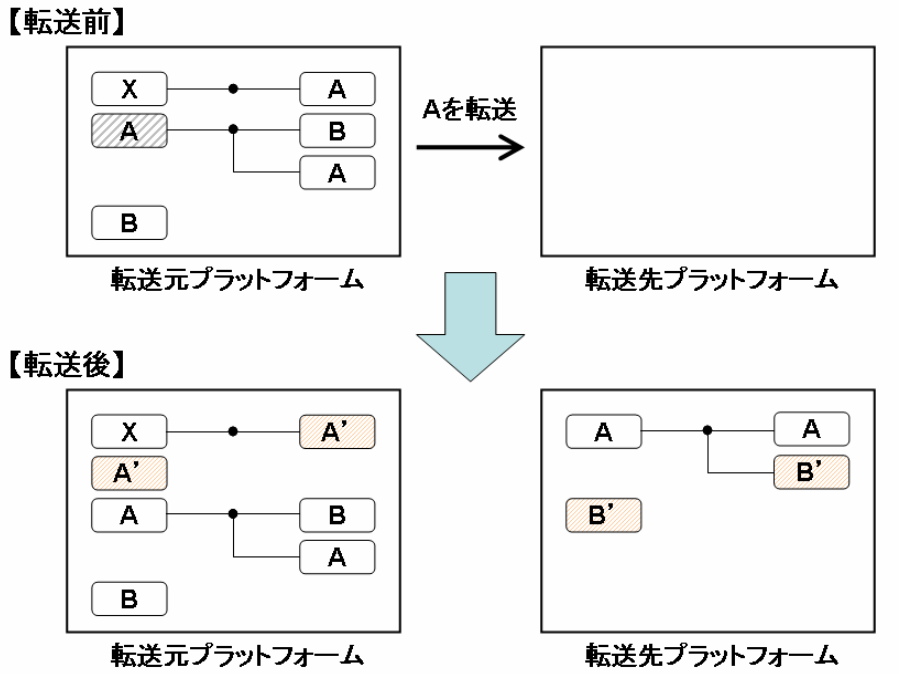
コンポーネントの接続情報も転送される。転送先プラットフォームにおいて、接続コンポーネントが転送対象ではない起動メソッドが、転送元プラットフォームへの連携に置き換わる。



3.2.4. コンポーネント連携置換転送

コンポーネントの接続情報も転送される。転送先プラットフォームにおいて、接続コンポーネントが転送対象ではない起動メソッドが、転送元プラットフォームへの連携に置き換わる。

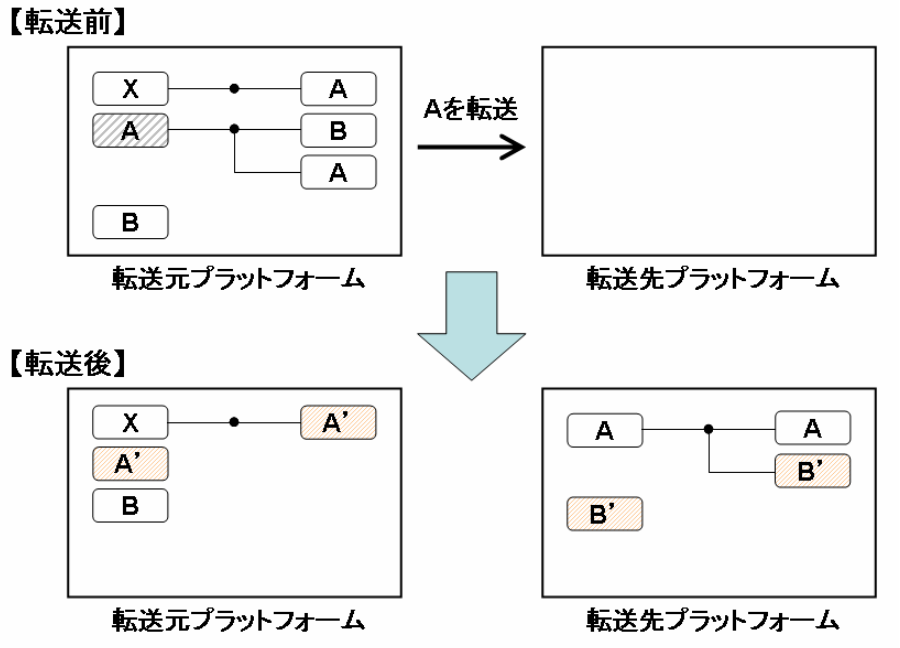
また、転送元プラットフォームにおいて、接続コンポーネントが転送対象である起動メソッドが転送先プラットフォームへの連携に置き換わる。



3.2.5. コンポーネント連携移動転送

コンポーネントの接続情報も転送される。転送先プラットフォームにおいて、接続コンポーネントが転送対象ではない起動メソッドが、転送元プラットフォームへの連携に置き換わる。

