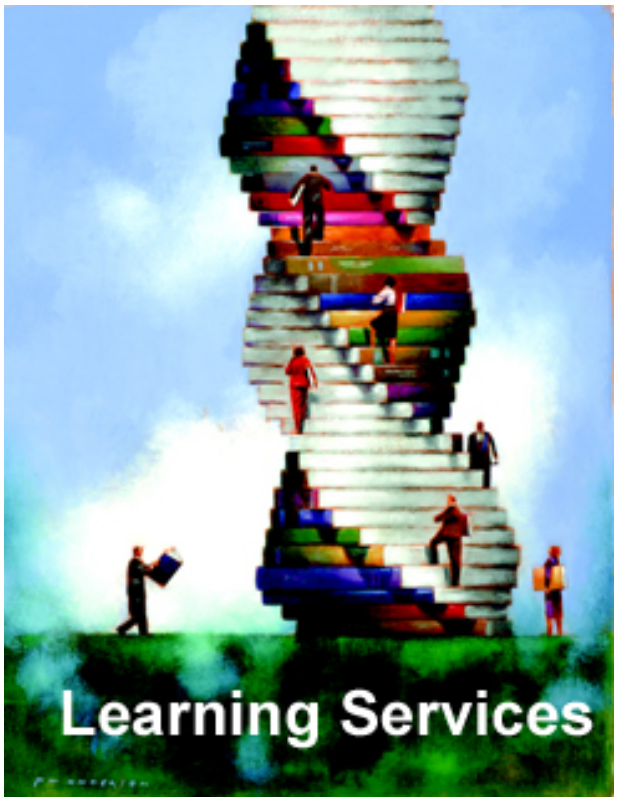


Caché Server Pages ガイド
（Caché Version 2015.1 ベース）
V1.0

2015 年 4 月

インターシステムズジャパン株式会社

目次

図表目次

1　CSP（Caché Server Pages）とは

CSP(Caché Server Pages)は、HTML ファイルの中に Caché ObjectScript やロジックを埋め込み、動的な Web ページを生成させるための技術です。
IIS や Apache といった Web サーバーを利用し、Web サービスが提供できます。

コンポーネントの構成は、以下の通りです。
CSP サーバーのホスト名や、CSP ファイルの物理パス、Caché のネームスペースなどの環境は、「アプリケーション」と呼ばれるリクエスト URL の一部を元に、設定できます。

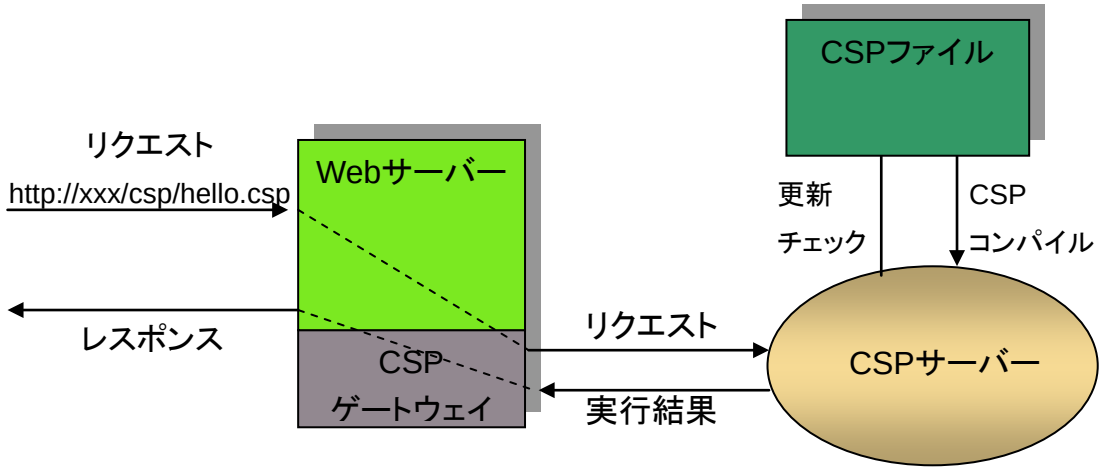


図 1 CSP の構成

- **Webサーバー**
  インターネットなど外部からのリクエストを受け付けるソフトウェアで、Apache、IISといった一般的なWebサーバーを利用します。
- **CSPゲートウェイ**
  CGI、ISAPIまたはNSAPIインターフェースでWebサーバーからリクエストを受け取り、CSPサーバーに転送します。
- **CSPファイル**
  CSPのスクリプトが記述されたファイル。Cachéと同じサーバー上に保管し、ファイルが更新されていれば、そのスクリプトを元に適宜コンパイルが行われます。ファイルの拡張子はcspです。
- **CSPサーバー**
  CSPゲートウェイからのリクエストにより、CSPファイルを元にコンパイルしたクラスを実行し、その実行結果をCSPゲートウェイに返します。

一般的に Web サーバーと Caché がインストールされた DB サーバーは別ホストとなりますが、今回は、同一サーバーに Web サーバーと Caché サーバーがインストールされていることを前提に説明します。

Web サーバーへの CSP ゲートウェイ インストレーション方法につきましては、ドキュメントの「Caché WEB 開発」にある「CSP ゲートウェイ構成ガイド」を、参照してください。
http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GCGI

（Web上のドキュメントは こちらから ご覧いただけます。）

2　CSP のアプリケーションに必要な設定

CSP のアプリケーションを構築する際には以下の設定が必要です。

- **Webサーバーの設定**
  仮想ディレクトリを設定し、スクリプトの実行権を設定します。
- **CSPゲートウェイの設定**
  リクエストされたURLのアプリケーション名から、どのサーバーのCaché環境に接続するかを設定します。
- **管理ポータルの設定**
  URLからどのネームスペースを使用するかを設定します。
  管理ポータル→セキュリティ→アプリケーション→ウェブ・アプリケーション

2.1　CSP ゲートウェイの設定

CSP ゲートウェイでは、ウェブ・アプリケーションごとにアクセスするサーバーや Caché ポート番号を設定します。
以下の説明では、Caché インストール時に自動的に設定されるプライベート Web サーバーを利用した CSP ゲートウェイ管理画面の設定を説明しています。（プライベート Web サーバーは Apache を利用しポート番号は 57772 を割り当てています。）

