Обмен данными

Привязка объектов к форме

- Простейший метод выполнить привязку объекта к форме - специальный тег **CSP:Object**

```
<csp:object name="obj" classname="Sample.Person" oid=1>
```

- Тег <csp:object> открывает объект.
- **name** - задает имя для использования на Web-странице
- **className** - устанавливает класс объекта
- **oid** - ID объекта внутри класса (если oid=””, то создается новый объект).

Привязка объектов к форме

- Для непосредственной привязки вызванного объекта к форме используется атрибут **cspbind:**

```
<form method cspbind=obj name=Employee>
Username:
<input type=TEXT name="Username" cspbind=Username><br>
…
```

- Если осуществляется привязка объекта к форме csp-страницы тегом <csp:object>, Caché автоматически создает методы ***formname_new***() и ***formname_save***() (где formname – имя формы) для добавления новых объектов и сохранения изменений.

Упражнение 2-0

Гиперсобытия (hyperevents)

- ***Гиперсобытия*** – совокупность серверного и клиентского кода (часть технологии CSP), позволяющая вызывать код на стороне сервера и получать результат на клиенте без перезагрузки страницы.
- Поддерживаются 2 типа вызова:
  - Синхронный (**#server**)
  - Асинхронный (**#call**)
- Отправляют зашифрованный HTTP-запрос (XMLHttpRequest) через веб-сервер на сервер Caché.

Синхронный запрос

- **Синтаксис:**

```
#server(package.class.method(JavaScript args))#
#server(..method(JavaScript args))#
```

- **Атрибуты**:
  - ***method*** – любой валидный **метод класса**.
  - ***args*** – это переменные JavaScript или ссылки на объектную модель документа (DOM), которые будут переданы на сервер в виде строки.
- **Описание:**
  - Блокирует работу браузера до завершения выполнения метода на сервере.
  - Используется для вызова методов, которые возвращают значения через стандартный механизм.

Асинхронный запрос

- **Синтаксис:**

```
#call(package.class.method(JavaScript args))#
#call(..method(JavaScript args))#
```

- **Описание:**
  - Используется для вызова методов **не** возвращающих значения через стандартный механизм.
  - Не блокирует окно браузера во время выполнения метода.
  - Может возвратить значение только за счет внутренней генерации JavaScript (*&js<…>*) .

Время ожидания сервера

- **Важно**:
  - При длительных серверных операциях, превышающих таймаут соединения CSP, вне зависимости от используемого типа запросов, соединение будет разорвано. Это в свою очередь приведет к ошибке «*Unable to process HyperEvent*».

Упражнение 2-1

Упражнение 2-2

Веб-сокеты (2013.1+)

- Стандартные средства Caché позволяют реализовать «общение» клиентской части с серверной с использованием протокола WebSocket.
- Главной особенностью данного механизма является поддержания соединения с сервером вне зависимости от установленного в настройках CSP таймаута.
- Веб-сокеты прекращают работу по завершению сессии или же после собственного уничтожения.

%CSP.WebSocket

- Для создания сокета на сервере от программиста требуется создать свой класс, унаследовав его от класса %**CSP.WebSocket**, и переопределить в нём несколько методов.
- Все WebSocket-сервера наследуются от %CSP.**WebSocket**.

Методы и свойства %CSP.WebSocket

Метод/Свойство	Описание
Read()	получить данные от клиента
Write()	отправить данные клиенту
OnPreServer()	обработчик события PreServer: вызывается перед стартом WebSocket-сервера
OnPostServer()	обработчик события PostServer: вызывается после останова WebSocket-сервера
Server()	собственно сам WebSocket-сервер
EndServer()	остановить WebSocket-сервер
AtEnd	свойство принимает значение true (1), когда во время чтения, WebSocket-сервер достигает конца текущего кадра данных
SharedConnection	свойство определяет, будет ли обмен информацией между клиентом и WebSocket-сервером происходить по выделенному соединению CSP-Шлюза или через пул разделяемых соединений (пока не используется)

Клиентская часть сокета (JavaScript)

- Адрес вызова сокета:

```
var url = socketProtocol + '://' + host + 'package.class.cls';
```

- Значения:
  - ***socketProtocol*** – ws или wss в зависимости от использования ssl.
  - ***host*** – адрес сервера сокета. В случае совпадения с адресом сервера CSP можно сгенерировать через:

```
var host = window.location.host + '#(%request.Application)#'
```

  - ***class*** – имя класса сокета (наследника %CSP.WebSocket).

Клиентская часть сокета (JavaScript)

- Кроссбраузерный источник для создания сокетов:

```
var Socket = window['MozWebSocket'] ? MozWebSocket
                    : window['WebSocket'] ? WebSocket : null;
```

- Создание объекта сокета:

```
_socket = new Socket( url );
```

- Обработчик основного события:

```
_socket.onopen = function() { … }
```

данный обработчик в свою очередь содержит переопределение основных функций сокета.