また、転送元プラットフォームにおいて、接続コンポーネントが転送対象である起動メソッドが転送先プラットフォームへの連携に置き換わり、転送対象そのものが削除される。



以上

A. 【付録】データ連携（旧バージョン）のサーバ環境設定

A.1. サーバ環境の準備

A.1.1. Sun Java System Application Server のダウンロード

以下の URL から Sun Java System Application Server（Platform Edition 7）をダウンロードし、インストールする。（ダウンロードには無料のアカウント登録が必要）

http://www.sun.com/download/products.xml?id=445bec68

インストールには、以下のサイトなどを参考にするとよい。

http://www.itmedia.co.jp/enterprise/0309/12/epn01_2.html#1

A.1.2. 事前設定（レジストリサーバ）

以下の項目が書かれた設定ファイル(registryserver.ini)を、以下の指定ディレクトリに置く。

（Windows の場合） C:¥registryserver.ini

（Linux の場合） /usr/etc/registryserver.ini

```
RegistryServerRoot=C:/RegistryServer/
AttachmentThreshold=0
```

パラメータ名	内容	備考
RegistryServerRoot	出力ファイルのディレクトリパス名	任意ディレクトリ
AttachmentThreshold	SOAP 通信において情報を添付にする為のしきい値	通常 0 に設定

※ RegistryServerRoot で指定するディレクトリは、任意だが必ず存在していなければならない。もしデフォルトでは存在しないディレクトリを記述した場合には、予め作成しておくこと。

A.1.3. 事前設定（ブローカ）

以下の項目が書かれた設定ファイル(broker.ini)を、以下の指定ディレクトリに置く。

（Windows の場合） C:¥broker.ini

（Linux の場合） /usr/etc/broker.ini

ブローカ名を“broker01”、ブローカ稼働コンピュータ名を“brokerpc”とした記述例。

```
BrokerName=broker01
BrokerAddress=brokerpc
BrokerSOAPPort=80
RegistryserverAddress=brokerpc:80
CodeBase=http://brokerpc:80/broker/rmi/
AccessLogFilePath=C:/BrokerLogs/BrokerAccess
DebugLogFilePath=C:/BrokerLogs/BrokerDebug
RetryMax=3
SleepTime=5000
```

パラメータ名	内容	備考
BrokerName	ブローカ名	任意の名称
BrokerAddress	ブローカアドレス	コンピュータ名のみ指定する
BrokerSOAPPort	ポート番号	基本は 80
RegistryServerAddress	レジストリサーバのアドレス	ポート番号まで含めて指定する （SOAS の場合、ブローカと同じ）
CodeBase	コードベース	
AccessLogFilePath	アクセスログのファイルパス名	ファイル生成時に曜日と拡張子が付け足されるので、名前どまりで指定する
DebugLogFilePath	デバッグログのファイルパス名	
RetryMax	最大リトライ回数	任意
SleepTime	通信失敗時、リトライに移るまでの待ち時間	ミリ秒で指定する。任意

※ AccessLogFilePath、及び DebugLogFilePath で指定するパスのディレクトリは、
必ず存在していなければならない。
もしデフォルトでは存在しないディレクトリを記述した場合には、予め作成しておくこと。

次に、以下のファイルに枠内の太字で示した修正を加える。

修正を加えるファイル：

（Application Serve のインストールフォルダ）/domains/domain1/server1/config/server.policy

```
//Basic set of required permissions granted to all remaining code
grant {
.....
...

//accept、resolveを追加する
 permission java.net.SocketPermission
      "*", "connect,accept,resolve";

//deleteを追加する
 permission java.io.FilePermission
      "<<ALL FILES>>", "read,write,delete";

// 設定そのものを追加する
 permission java.util.PropertyPermission
            "java.rmi.server.useCodeBaseOnly", "write";
 permission java.util.PropertyPermission
            "java.rmi.server.codebase", "write";
};
```

※各Permissionは一行で書く（改行しないこと）

A.2. 管理サーバの起動と終了

SOAS は、管理機能を提供する管理サーバと、Web アプリケーションを配備するアプリケーションサーバインスタンスから成り立っているので、まず管理サーバを起動する。

（Windows の場合）

【起動】

スタートメニューの「プログラム」→「Sun Microsystems」→「Sun ONE Application Server 7」から、「Start Application Server」を選択する。

【終了】

スタートメニューの「プログラム」→「Sun Microsystems」→「Sun ONE Application Server 7」から、「Stop Application Server」を選択する。