以下 URL は、CSP ゲートウェイ管理画面を起動する時の URL です。（全 Web サーバーで共通の URL です。）

エラー！ ハイパーリンクの参照に誤りがあります。

プライベート Web サーバーを利用した CSP ゲートウェイ管理画面の起動は、管理ポータル→システム管理→構成→CSP ゲートウェイ管理　から起動します。

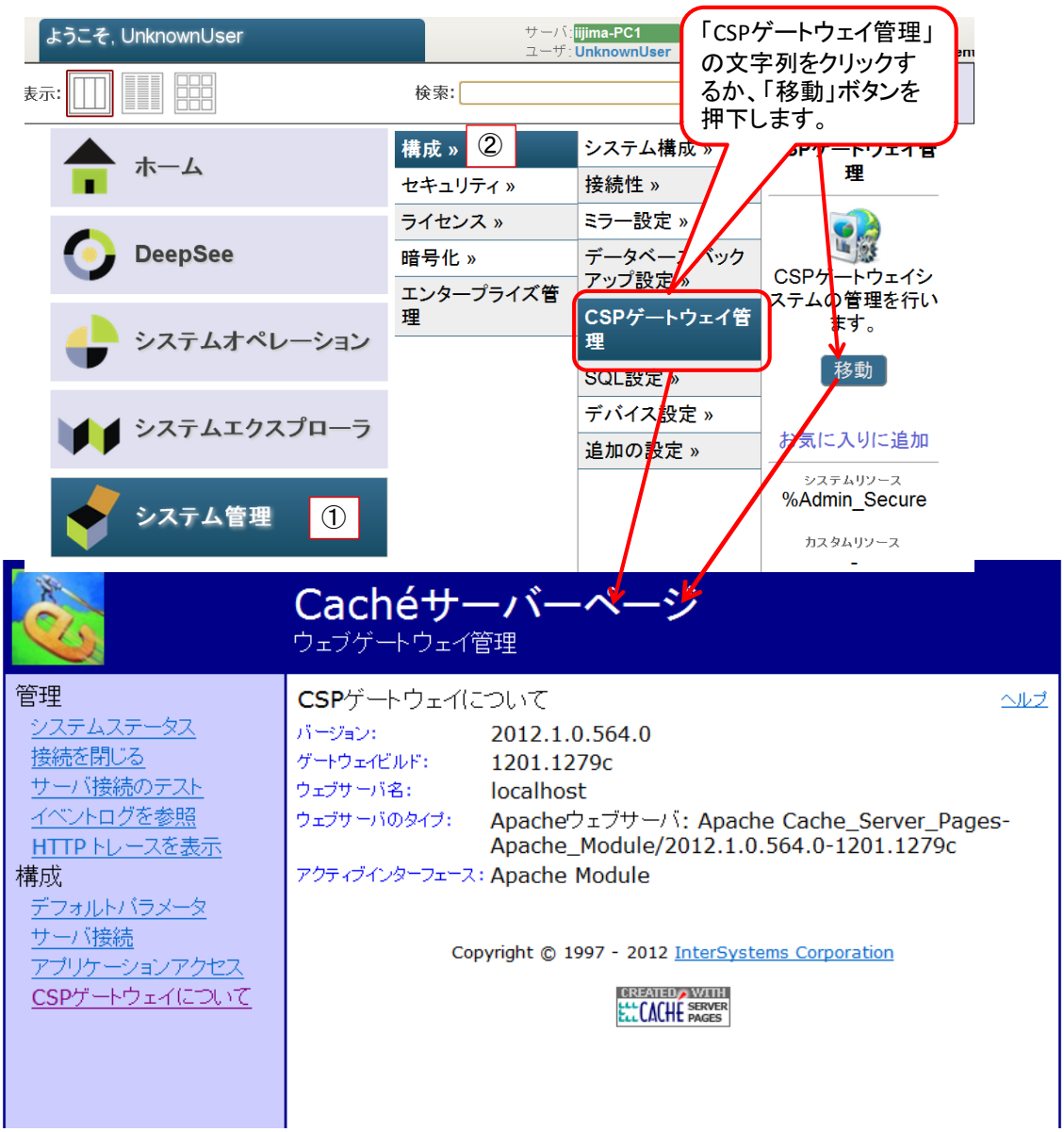


図 2 プライベート Web サーバーを利用した CSP ゲートウェイ管理画面の起動

2.2 CSP 実行時の URL について

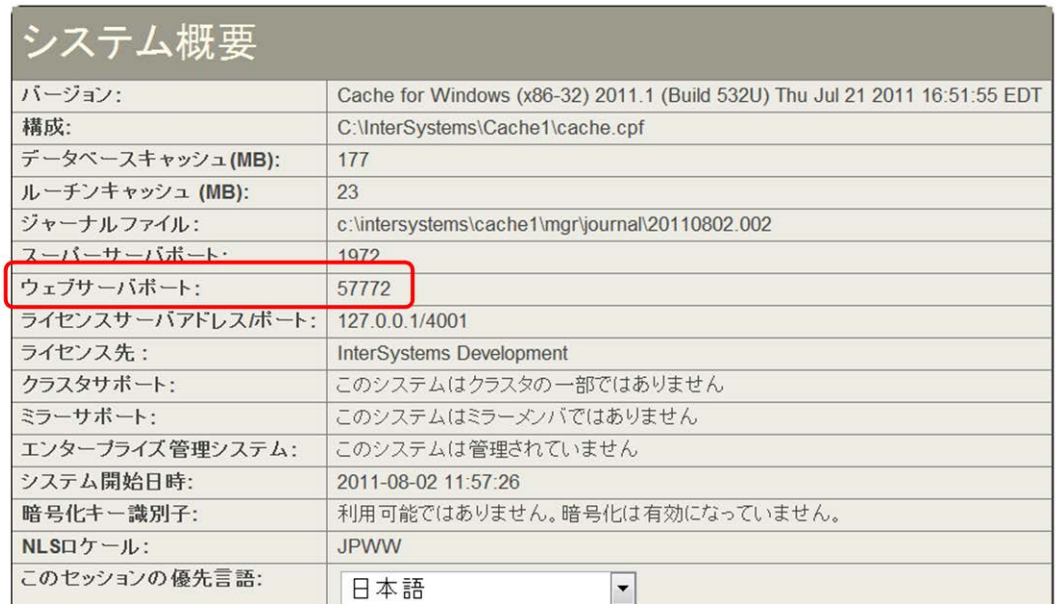
今回は、作成した CSP ページの実行には、インストール時自動でインストールされる Apache のプライベート Web サーバーを利用します。
ページを実行する時の URL は以下の通りです。

http://localhost:57772/csp/user/****.csp

localhost:57772	プライベート Web サーバー（Apache）のアドレスです。 Web サーバーポートの確認： 図 4 Web サーバーポート 確認 参照
/csp/user	ウェブ・アプリケーションのパス名 ネームスペース作成時、ネームスペース名に合わせてウェブ・アプリケーション名が作成されます。（/csp/＜ネームスペース名＞） /csp/user は、USER ネームスペースの利用を示す、ウェブ・アプリケーション名です。 CSP ファイルの物理パスは、/csp/user の場合 ＜インストールディレクトリ＞¥csp¥user が設定されています。
****.csp	CSP ファイル名を指定します。

図 3 CSP ページ実行時の URL

Web サーバーポートは、管理ポータルの「概要」メニューにある「ウェブサーバーポート」から確認できます。



システム概要

バージョン:	Cache for Windows (x86-32) 2011.1 (Build 532U) Thu Jul 21 2011 16:51:55 EDT
構成:	C:\InterSystems\Cache1\cache.cpf
データベースキャッシュ(MB):	177
ルーチンキャッシュ (MB):	23
ジャーナルファイル:	c:\intersystems\cache1\mgr\journal\20110802.002
スーパーサーバポート:	1972
ウェブサーバポート:	57772
ライセンスサーバアドレスポート:	127.0.0.1/4001
ライセンス先 :	InterSystems Development
クラスタサポート:	このシステムはクラスタの一部ではありません
ミラーサポート:	このシステムはミラーメンバではありません
エンタープライズ管理システム:	このシステムは管理されていません
システム開始日時:	2011-08-02 11:57:26
暗号化キー識別子:	利用可能ではありません。暗号化は有効になっていません。
NLSロケール:	JPWW
このセッションの優先言語:	日本語

図 4 Web サーバーポート 確認画面

今回利用する、ウェブ・アプリケーション「/csp/user」は、USER ネームスペース用に用意されたウェブ・アプリケーション名で、Caché インストール時に用意されます。
CSP ファイルは、ウェブ・アプリケーションパス配下に配置する必要があります。
スタジオで CSP ページを作成した場合は、CSP ファイル保存時、アプリケーションパス毎にフォルダが表示されますので、該当フォルダ以下に保存します。

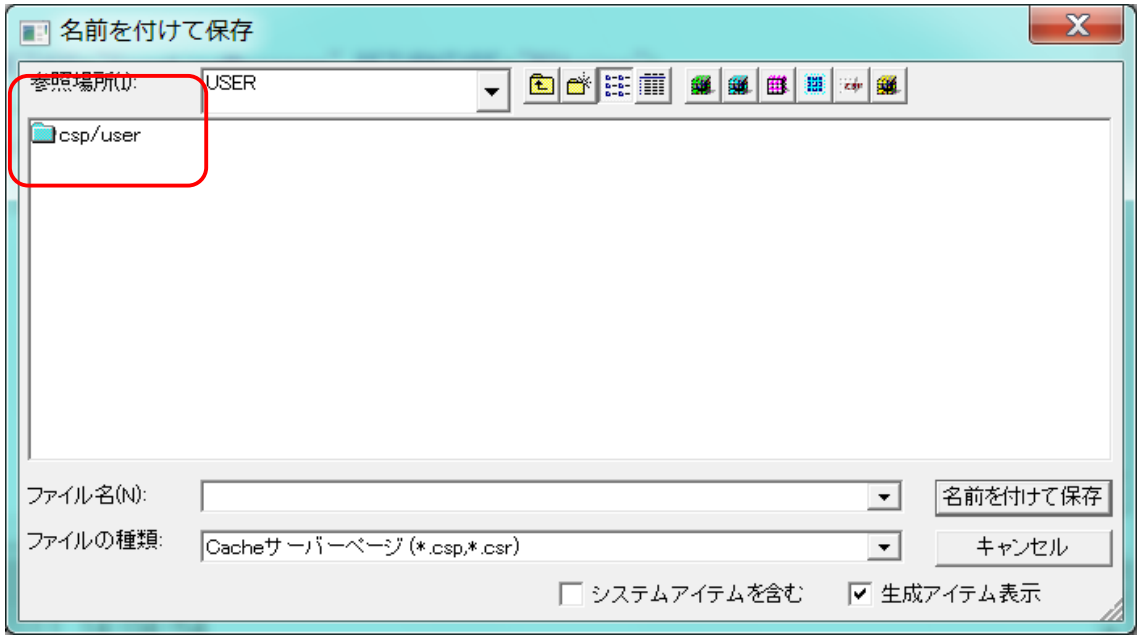


図 5 スタジオでの CSP ファイルの保存（アプリケーションパス用フォルダ）

別のネームスペースを作成し（例えば、TEST ネームスペース）、その環境に CSP ページを作成する場合、ネームスペース作成時に、**「このネームスペースにデフォルトのウェブアプリケーションを作成」**に、チェックがある状態であれば、**/csp/test** が自動的に作成されます。