Основные функции сокета (**JavaScript**)

- Для корректной работы клиентской части необходимо переопределить следующие функции:

```
//Обработка полученных сообщений
_socket.onmessage = function(e) {
    ...
};

//Обработчик закрытия сокета
_socket.onclose = function(e) {
    _socket = null;
};

//Обработчик ошибок
_socket.onerror = function(e) {
    ...
};

//Отправка сообщения серверу
_socket.send('Hello world!');
```

Упражнение 2-3

JSON

- **J**ava**S**cript **O**bject **N**otation - текстовый формат описания сложных структур данных, принятый в JavaScript.
- Данные в формате JSON представляют собой значения или JavaScript-объекты { ... } или массивы [...], содержащие значения одного из типов:
  - строки в двойных кавычках
  - число
  - логическое значение true/false
  - null

Пример JSON

```
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "streetAddress": "Московское ш., 101, кв.101",
        "city": "Ленинград",
        "postalCode": 101101
    },
    "phoneNumbers": [
        "812 123-1234",
        "916 123-4567"
    ]
}
```

Передача данных в формате JSON

- Наиболее удобным инструментом для работы с JSON на стороне Caché является класс %ZEN.Auxiliary.**jsonProvider**.
- Данный класс содержит методы для конвертации данных как в так и из JSON.
- Для работы с данными на клиентской части можно использовать как стандартный js класс JSON, так и класс предоставляемый js библиотекой поставляемой ISC - **zenCSLM.js**

Пример обмена данными

- Передача с сервера:

```
    try{
        $$$ThrowOnError(##class(%ZEN.Auxiliary.jsonProvider).%
WriteJSONStreamFromObject(.stream,obj,,,1,"aeloq"))
        s res=stream.Read()
    }catch ex{
        d ex.Log()
    }
    q res
```

- Прием на клиенте:

```
    var str=#server(sp.utils.getList())#;
    var listOfData=ZLM.jsonParse(str)
```

Упражнение 2-4

Tips & Tricks

Использование %ZEN.**proxyObject**

- В Cache существует универсальный объект для передачи информации между клиентом и сервером.
- Объект класса %ZEN.**proxyObject** может иметь произвольный набор параметров, которые задаются «на лету».

```
s proxyObj=##class(%ZEN.proxyObject).%New()
s proxyObj.Name = “Иван”
s proxyObj.SSN = 2235623
s proxyObj.Arr = ##class(%ListOfDataTypes).%New()
```

Запуск долгих процессов

- Несмотря на ограничения, которые накладываются таймаутом ожидания сервера, существует способ запуска долгих процессов через **гиперсобытия**.
- Для этого используется стандартный для веб-приложений механизм запуска фоновых процессов и опроса сервера.
- В Cache для запуска фоновых процессов применяется команда **Job:**

```
job ..SomeMethod(params)
```

Запуск долгих процессов

- Общая схема запуска на сервере может выглядеть следующим образом:

```
ClassMethod Test(N As %Integer)
{
        s ^calkResult=""
        j ..SpecialJob(N)    //Запускаем фоновый процесс
}

ClassMethod SpecialJob(N As %Integer)
{
        for i=1:1:N   //Имитируем долгий серверный процесс
        {
                h 10
        }
        s ^calkResult=1
        //Оповещаем о завершении процесса
        d $system.Event.Signal($zparent,”end”)
        q
}
```

Запуск долгих процессов (клиент)

- На клиенте асинхронно вызывается метод запуска расчета и периодический опрос:

```
//Вызов расчета
#call(sp.main.TestApp.Test(1000))#;

//Вызов опроса раз в 3 секунды.
resultTimer = setInterval(function() {
      #server(sp.main.TestApp.GetResult())#
}, 3000);
```

Запуск долгих процессов (сервер)

- Метод опроса на сервере:

```
ClassMethod GetResult()
{
	if (^tempRes'="")
	{
		&js<
			alert("Расчет окончен");
			//Прекращаем опрос
			clearInterval(resultTimer);
		>
	}
}
```

Загрузка файлов (**upload**)

- Cache предоставляет стандартный механизм загрузки файлов клиента на сервер.
- С помощью компонента:

```
<form action="upload.csp" enctype="multipart/form-data"
method="post">

   <input type=file size=30 name=FileStream>

</form>
```

Загрузка файлов (**upload**)

- Можно получить все данные выбранного файла:

```
<csp:if condition='($data(%request.MimeData("FileStream",1)))'>

    #(..EscapeHTML(%request.MimeData("FileStream",1).FileName))#
    #(..EscapeHTML(%request.MimeData("FileStream",1).Size))#
    #(..EscapeHTML(%request.MimeData("FileStream",1).MimeSection))#
    #(..EscapeHTML($classname(%request.MimeData("FileStream",1))))#
    #(..EscapeHTML(%request.MimeData("FileStream",1).ContentType))#

    <script language="Cache" runat="server">
            New bytes
            Set bytes=%request.MimeData("FileStream",1).Read(200)
            Set bytes=##class(%CSP.Utils).DecodeData(bytes)
            Write bytes,!
    </script>

</csp:if>
```

Упражнение 2-5

Документация

- Образцы CSP:
  http://localhost:57772/csp/samples/menu.csp
- Правила CSP
  http://localhost:57772/csp/samples/rulemgr.csp
- Документация по CSP Gateway :
  http://localhost:57773/csp/bin/Systems/Module.cxw
  (раздел «Справка» в правом верхнем углу)