（Linux の場合）

【起動】

以下のコマンドを実行する。

$ asadmin start-appserv

【終了】

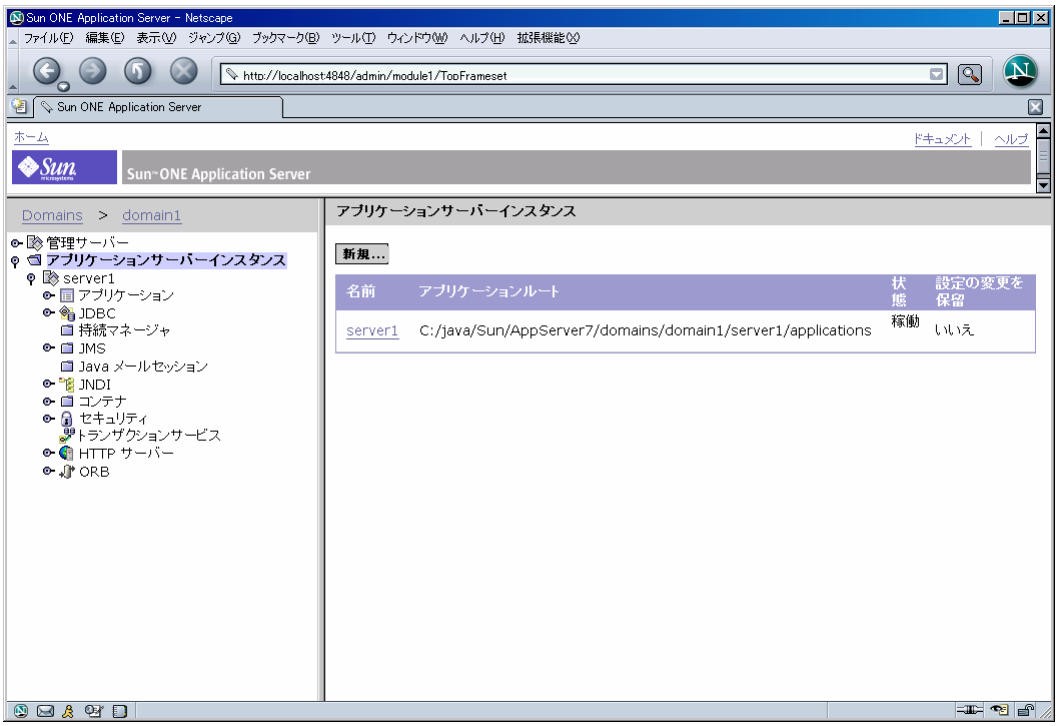
以下のコマンドを実行する。

$ asadmin stop-appserv

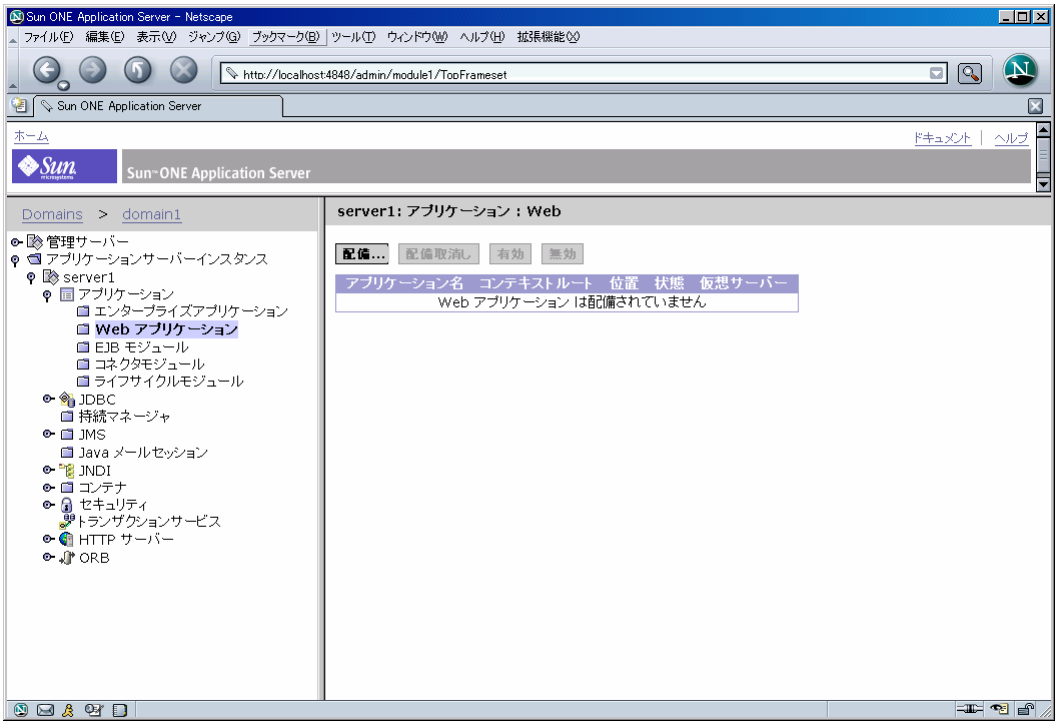
※これらのコマンドは、Wndows のコマンドプロンプトでも使用可能である

A.3. Web アプリケーションの配備