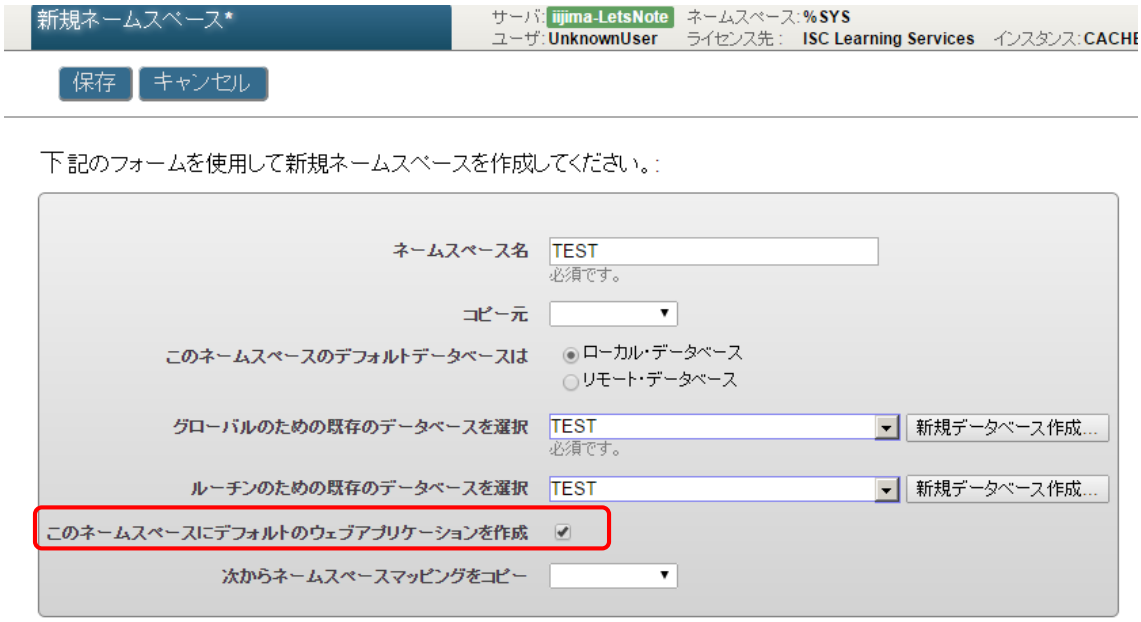


図 6 ネームスペース作成時の CSP 用オプション

ネームスペース名に依存しないパス名や、/csp 以外のパス名を利用したい場合は、新規でウェブ・アプリケーション名を作成することもできます。
本ガイドでは、簡単に CSP 画面の作成を試すため、Caché インストール時用意される、ウェブ・アプリケーションパス＝「/csp/user」を利用します。

なお、Caché インストール時、IIS や Apache がインストール済の状態であると、自動で、/csp を仮想ディレクトリとして Web サーバーに設定します。
また、CSP ゲートウェイ管理では、/csp が URL に指定されると、ローカル（127.0.0.1）の Caché に接続するように設定されています。

CSP ゲートウェイ管理画面での　/csp　の設定は以下の図のとおりです。
（アプリケーションアクセス→　/csp　を選択→　アプリケーション編集を選択→　実行ボタン押下）

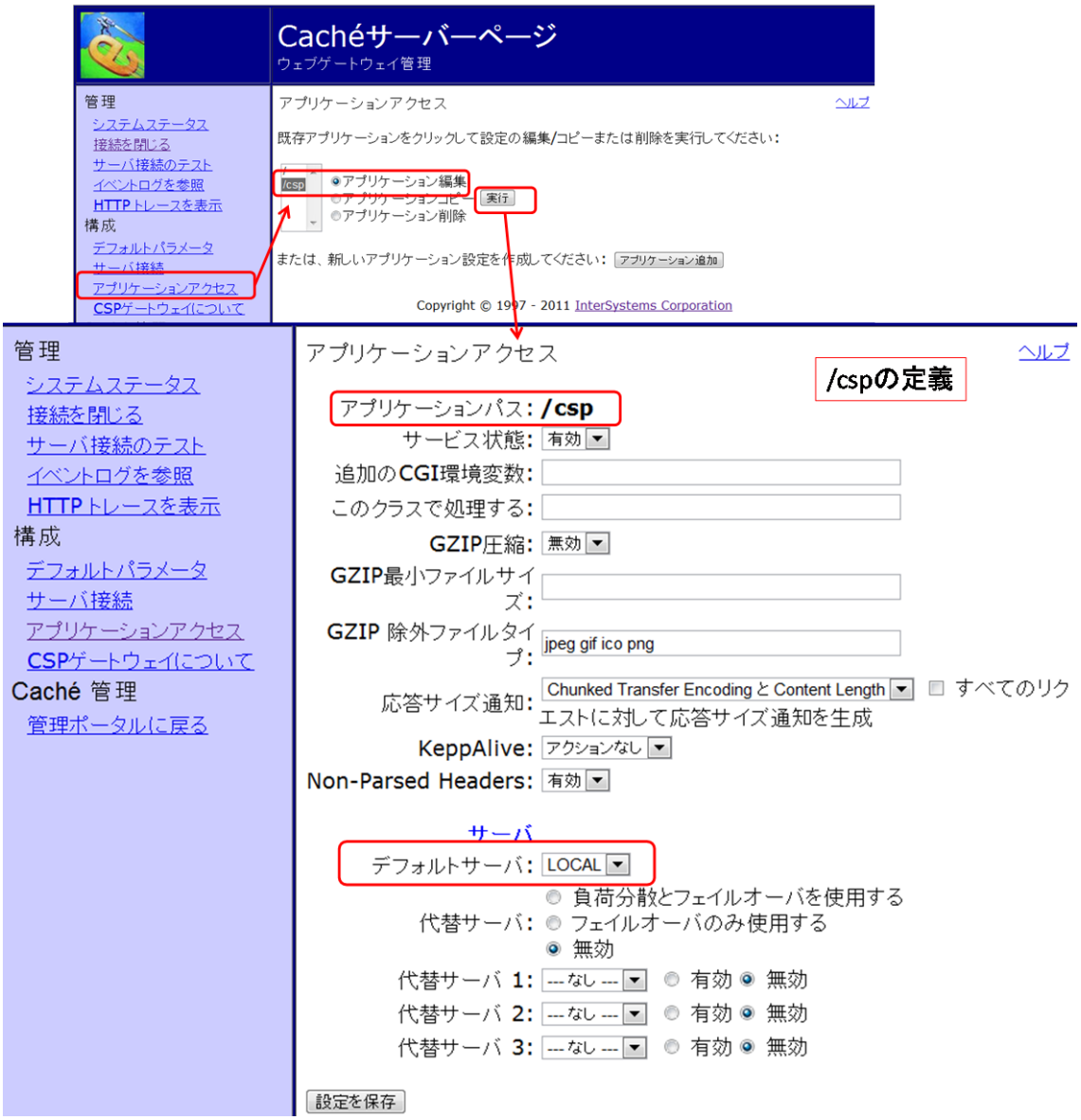


図　7 CSP ゲートウェイ管理画面：/csp の定義

/csp に定義されていた サーバ:LOCAL の設定は以下の図のとおりです。
（サーバ接続→ LOCAL を選択→ サーバ編集を選択→ 実行ボタン押下）

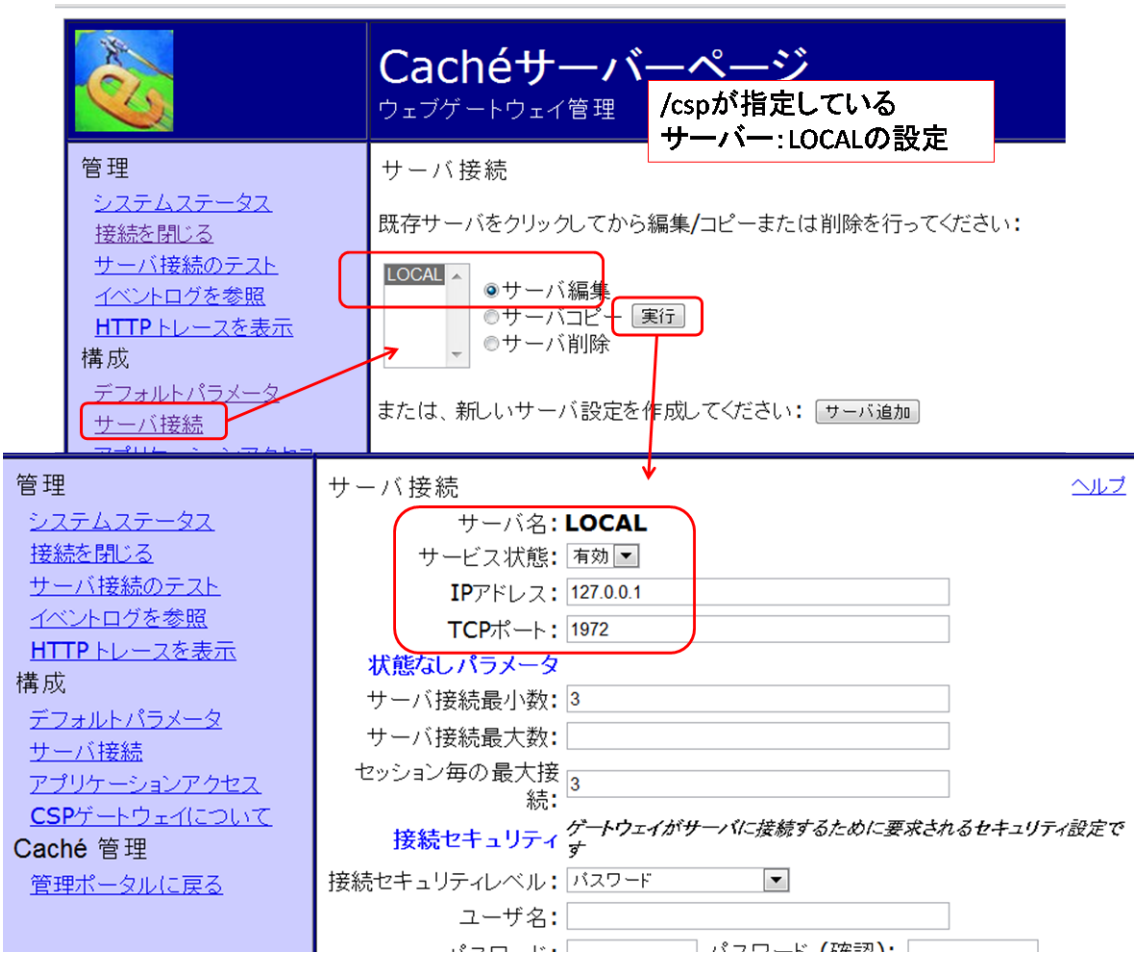


図 8 CSP ゲートウェイの設定:サーバ「LOCAL」の設定

続いて、DB サーバー側の設定を説明します。
今回利用する、「/csp/user」 は、管理ポータル→セキュリティ→アプリケーション→ウェブ・アプリケーション　のメニューから設定を確認できます。

