Введение в REST

Representational state transfer

- **REST** (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб.
- Любые данные однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных.

Свойства REST

- Разделение клиента и сервера
- Независимость от состояния (stateless)
- Кэшируемая и многоуровневая архитектура
- Единый интерфейс
- Все запросы к RESTful web API состоят из корневого URL приложения плюс частные подзапросы
- CRUD (Create , Read, Update, Delete) через HTTP

Преимущества REST

- Независимость от языка/фреймворка/платформы
- Легкость разработки
  - Проще чем SOAP
  - Нет необходимости в специальных инструментах
- Соответствует дизайну и принципам Web
  - Нет необходимости в дополнительных сообщениях
- Легковесность

Методы REST (коллекции)

Метод	**Значение** (example.com/resources)
GET	Получить список URI элементов коллекции, возможно доп. информацию
PUT	Заменить существующую коллекцию на новую
POST	Создать новый элемент коллекции
DELETE	Удалить всю коллекцию

Методы REST (элемент)

Метод	**Значение** (example.com/resources/itemID)
GET	Получить всю информацию об элементе
PUT	Заменить существующий элемент на новый
POST	Обычно не используется. Но может использоваться одновременно для всех действий изменения в зависимости от используемого «REST-Pattern»
DELETE	Удалить элемент коллекции

Класс-брокер

- Чтобы создать REST приложение, нужно в настройках веб-приложения Caché определить класс-брокер, в котором указываются возможные расширения базового URL и соответствующие действия приложения при запросе этих расширений.
- Класс-брокер создается как наследник класса %**CSP.REST.**

Обработка запросов к брокеру

- Стандартный вид пути:

```
<Route
    Url="/path/:param1/:param2"
    Method="GET"
    Call="Package.Class:ClassMethod"
/>
```

- **ClassMethod** – любой метод класса Caché, обрабатывающий JSON

Пример класса-брокер

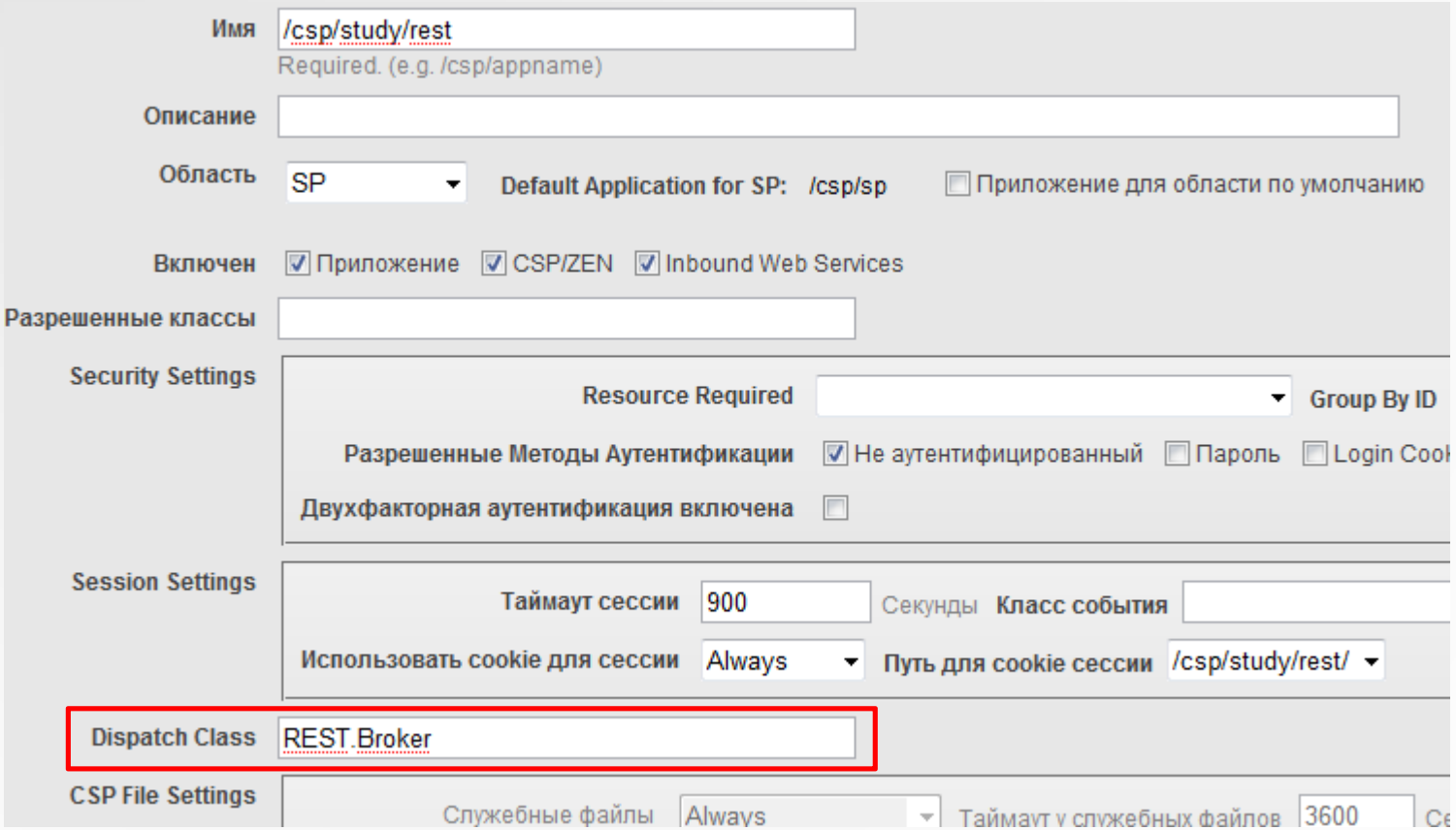
```
Class REST.Broker Extends %CSP.REST
{
XData UrlMap
 {
 <Routes>
  <Route Url="/test" Method="GET" Call="Test"/>
 </Routes>
 }
}
```