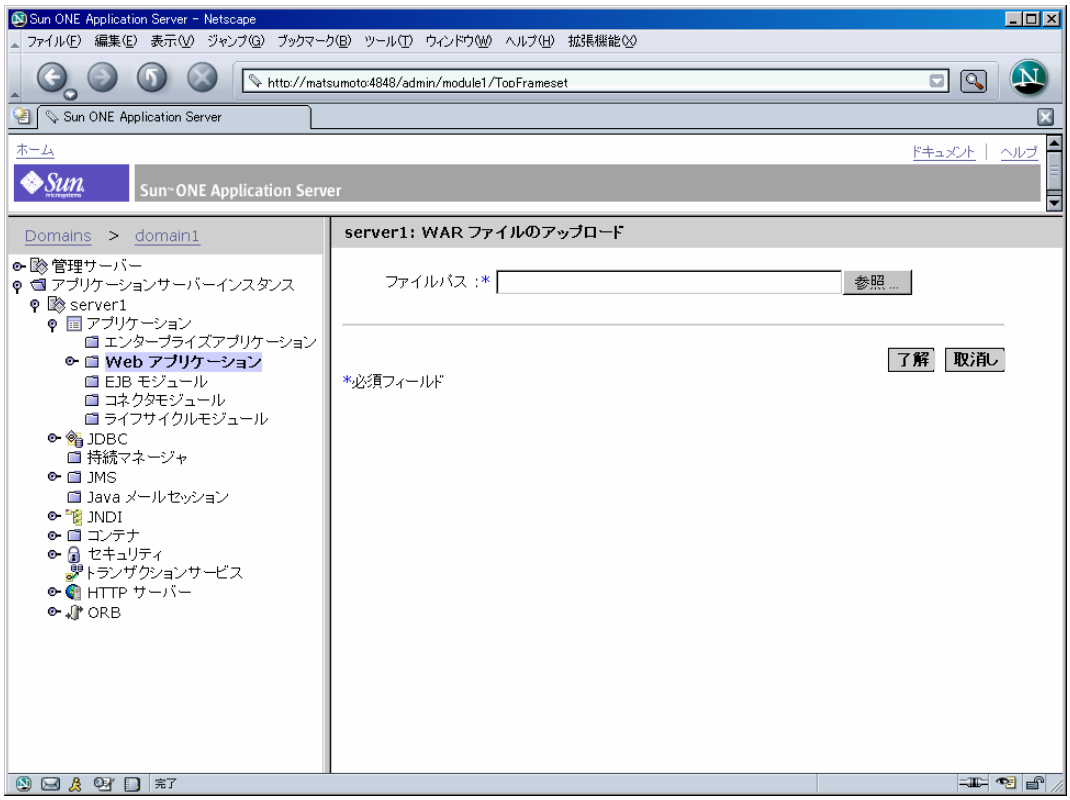
管理サーバを稼動させている状態で、ブラウザから http://localhost:4848/ へアクセスし、インストール時に設定したユーザ名とパスワードを入力すると、以下のような管理画面が表示される。



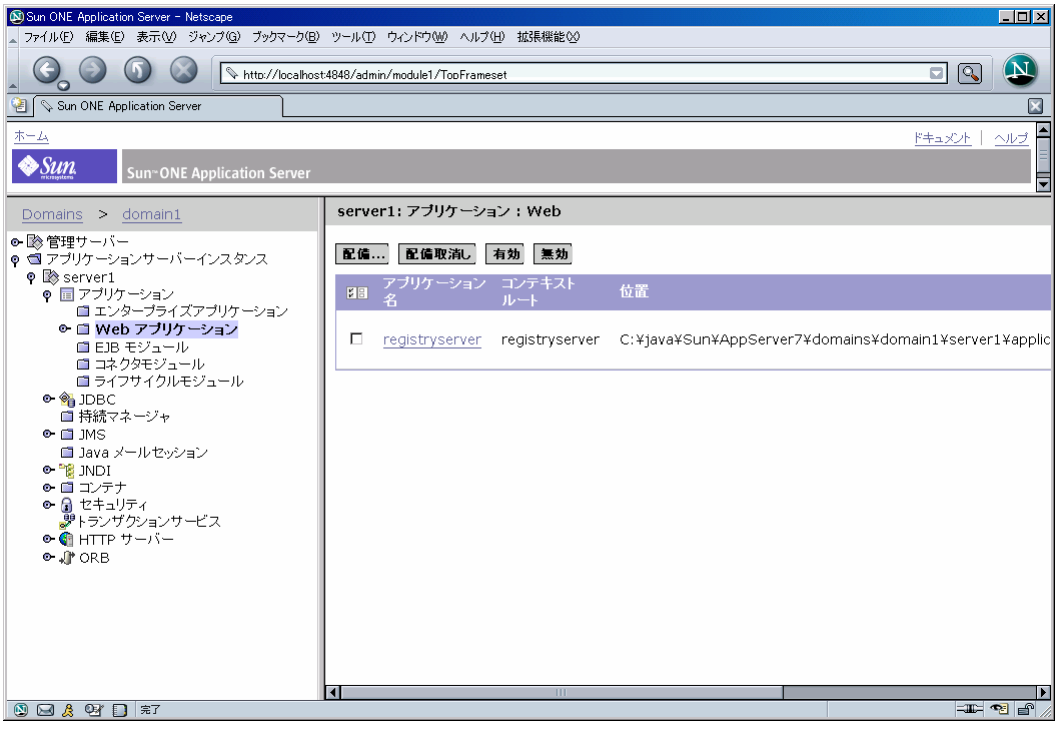
画面の左にあるツリーを「アプリケーションサーバーインスタンス→server1→アプリケーション」とたどり、「Web アプリケーション」というノードを選択する。