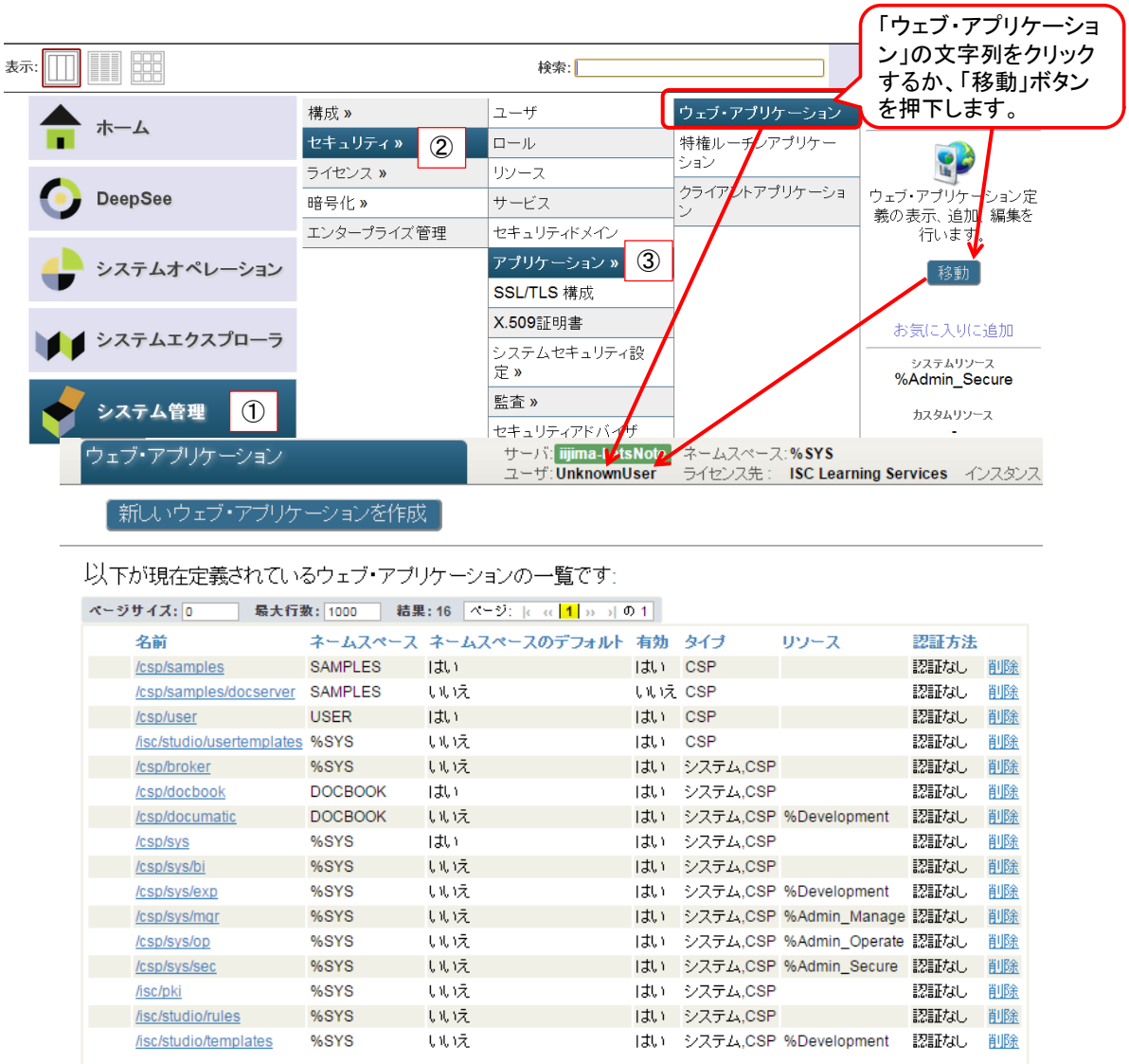


図 9 管理ポータル：ウェブ・アプリケーション

/csp/user の設定には、利用するネームスペース＝USER が割り当てられています。また、CSP ファイルの格納先物理パスも設定されています。（Caché インストールディレクトリ¥csp¥user）スタジオ以外のエディタで、CSP ファイルを作成した場合は、「CSP ファイル物理パス」で定義されたディレクトリに CSP ファイルを配置する必要があります。（図解は次ページをご参照ください。）

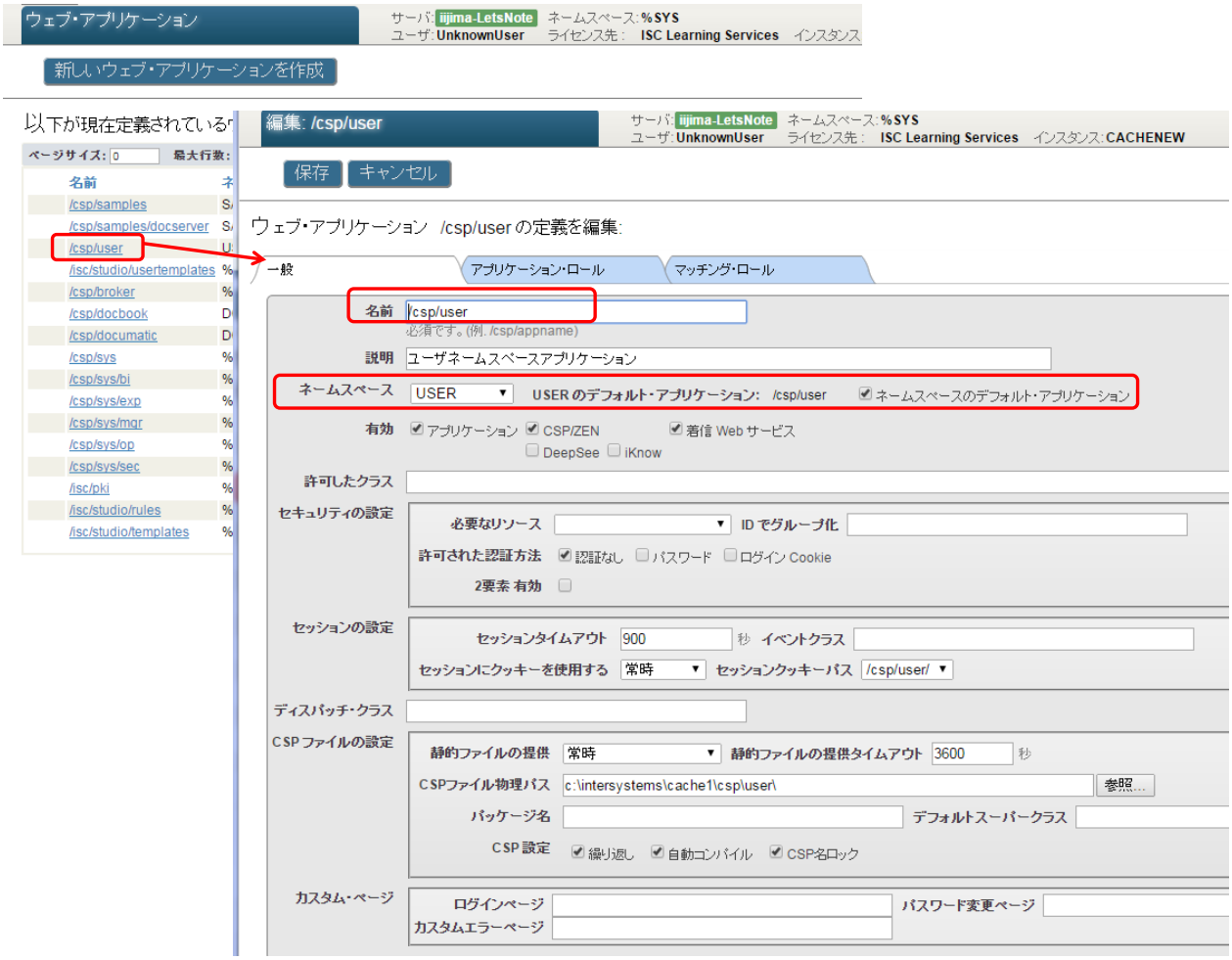


図 10 管理ポータル: /csp/user の設定

3　まずは簡単な CSP ページをつくってみよう

CSP ページは CSP ファイルを作成、編集することで作成できます。エディタは Caché スタジオで最初から作成してもかまいませんが、Web オーサリングツールを利用し、画面デザインを作成してから Caché スタジオにコピーする方法がいいでしょう。