- В данном примере Test – метод вызываемый при получении URL вида:

```
http://<адрес сервера>/<rest приложение>/test
```

Настройка приложения

- Для создаваемого RESTful приложения необходимо указать в качестве Dispatch Class созданный брокер:



Передача данных (**JSON**)

- Наиболее популярный способ передачи данных при работе в REST приложениях является JSON.
- За счёт своей читабельности и удобству обработки на клиенте, по сравнению с XML, формат JSON может быть более подходящим для представления сложных данных.

Пример **JSON**

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address":
       {
          "street": "Арбузная 177. д 3.",
          "city": "Новосибирск",
          "postalCode": 101101
       },
   "phoneNumbers": [ "812 123-1234", "916 123-4567" ]
}
```

Пары ключ/значение

Вложенный объект

Массив строк

Классы для работы с **JSON** в Caché

- Начиная с версии 2013.2 в Caché появилась поддержка REST и расширенная поддержка JSON.
- Классы для работы с JSON:
  - %ZEN.Auxiliary.**jsonProvider**;
  - %ZEN.Auxiliary.**jsonArrayProvider**;
  - %ZEN.Auxiliary.**jsonSQLProvider**;
  - %ZEN.**proxyObject**.

Класс %ZEN.Auxiliary.**jsonProvider**

Метод	**Описание** (*могут использоваться вне ZEN*)
WriteJSONFromArray	Генерация строки в формате JSON из массива Cache
WriteJSONFromObject	Генерация строки в формате JSON из объекта Cache
WriteJSONStreamFromArray	Генерация строки JSON в поток из массива Cache
WriteJSONStreamFromObject	Генерация строки JSON в поток из объекта Cache
ConvertJSONToObject	Обратное преобразование из строки JSON в объект класса %ZEN.**proxyObject**, если не указан параметр *pTargetClass*

Класс %ZEN.**proxyObject**

- ***proxyObject* -** наследник класса %RegisteredObject.
- Используется в качестве транспортного объекта, поскольку позволяет создавать свойства динамически:

```
set student  = ##class(%ZEN.proxyObject).%New()
set student.name = "Иван"
set student.age = 21
set student.subjects = ##class(%ListOfDataTypes).%New()
do student.subjects.Insert("История")
do student.subjects.Insert("Химия")
do student.subjects.Insert("Физика")
```

Класс %ZEN.**proxyObject**

- В результате, данный объект на клиенте может иметь следующий вид js-объетка:

```
var student = {
        name: 'Иван',
        age: 21,
        subjects: ['История', 'Химия', 'Физика']
};
```

- Кроме того класс proxyObject содержит метод для преобразования в JSON:

```
do student.%ToJSON()
```

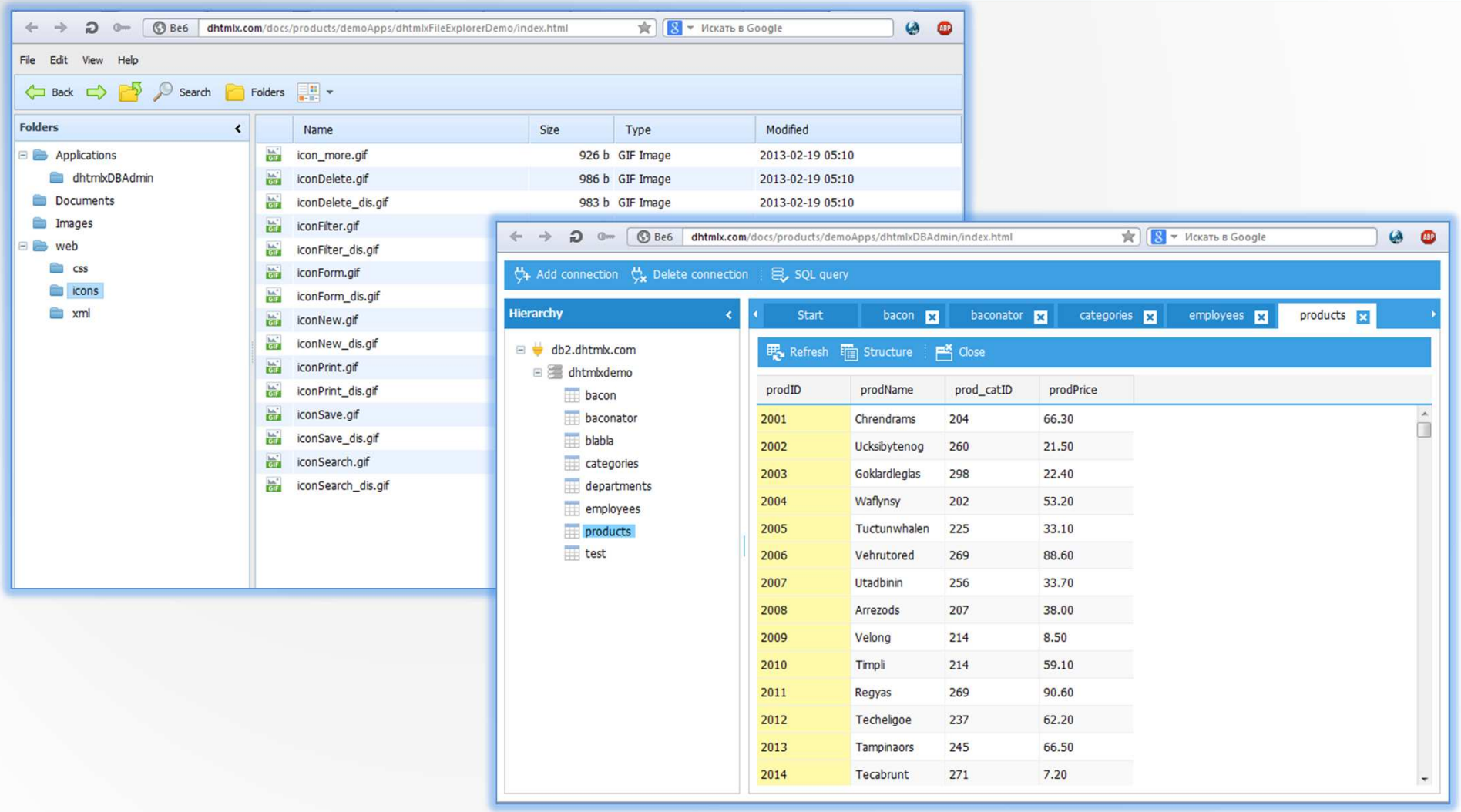
```
{"name":"Иван","age":21,"subjects":["История","Химия","Физика"]}
```

Упражнение 1-1

DHTMLX

- DHTMLX – компонентная библиотека, которая предоставляет простое и чистое решение для создания веб интерфейсов при помощью JavaScript/CSS.
- Основное преимущество – используется с большинством существующих серверных технологий.
- Поддерживает HTML5 решения, для использования на тач-скринах.
- Распространяется свободно под лицензией GNU GPL v2. Существует коммерческая PRO-версия, с расширенными функциями и компонентами.

Пример DHTMLX



DHTMLX и REST

- В качестве источника данных компоненты DHTMLX могут использовать REST-приложения.
- При использовании функций по-умолчанию компоненты опрашивают REST-приложение и принимают строку в формате JSON или XML.
- Для рэндеринга по-умолчанию данные должны иметь определенную структуру:

```
{
    rows:[
        { id:1, data: ["A Time to Kill", "John Grisham", "100"]},
        { id:2, data: ["Blood and Smoke", "Stephen King", "1000"]},
        { id:3, data: ["The Rainmaker", "John Grisham", "-200"]}
    ]
};
```

DHTMLX и REST

- Основной метод для загрузки данных:

```
mygrid.load(url,"json");
```

- При работе с сформированными JSON объектами применяется метод:

```
mygrid.parse(jsstring,"json");
```

Замечание по DHTMLX и REST

- JavaScript придерживается политики **Same Origin Policy**. T.e. js в окне браузера может делать запросы только к страницам находящимся на том же домене которому принадлежит объект окна Причём два окна могут обмениваться информацией только если у них совпадают домен, порт и протокол (окна http не могут общаться с окнами https).
- Для обхода данного ограничения применяются различные приемы с iframe, серверными скриптами или же технологиями **JSONP** или **CORS**

Упражнение 1-2