右側の画面に現れている「配備」ボタンを押すと、以下の画面のようにファイルを指定する画面になるので、リリースモジュール内の registryserver.war ファイルを選択する。



正しく配備されると最初の画面に戻り、配備された Web アプリケーションが表形式で表示される。



続いて、同じ手順でリリースモジュール内の broker.war ファイルを配備する。

注意しなければならないのは、**アプリケーションサーバーインスタンスが既に稼動している場合、Web アプリケーションは配備と同時に起動されるので、レジストリサーバより先にブローカを配備するとエラーが発生してしまう**ことだ。

必ず、レジストリサーバを先に配備するか、もしくは後述の方法でアプリケーションサーバインスタンスを一旦停止させてから両方の配備を行い、ブローカを無効にした状態でアプリケーションサーバインスタンスを再起動する。

A.4. Web アプリケーションの起動と終了

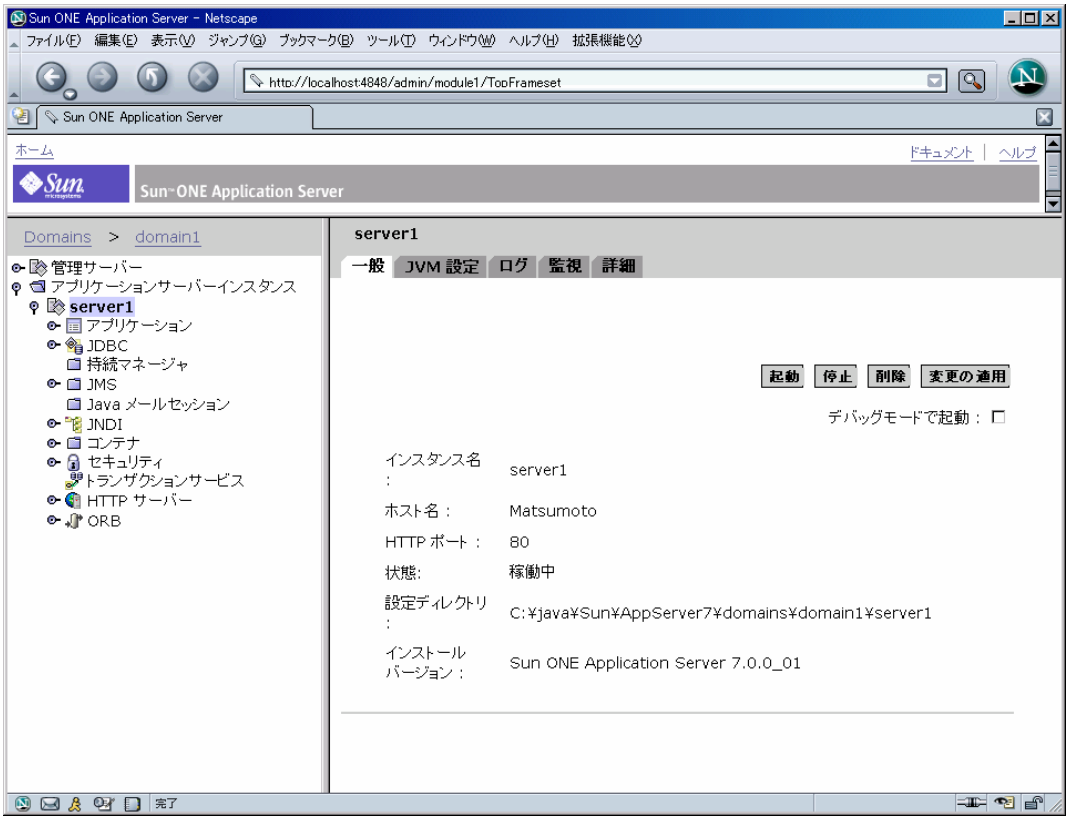
A.4.1. アプリケーションサーバーインスタンスの起動と終了

アプリケーションサーバーインスタンスは初回のみ管理サーバと同時に起動するが、その後は独立に停止させたり再起動させたりすることができる。

また、Web アプリケーションの有効・無効設定を用いて、一つのアプリケーションサーバーインスタンスに配備した複数の Web アプリケーションを、選択的に起動・終了させることもできる。