3.1　固定ページの作成

固定のページを作成する場合は HTML をそのまま記述した CSP ファイルを作成するだけです。たとえば以下のようなファイルを作成し、それを welcome.csp として CSP ファイルを保管するディレクトリにコピーすれば、Web 経由で見ることが出来るようになります。
今回の /csp/user の環境では、「Caché インストールディレクトリ/csp/user」に welcome.csp ファイルを保管します。（デフォルトインストールの場合 C:¥intersystems¥cache¥csp¥user）

```
<HTML>
<HEAD><TITLE>ようこそ</TITLE></HEAD>
<BODY>
CSP へようこそ！
</BODY>
</HTML>
```

図 11 welcome.csp:文字の表示

ローカル変数、グローバル変数、計算式を　#( )#　でくくるだけで簡単に値を表示することが出来ます。例えば、以下のようなページの場合

```
<HTML>
<HEAD><TITLE>現在時刻</TITLE></HEAD>
<BODY>
ただいまの時刻は#($zdatetime($horolog,3))#です。
</BODY>
</HTML>
```

図 12 welcome.csp　データ表示:　#()#の追加

$zdatetime()関数は$horolog　特殊変数で取得した現在時刻(数値)を「YYYY-MM-DD HH:MM:SS」というフォーマットで表示するものです。つまり、このページは、表示するたびに現在時刻を求め「YYYY-MM-DD HH:MM:SS」フォーマットに直して表示します。
【ご参考】
Cachéが用意する$zdatetime()関数や$horolog特殊変数などは、「Caché ObjectScript」リファレンスよりご参照いただけます。（Web上ドキュメントは　こちら　から）
http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=RCOS

3.2 フォームデータを表示する

Web アプリケーションでは、フォームと呼ばれる入力画面を作成し、そこで入力されたデータをもとに処理を行うことになります。フォームに入力されたデータを表示する場合、%request オブジェクトを使用します。
使用方法を例示する前に、以下のフォームを作成します。

```
<HTML>
<HEAD>
<META http-equiv=content-type content="text/html; charset=UTF-8">
<TITLE>入力フォーム</TITLE>
</HEAD>
<BODY>
<FORM ACTION=printname.csp>
あなたの氏名を入力してください。
<INPUT TYPE=TEXT NAME=name>
<INPUT TYPE=SUBMIT VALUE="OK">
</FORM>
</BODY>
</HTML>
```

図 13 ログイン画面:Login.html

作成した画面を Login.html とします。

つぎに入力フォームにて OK ボタンを押した際にリクエストされる printname.csp を作成します。ここでは%request オブジェクトにある Get()メソッドを使用し、先ほど入力した name というテキストボックスのフォームデータを取得します。

```
<HTML>
<HEAD><TITLE>ようこそ</TITLE></HEAD>
<BODY>
#(%request.Get("name"))#さん、こんにちは
</BODY>
</HTML>
```

図 14 printname.csp ： %request オブジェクトの練習

＜サンプル HTML、CSP ファイルは Step1 フォルダ以下にあります。＞

※ サンプルでご用意している Login.html は文字コード:UTF-8 で保存しています。

4　処理を行うページをつくる

ここではさらにプログラムロジックを埋め込み、Web アプリケーションとして処理を行う画面を作成します。

4.1　ページ表示時に実行する Caché ObjectScript の記述

CSP ページ内に Caché ObjectScript のロジックを埋め込む場合、以下の例のように＜SCRIPT＞ タグを用います。また、LANGUAGE 属性として、Caché ObjectScript であることを示す”CACHE”を指定します。ページ表示時に実行させるため RUNAT 属性に、”SERVER”を指定します。

```
<HTML>
<HEAD><TITLE>COS サンプル</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="CACHE" RUNAT="SERVER">
 set msg="こんばんは"
 set time=$piece($horolog,",",2) ; 00:00:00 からの経過秒を取得
 if (time>=(5*3600))&(time<(12*3600)) {
        set msg="おはようございます"
 } elseif (time>=(12*3600))&(time<(18*3600)) {
        set msg="こんにちは"
 }
 write msg
 quit
</SCRIPT>
</BODY>
</HTML>
```

図 15 ＜SCRIPT＞タグ:RUNAT="SERVER"属性

あとは</SCRIPT>タグまでの間に Caché ObjectScript でプログラムを書くだけです。上記の例のようにロジックにあわせて出力したい HTML 文書がある場合は、write 文を使用して HTML タグを生成します。

また、Caché ObjectScript 部分では左端の文字は空白あるいはタブでなければなりませんのでご注意ください。

4.2 メソッドの記述

メソッドの記述は、P18 「図 15 <SCRIPT>タグ:RUNAT="SERVER"属性」 で記述した Caché ObjectScript の記述と同様に<SCRIPT>タグを使用します。ただし、以下の例のように RUNAT 属性のかわりに METHOD 属性を指定します。

```
<SCRIPT LANGUAGE="CACHE" METHOD="GreetingMessage">
 set msg="こんばんは"
 set time=$piece($horolog,",",2) ; 00:00:00 からの経過秒を取得
 if (time>=(5*3600))&(time<(12*3600)) {
        set msg="おはようございます"
 } elseif (time>=(12*3600))&(time<(18*3600)) {
        set msg="こんにちは"
 }
 write msg
 quit
</SCRIPT>
```

図 16 <SCRIPT>タグ METHOD 属性

この例では$horolog 特殊変数から 0 時からの経過秒を取得し、5 時から 12 時ならば「おはようございます」、12 時から 18 時なら「こんにちは」、それ以外なら「こんばんは」を表示しています。ただし、このままではページ表示時に実行されませんので、以下のように別の<SCRIPT>タグを設定します。

```
<HTML>
<HEAD><TITLE>ようこそ</TITLE></HEAD>
<BODY>
#(%request.Get("name"))#さん、
<SCRIPT LANGUAGE="CACHE" RUNAT="SERVER">
  do ..GreetingMessage()
</SCRIPT>
<SCRIPT LANGUAGE="CACHE" METHOD="GreetingMessage">
        ... 省略 ...
</SCRIPT>
</BODY>
</HTML>
```

図 17 <SCRIPT>タグ METHOD 属性の呼び出しその 1

この例では GreetingMessage()メソッド内で文字列を表示していますが、これでは汎用性がありませんので、やはりメッセージの文字列を呼び出し元に返したいものです。

そのような場合には、以下の例のように<SCRIPT>タグに RETURNTYPE 属性を設定し、メソッドの終了に使用している QUIT コマンドにパラメータを設定します。

```
<SCRIPT LANGUAGE="CACHE" METHOD="GreetingMessage" RETURNTYPE="%String">
 set msg="こんばんは"
 set time=$piece($horolog,",",2) ; 00:00:00 からの経過秒を取得
 if (time>=(5*3600))&(time<(12*3600)) {
        set msg="おはようございます"
 } elseif (time>=(12*3600))&(time<(18*3600)) {
        set msg="こんにちは"
 }
 quit msg
</SCRIPT>
```

図 18 <SCRIPT>タグ METHOD 属性で利用する RETURNTYPE 属性

ページ表示時に上記のメソッドの戻り値を表示する場合には、#( )#を使用して以下のように記述します。

```
<HTML>
<HEAD><TITLE>ようこそ</TITLE></HEAD>
<BODY>
#(%request.Get("name"))#さん、#(..GreetingMessage())#
<SCRIPT LANGUAGE="CACHE" METHOD="GreetingMessage" RETURNTYPE="%String">
        ... 省略 ...
</SCRIPT>
</BODY>
</HTML>
```

図 19 <SCRIPT>タグ METHOD 属性の呼び出しその 2

画面の作成が完了したら、Login.html を起動し、動作を確認します。
URL は以下の通りです。

http://localhost:57772/csp/user/Login.html

＜ここまでの編集内容は Step2 フォルダ以下の printname.csp をご覧ください。＞

4.3 画面表示時に実行されるメソッド

CSP ファイルをコンパイルすると、それに対応するクラスが生成されます。また Web ブラウザから URL として CSP ファイルを指定すると、CSP サーバーでは以下の順序でメソッドが実行されます。

1. **OnPreHTTP()メソッド**
   Webブラウザに応答を返す前に実行されるメソッド。この部分を記述することによって、プログラムロジックに応じて表示するページを変更することができます。戻り値は%Boolean型で、0 を指定するとそれ以降のメソッド(OnPage(),OnPostHTTP()メソッド)は実行しません。（画面表示用のOnPage()メソッドが呼び出されないため、画面表示が行われません。）
2. **OnPage()メソッド**
   画面表示を行うメソッド。CSPファイルに記述されたHTML文書は、それを表示させるプログラムに変換され、このメソッドに格納されます。
3. **OnPostHTTP()メソッド**
   画面表示終了後に実行されるメソッド。表示ログをデータベースに追加する場合などはここで記述することになります。

これらのメソッドは、「4.2 メソッドの記述」 で解説したメソッドの記述従い、以下のように CSP ファイルに記述します。各メソッドの戻り値が異なりますのでご注意ください。

```
<SCRIPT LANGUAGE="CACHE" METHOD="OnPreHTTP" RETURNTYPE="%Boolean">
        ... 省略 ...
 quit 1
</SCRIPT>
```

```
<SCRIPT LANGUAGE="CACHE" METHOD="OnPage" RETURNTYPE="%Status">
        ... 省略 ...
 quit $$$OK
</SCRIPT>
```

```
<SCRIPT LANGUAGE="CACHE" METHOD="OnPostHTTP">
        ... 省略 ...
 quit
</SCRIPT>
```