■アプリケーションサーバーインスタンス自体の起動・停止

左側のツリーにおいて「アプリケーションサーバーインスタンス→server1」のノードを選択すると、以下のような管理画面が表示される。



この画面にある「起動」「停止」ボタンを押下すると、アプリケーションサーバーインスタンスが起動、または停止する。つまり、このアプリケーションサーバーインスタンスに配備されている Web アプリケーションのうち、有効状態である（後述）もの全てが同時に起動または停止する。

■Web アプリケーションの選択的な起動・停止

アプリケーションサーバーインスタンスに配備される Web アプリケーションには、それぞれに「有効」「無効」という状態を設定することができる。

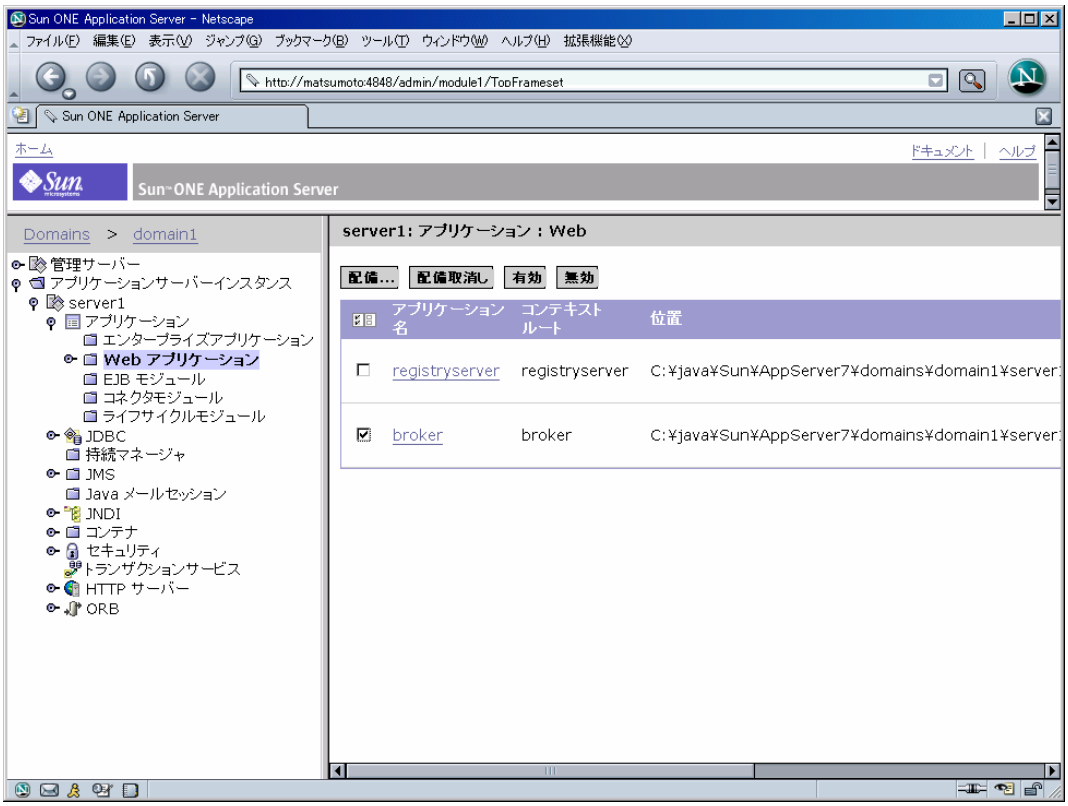
無効に設定された Web アプリケーションは、アプリケーションサーバーインスタンスが起動された場合でも、起動されない。稼動中の Web アプリケーションの設定を有効から無効に変更すれば、アプリケーションサーバーインスタンスそのものを停止することなく、その Web アプリケーションだけを停止させることができる。