図 20 CSP コールバックメソッド :OnXXX()

4.4 処理結果によって表示する画面をかえる

処理結果によって表示する画面をかえるためには、「4.3 画面表示時に実行されるメソッド」 で説明したように OnPreHTTP()メソッドを使用し、その中で%response オブジェクトの ServerSideRedirect プロパティに表示したい csp ファイル名を代入することにより、表示するページの切り替えが行えます。
例えば以下のようになります。

```
<HTML>
<HEAD><TITLE>ようこそ</TITLE></HEAD>
<BODY>
        ... 省略 ...
<SCRIPT LANGUAGE="CACHE" METHOD="OnPreHTTP" RETURNTYPE="%Boolean">
 set status=##class(CSPShop.Customer).FindCustomer(%request.Get("name"),.oid)
 if $$$ISERR(status) {
        set %response.ServerSideRedirect="errormsg.csp"
 }
 quit 1
</SCRIPT>
</BODY>
</HTML>
```

図 21 OnPreHTTP()の利用例

ここでは、入力された name という名前のフォームデータをパラメータとして CSPShop.Customer クラスの FindCustomer()メソッドを実行し、その名前の顧客が存在するかを、確認しています。
もし、その顧客が存在しなければ errormsg.csp へリダイレクトし、画面を表示します。

FindCustomer()メソッドでは、CSPShop.Customer クラスに定義された FindCustomer クエリを実行しています。

```
20 Query FindCustomer(iName As %String) As %SQLQuery(CONTAINID = 1)
21  [
22  SELECT %ID FROM Customer
23   WHERE (Name = :iName)
24  ]
```

図 22 CSPShop.Customer クラスの FindCustomer クエリ

＜ここまでの内容は、Step3 フォルダ以下にあります＞

サンプルの Step3 フォルダには、CSP の画面表示に利用している CSPShop.Customer クラスとデータ自動生成に必要なクラスをエクスポートした XML ファイルが含まれています。XML ファイルを USER ネームスペースにインポートした後、ターミナルで、以下クラスメソッドを実行するとサンプルデータが自動で生成されます。

```
Do ##Class(CSPShop.Customer).Populate(n)   // n には作成件数を指定します
```

作成されたデータの確認には、管理ポータルの SQL メニューを利用します。
管理ポータル→システムエクスプローラ→SQL　を選択し、USER ネームスペースに切り替えます。

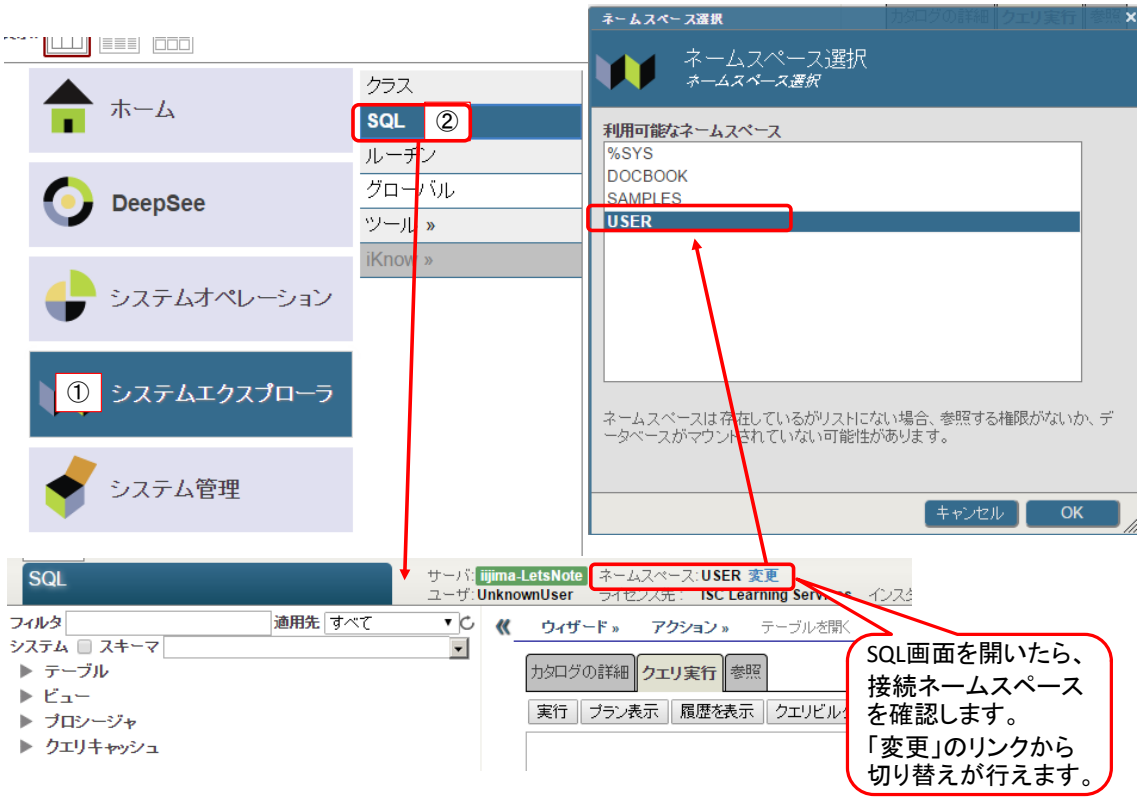


図 23 サンプルデータの確認：管理ポータル：SQL 画面の開き方

続いて、今回使用する CSPShop.Customer テーブルを参照します。

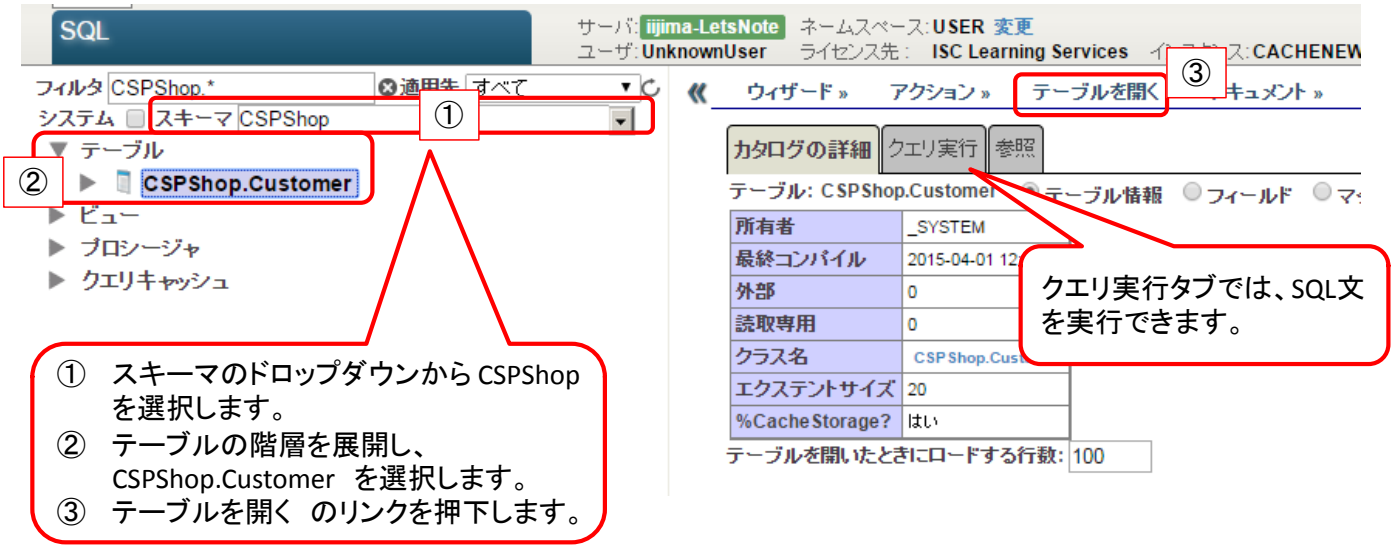


図 24 サンプルデータの確認:CSPShop.Customer テーブルの参照

CSP画面の確認は、Login.htmlの表示から行います。

http://localhost:57772/csp/user/Login.html

Login.html のテキストボックスには、Customer テーブルの Name カラムに登録のある名前を指定します。

図 25 CSP ページの動作確認(Login.html→printname.csp)

また、%responseオブジェクトの使用方法については、Web上ドキュメント、または以下ドキュメントをご参照ください。

[ドキュメント] > [Caché Web 開発] > [Caché Server Pages (CSP)の使用法]

3.5 %CSP.Response オブジェクトおよび OnPreHTTP メソッド

4.5 エラー発生時の表示画面を設定する

アプリケーション実行中に何らかのエラーが発生した場合、通常以下の画面が表示されます。

図 26 デフォルトの CSP エラーページ

この画面を変更するには、まず、変更したいページを作成します。

基本的にはCSPページですが、以下の例のように<csp:class>タグを利用して、スーパークラスの指定を変更します。通常の CSP ページは、%CSP.Page クラスをスーパークラスに持っていますが、エラーページを作成する場合は、スーパークラスに%CSP.Error を指定します。

```
<HTML>
<CSP:CLASS SUPER="%CSP.Error">
<HEAD><TITLE>エラーが発生しました</TITLE></HEAD>
<BODY>
お手数ですが、再度ログインくださいますようお願いいたします。
<SCRIPT LANGUAGE=CACHE RUNAT=SERVER>
 set status=%request.Get("Error:ErrorCode")
 do ..DecomposeError(status,.err)
 set errcode=err(1,"Error")
 set user=$get(%session.Data("user"))
 set id=$I(^ErrorLog)
 set ^ErrorLog(id)=$zdatetime($horolog,3)_";"_user_";"_errcode
</SCRIPT>
</BODY>
</HTML>
```

図 27 エラーページ<CSP:CLASS>タグの SUPER 属性の指定

エラーページではフォームデータと同様に、%request.Get()メソッドを使用し、以下のエラー情報を取得できます。

- **Error:ErrorCode**
  %Status 形式でエラーメッセージを保持しています。
- **Error:ErrorNumber**
  発生したエラーのエラー番号を保持しています。
- **Error:Namespace**
  エラーが発生したネームスペース名
- **Error:URL**
  エラーが発生した際に指定した URL

エラーページの登録は以下の方法があります。

- ウェブ・アプリケーション設定画面でアプリケーションのエラーページを変更する
  管理ポータル→セキュリティ→アプリケーション→ウェブ・アプリケーション→/csp/user の編集を選択します。

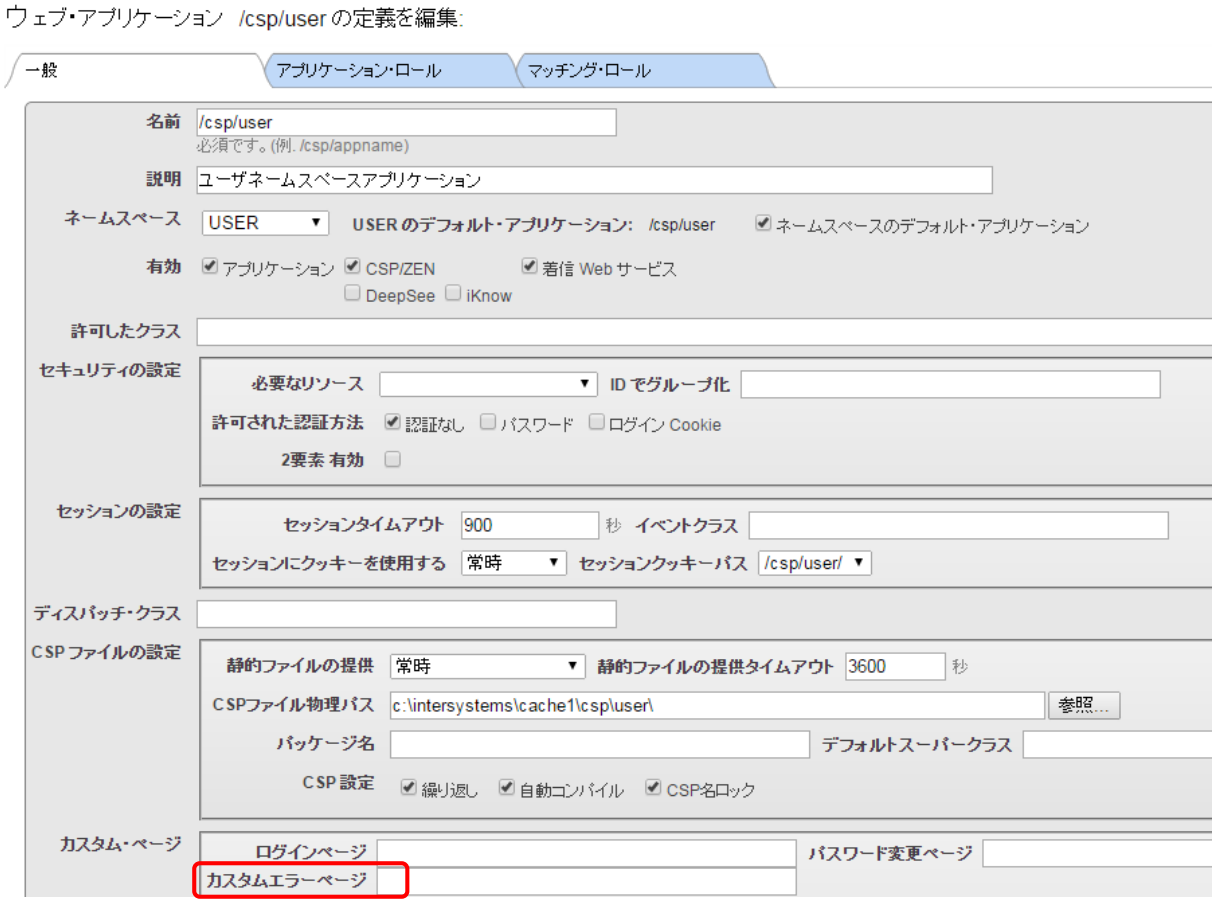


図 28 ウェブ・アプリケーションパス毎のカスタムエラーページの指定

カスタムエラーページの欄に、作成したエラーページを指定します。(例:errpage.csp)

- ページごとにエラーページを定義する

  以下の例のように、<CSP:CLASS>タグに ERRORPAGE 属性を追加、作成したエラーページの CSP ファイル名を指定します。

```
<HTML>
<CSP:CLASS ERRORPAGE="errpage.csp">
<HEAD><TITLE>ようこそ</TITLE></HEAD>
<BODY>
        ... 省略 ...
</BODY>
</HTML>
```

図 29 <CSP:CLASS>タグ ERRORPAGE 属性の指定

5　作業データをサーバーで保管する

Web アプリケーションが高度になり、多数の画面で構成されるようになると、アプリケーションの状態をどこかに保存する必要がでてきます。ここではそのような場合に作業データを保存する方法や、保存されているデータを識別するためのセッションについて説明します。

5.1　セッションとは

Web アプリケーションにアクセスしたユーザが行う一連の操作をセッションといいます。Caché ではフォームデータや Cookie にセッション ID と呼ばれる ID を埋め込むことにより、Web ブラウザからのリクエストが一連の操作であるかどうかを管理しています。また、セッションごとにデータを保持することも出来ます。これにより状態や作業データをサーバー側に保持し、効率の良い Web アプリケーションが構築できます。

セッションに関する情報は%sesssion オブジェクトを使用します。%session オブジェクトは Data プロパティを持っており、そこにデータを設定すると、それらは一旦データベースに格納され、次回の処理時に取得できるため、作業データが保持されているように見えます。

たとえば、インターネットショッピングのシステムを考えた場合、ユーザ認証から商品を注文するまでの間に、ユーザ情報や選択している商品の名称、数量を保持する必要があります。そのため、＜Input type＝hidden＞タグを使用してフォーム上にさまざまな情報を埋め込んでいました。
CSP ではこれらの情報を%session の Data プロパティに逐一格納することにより、余計なタグやフォームデータを減らすことができます。

5.2 作業データを保存、取得する

作業データの保存、取得は%session オブジェクトの Data プロパティを使用します。Data プロパティは多次元構造になっていますので、ローカル変数や、グローバル変数と同じ感覚で操作できます。

例えば、データを保存するには以下のように%session オブジェクトの Data プロパティに値を代入します。

```
<SCRIPT LANGUAGE="CACHE" RUNAT="SERVER">
 set %session.Data("username")=%request.Get("name")
</SCRIPT>
```

図 30 %session.Data プロパティの利用例

Data プロパティは多次元配列になっていますので、以下のように"order"と商品名といった複数のキーを指定して値を保持することもできます。

```
<SCRIPT LANGUAGE="CACHE" RUNAT="SERVER">
 set article=%request.Get("article"),amount=+%request.Get("amount")
 if amount>0 {
        set %session.Data("order",article)=amount
 } else {
        kill %session.Data("order",article)
 }
</SCRIPT>
```

図 31 %session.Data プロパティ 多次元配列の例

数量が 0 の注文を入れると、該当する商品のエントリを消去します。この場合、kill コマンドを使用して Data プロパティの一部を消去できます。

作業データの取得は必要なデータをローカル変数に代入します。
以下の例の場合、%session.Data(“order”,article) のエントリが存在しない場合、<UNDEFINED>エラーが発生しますのでご注意ください。

```
<SCRIPT LANGUAGE=”CACHE” RUNAT=”SERVER”>
 set article=%request.Get(“article”)
 set amount=%session.Data(“order”,article)
</SCRIPT>
```

定義されていないエントリにアクセスする可能性がある場合は以下の例のように$get()関数を使用すると良いでしょう。

```
<SCRIPT LANGUAGE=”CACHE” RUNAT=”SERVER”>
 set article=%request.Get(“article”)
 set amount=$get(%session.Data(“order”,article))
</SCRIPT>
```

5.3 セッションのタイマー値の調整

セッションといえども Web ブラウザとサーバーとが常時接続されているわけではありません。そのため、Web ブラウザを閉じたり、クライアント PC が途中で終了した場合などは、サーバーへのアクセスが行われず、サーバー側ではクライアントが終了したことを判定できません。そこでセッションにタイマーを設け、一定時間にアクセスがなかった場合は自動的に保持している作業データやライセンスを開放する必要があります。

セッションのタイマー値は、管理ポータルでウェブ・アプリケーション毎に設定できますが、処理に応じてタイマー値を変更したい場合は以下のようにタイマー値を変更することが可能です。

```
<SCRIPT LANGUAGE=”CACHE” RUNAT=”SERVER”>
 set %session.AppTimeout=300
</SCRIPT>
```