このように、一つのアプリケーションサーバーインスタンスに配備された複数の Web アプリケーションは選択的に起動・停止することができる。

配備した直後の Web アプリケーションのデフォルトの状態は、有効状態である。

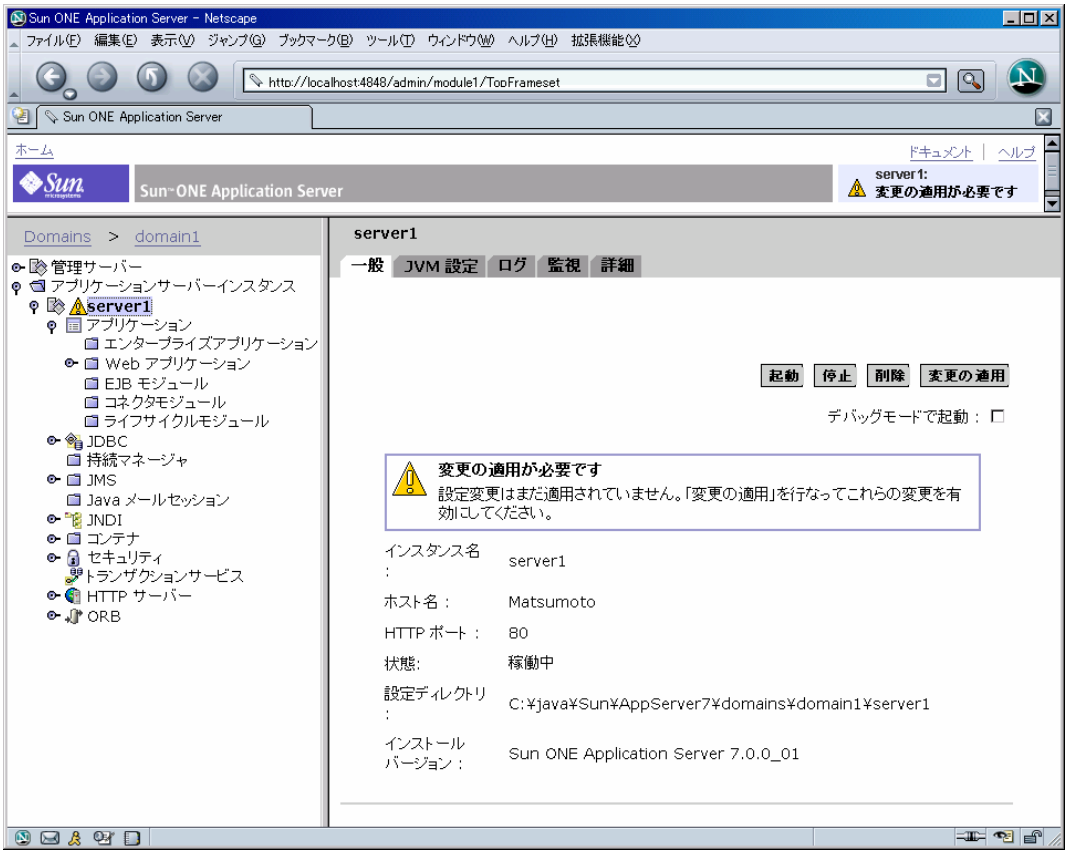
有効・無効状態の設定は、ツリーの「アプリケーションサーバーインスタンス→server1→アプリケーション→Web アプリケーション」ノードを選択して表示される Web アプリケーションの管理画面から行うことができる。

まず、配備された Web アプリケーションの中から、状態を変更させたい Web アプリケーションのチェックボックスにチェックを入れて「有効」または「無効」ボタンを押下する。



すると、上部フレームとアプリケーションサーバーインスタンスのノードに、変更された設定の適用を促す黄色のマークが現れるので、アプリケーションサーバーインスタンスの管理画面へ移る。

「変更の適用」ボタンを押下すると、変更が適用される。



■レジストリサーバの起動と終了

レジストリサーバは、アプリケーションサーバーインスタンスと共に起動・終了させて問題ない。

■ブローカの起動と終了

・ 起動

ブローカを起動する場合には、ブローカ情報の登録の関係上、どうしても事前にレジストリサーバが起動している必要がある。そこで、配備の順序や選択的な起動などを利用して、ブローカは必ずレジストリサーバの後に起動動作に入るように留意する。

・ 終了

ブローカを終了させる場合には、アプリケーションサーバーインスタンス自体を停止するのではなく、まずブローカだけを選択的に停止させる（無効状態へ変更させる）ようにすると良い。これは、次回アプリケーションサーバーインスタンスを起動させる時、レジストリサーバより先にブローカが起動しようとしてエラーになるのを防ぐ為である。

もしアプリケーションサーバーインスタンスごと停止させてしまった場合でも、管理サーバを終了させていなければ、ブローカの設定変更は行える。ただ、アプリケーションサーバインスタンスを停止させた後にブローカを無効にするのは忘れやすいので、ブローカ停止→アプリケーションサーバインスタンス停止という手順を守るようにすると良い。

B. 【付録】データ連携（旧バージョン）の操作手順

B.1. 初期設定（Platform.ini の編集）

配布された Platform.ini を、環境に合わせて変更する。

1) データ連携を使用するかのフラグ：UseDataCooperation を true（使用する）に変更する
2) 旧バージョンのデータ連携機能を使用しないかのフラグ：UseOnlyLightDataCooperation を false（使用する）に設定する
3) プラットフォーム名：PlatformName に、プラットフォーム名を設定する（所属するブローカ内で一意であること）
4) ブローカアドレス：BrokerAddress に、このプラットフォームが接続するブローカのアドレスを設定する