図 32 %session.AppTimeout の設定

また、%session.AppTimeout プロパティに 0 を代入しますと、タイマーが作動せず、作業データ、ライセンスが残ったままとなりますので、ご注意ください。

5.4 セッションを終了する

セッションを終了する場合は、実行する CSP ファイルにて %session オブジェクトの EndSession プロパティに 1 を代入します。

```
<HTML>
<HEAD><TITLE>ご利用ありがとうございました</TITLE></HEAD>
<BODY>
#(%session.Data("username"))#さん、
またのご利用をお待ちしております。
<SCRIPT LANGUAGE="CACHE" RUNAT="SERVER">
 set %session.EndSession=1
</SCRIPT>
</BODY>
</HTML>
```

図 33 セッションの終了（%session.EndSession＝1）

5.5 セッションタイムアウト時、ログアウト時の処理を設定する

Web アプリケーションでは、セッション中に常時起動しているプロセスというものは存在しません。そのため排他制御を実現する場合、ロックが使用できないため、グローバル変数やクラスを利用してフラグを設定することがあります。そうするとデッドロックを起こさないためにタイムアウトやログアウト時にセッションのイベントを利用してそれらのフラグを元に戻す必要があります。

イベント処理メソッドを実行させるには、まず%CSP.SessionEvents クラスを継承した新たなクラスを作成します。ここで、セッションが終了したとき(タイムアウトが発生した場合も含む)に、処理を行いたい場合、OnEndSession()メソッドをオーバーライドします。
タイムアウトが発生した場合のみ処理を行いたい場合、OnTimeout()メソッドをオーバーライドします。
以下の例ではセッション終了時に^LockData グローバルのフラグを消去し、ユーザに対する仮想的なロックを解除する処理を記述しています。

```
/// セッションイベント処理クラス
Class Shop.SesEvent Extends %CSP.SessionEvents [ ProcedureBlock ]
{
ClassMethod OnEndSession()
 {
        if $data(%session.Data("username")) {
                kill ^LockData("USER",%session.Data("username"))
        }
        Quit
 }
}
```

図 34 セッションイベント処理クラスの例

さらに、以下の例のように CSP ファイル内で%session オブジェクトの EventClass プロパティに、作成したクラスの名称を設定します。

```
<SCRIPT LANGUAGE="CACHE" RUNAT="SERVER">
 set %session.AppTimeout=300
 set %session.EventClass="Shop.SesEvent"
</SCRIPT>
```

図 35 イベントクラスの指定(ページ単位での例)

(セッションイベントクラスは、ウェブ・アプリケーションの設定単位でも指定できます。)
以上でタイムアウトやセッションを終了した時点で処理が行われるようになります。

6　JavaScript を使った動的なページをつくる

ここではハイパーイベントを使用した、JavaScript による動的なページの作成方法について説明します。

6.1　ハイパーイベントとは

ハイパーイベントとは、JavaScript から Caché を呼び出す仕組みで、現在表示している画面をリフレッシュしなくても、Caché で処理した結果を JavaScript で画面に反映することが出来ます。たとえば、以下のプログラムのように各テキストボックスの値を変更すると、合計金額が更新されます。

```
<HTML>
<HEAD><TITLE>合計金額計算</TITLE></HEAD>
<BODY>
<FORM NAME=form>
<TABLE border=1><TR><TD>品名</TD><TD>単価</TD><TD>数量</TD></TR>
  <TR><TD>ノート PC</TD><TD>¥200,000</TD>
      <TD><INPUT TYPE=TEXT NAME=NOTEPC ONBLUR="update();"></TD></TR>
  <TR><TD>デジカメ</TD><TD>¥40,000</TD>
      <TD><INPUT TYPE=TEXT NAME=CAM ONBLUR="update();"></TD></TR>
  <TR><TD>ハードディスク</TD><TD>¥20,000</TD>
      <TD><INPUT TYPE=TEXT NAME=HDD ONBLUR="update();"></TD></TR>
  <TR><TD COLSPAN=2>合 計</TD>
      <TD><INPUT TYPE=TEXT NAME=TOTAL VALUE=0></TD></TR>
</TABLE>
<SCRIPT LANGUAGE=JavaScript>
function update() {
        form.TOTAL.value = #server(..Calc(form.NOTEPC.value,form.CAM.value,
                form.HDD.value))#;
}
</SCRIPT>
<SCRIPT LANGUAGE="CACHE" METHOD="Calc"
        ARGUMENTS="a:%Numeric,b:%Numeric,c:%Numeric" RETURNTYPE="%Numeric">
        quit $fnumber(200000*a+(40000*b)+(20000*c),",")
</SCRIPT>
</BODY>
</HTML>
```

図 36 ハイパーイベント #server()# の例

この場合、数量の欄からフォーカスが他のフィールドに移るとONBLUR 属性に従って、update()という JavaScript の関数が呼ばれます。その関数内で、以下のように#server( )#でくくられた Caché の Calc()メソッドを呼んでいます。パラメータにはノート PC、デジカメ、ハードディスクの数量を入れています。

Calc()メソッドの戻り値は合計欄に入れられます。
Caché の Calc()メソッドには、引数の ARGUMENTS 属性があります。ARGUMENTS 属性では、仮引数とその型を指定しています。

```
<SCRIPT LANGUAGE="CACHE" METHOD="Calc"
        ARGUMENTS="a:%Numeric,b:%Numeric,c:%Numeric" RETURNTYPE="%Numeric">
        quit $fnumber(200000*a+(40000*b)+(20000*c),",")
</SCRIPT>
```

Calc()メソッドでは受け付けた各商品の数量に単価を掛け合わせ、$fnumber()関数によって 3 桁ごとに「,」をつけるようにしています。

なお、ハイパーイベントは、元の画面のまま長時間放置することによってセッションが終了した場合、実行できずエラーが発生しますのでご注意ください。

6.2 動的に javascript を生成する

P34「ハイパーイベントとは」で説明した例のように、Caché のメソッドの戻り値を JavaScript に渡す方法のほかに、Caché メソッド内で JavaScript を生成し、ブラウザ上で実行することも出来ます。そうすることで複数の値を Caché から JavaScript に渡すことが出来ます。
下の例では ^OrderData に注文情報を定義しておき、「前」「次」といったボタンをクリックするとハイパーイベントで Caché の Rec()メソッドが実行されます。

Rec メソッドには&JS< >でくくられた部分があり、これが JavaScript としてクライアントに渡され、実行される部分となります。例では ^OrderData に対して$Order 関数を実行し、次のレコード、前のレコードの ID を求めた上で、注文番号、発注者、金額欄に ^OrderData の値を代入する JavaScript を生成しています。

```
<HTML><HEAD><TITLE>注文表示</TITLE></HEAD>
<BODY><SCRIPT LANGUAGE=CACHE RUNAT=SERVER>
 kill ^OrderData
 set ^OrderData(1)="佐藤;230000",^OrderData(2)="鈴木;235000"
 set ^OrderData(3)="中村;4230000",^OrderData(4)="吉本;2321000"
 set %session.Data("rec")=""
</SCRIPT>
<FORM NAME=form>
<TABLE border=1>
<TR><TD>注文番号</TD><TD><INPUT TYPE=TEXT NAME=ORDNO SIZE=6></TD>
<TD>発注者</TD><TD><INPUT TYPE=TEXT NAME=NAME SIZE=20></TD></TR>
<TR><TD COLSPAN=2>合 計</TD>
<TD COLSPAN=2><INPUT TYPE=TEXT NAME=TOTAL></TD></TR>
</TABLE>
<INPUT TYPE=BUTTON ONCLICK="#server(..Rec(1))#;" VALUE="次">
<INPUT TYPE=BUTTON ONCLICK=" #server(..Rec(-1))#;" VALUE="前">

<SCRIPT LANGUAGE=CACHE METHOD="Rec" ARGUMENTS="dir:%Numeric">
 set id=$order(^OrderData(%session.Data("rec")),dir)
 if id'="" {
        set name=$piece(^OrderData(id),";")
        set total=$piece(^OrderData(id),";",2)
        &JS<self.document.form.ORDNO.value = #(id)#;
            self.document.form.NAME.value = '#(name)#';
            self.document.form.TOTAL.value = #(total)#;>
        set %session.Data("rec")=id
 } else {
         &JS<alert('データはありません');>
 }
</SCRIPT></FORM>
</BODY></HTML>
```

＜上記内容は Step3 フォルダにあります。(displayorder.csp)＞

6.3 ハイパーイベント処理中のエラー処理をおこなう

P25「エラー発生時の表示画面を設定する」で作成したエラーページに、以下のようにHyperEventError()メソッドを追加し、そこで実行するコードを記述します。ハイパーイベント処理中にエラーが発生すると、HyperEventError()メソッドが実行し、^ErrorLog にエラー情報が記録され、「エラーが発生しました」といった警告画面が表示されます。

```
<SCRIPT LANGUAGE=CACHE METHOD="HyperEventError">
 set status=%request.Get("Error:ErrorCode")
 do ..DecomposeError(status,.err)
 set errcode=err(1,"Error")
 set user=$get(%session.Data("user"))

 set id=$INCREMENT(^ErrorLog)
 set ^ErrorLog(id)=$zdatetime($horolog,3)_";"_user_";"_errcode
write "alert('エラーが発生しました。お手数ですが、再度ログインをお願いします');",!
 write "ret=1;",!
```

図 37 ハイパーイベント処理中のエラーメッセージ カスタマイズ