以下はブローカアドレスを“http://xxx.example.com:80/broker/”、プラットフォーム名を“Platform01”とした場合の Platform.ini の記述例である。

```
LogLevel=1
LookAndFeel=
RuntimeLocale=
ComponentListFile_ja=etc¥¥PlatformComponents_ja.ini
ComponentListFile_en=etc¥¥PlatformComponents_en.ini
ComponentListFile=etc¥¥PlatformComponents_en.ini
ComponentInformationFolder=components
BinaryDataAutoSave=true
UseDataCooperation=true
UseOnlyLightDataCooperation=false
BrokerAddress=http://xxx.example.com:80/broker/
PlatformName=Platform01
LocalhostAddress=
datamanagement.default_componentcooperation_policy=true
datamanagement.default_componentpulltransfer_policy=true
datamanagement.default_componentpushtransfer_policy=true
java.rmi.server.codebase=
java.security.policy=etc¥¥java.policy
brokermonitor.broker_access_log=
brokermonitor.broker_debug_log=
RMIPort=
CombinativeComponentsFolder=AP_DATA_COMB
```

※各プロパティについての詳細は、Readme.txt に記されているのでそちらを参照のこと。

B.2. コンポーネント連携

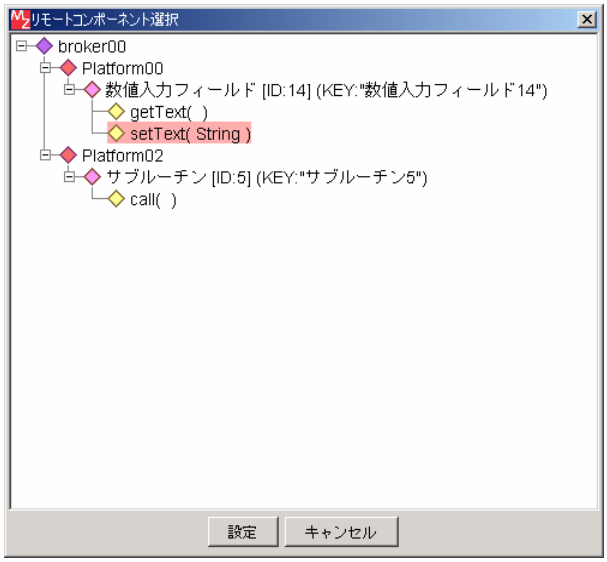
旧バージョンのコンポーネント連携では、リモートコンポーネントの設定方法として、レジストリサーバに登録されたコンポーネントの一覧からメソッドを選択する方法と、現バージョンと同じく必要項目をダイアログに手入力する方法と２種類用意されている。

B.2.1. リモートコンポーネント選択

接続コンポーネントを示す四角の上で右クリックをしてポップアップメニューを表示させる。

メニューの中から「リモートコンポーネント選択」－「旧データ連携」－「リモートコンポーネント一覧」を選択し、クリックする。

以下のような選択画面が表示されるので、呼び出したいメソッドを選択して「設定」ボタンを押す。

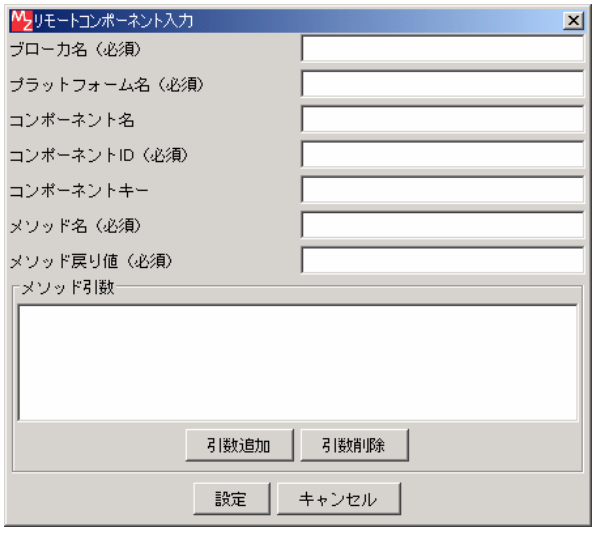


B.2.2. リモートコンポーネント入力

接続コンポーネントを示す四角の上で右クリックをしてポップアップメニューを表示させる。

メニューの中から「リモートコンポーネント選択」－「旧データ連携」－「リモートコンポーネント入力」を選択し、クリックする。

以下のようなダイアログが表示されるので、必要事項を入力する。



B.3. コンポーネント転送

コンポーネント転送に関しては、旧バージョンのコンポーネント転送コンポーネントをビルダーに配置するだけで、その他の操作は現バージョンや一般のコンポーネントと変わりはない。

以上