ModCBroker: Руководство программиста
Версия 3.2.0
www.gradsoft.kiev.ua
DocumentId: GradSoft-PR-r-05.12.2000-3.2.0

October 15, 2008

Contents

1 Введение

`ModCBroker` предоставляет простой но мощный механизм разработки WEB приложений и интеграции CORBA приложений в WEB. Работа `ModCBroker`, как и любого другого сервера приложений, заключается в трансляции поступающих http запросов в CORBA запросы, и предоставлении программисту программного интерфейса (API) для чтения параметров и вывода результата запросов в WWW браузере пользователя.

Т. е. в общем виде картина выглядит так:

$$(\text{CORBA Server}) \leftarrow \frac{Intranet}{CORBA} \rightarrow (\text{cbroker}) \leftarrow \frac{Internet}{http} \rightarrow (\text{WWW browser})$$

Предоставляемое API полностью описанно в разделе **IDL интерфейсы** 7, с точки зрения "птичьего полета" процесс построения WWW приложения с помощью `cbroker` можно описать следующим образом:

1. Автор приложения реализует интерфейсы HTTP::Servlet и HTTP::Handler.
2. В HTTP::Servlet происходит авторизация запроса, и если она произошла удачно, то создается Handler (обработчик).
3. ModCBroker вызывает функцию handle, где параметрами являются информация о поступившем запросе и объект типа HTTPStream.
4. HTTPStream предоставляет API для вывода информации в клиентский браузер.

Это руководство описывает интерфейсы программирования, компиляция, установка и администрирование описанно в [?], аспекты использования `mod_cbroker` с точки зрения архитектуры программных комплексов описанны в [?].

2 Основные объекты

2.1 Информация о запросе

Передается в структуре `RequestInfo`

```
struct RequestInfo
{
  ClientInfo        client_info;
  ServerInfo        server_info;
  ParameterSequence parameters;
  ParameterSequence cookies;
  BinaryParameterSequence binary_parameters;
  string            method;
};
```

Как мы видим, в ней содержится:

- - информация о клиенте:

```
struct ClientInfo
{
  string ip;
  string hostname;
  string user_name;
};
```

где

ip – IP клиентского WWW браузера (или его proxy).

hostname – доменное имя Интернет хоста клиента.

user_name – имя пользователя. Если сервлет не требует авторизации, то передается строка "unknown"

- - информация о WWW сервере. (заметим, что несколько WWW серверов могут обращаться к одному и тому-же CORBA сервлету).

```
struct ServerInfo
{
  string hostname;
  short  port;
};
```

- - информация о текстовых GET и PUT параметрах запроса.

```
struct Parameter
{
  string  name;
  string value;
};
typedef sequence<Parameter>  ParameterSequence;
```

Заметим, что GET и PUT параметры с точки зрения CORBA сервлета неразличимы.

- - информация о cookies
- - информация о двоичных POST и PUT параметров запроса.

```
struct BinaryParameter
{
  string  name;
  CORBA::OctetSeq value;
};
typedef sequence<BinaryParameter>  BinaryParameterSequence;
```

С их помощью можно передавать двоичные данные, организовав, например, загрузку файлов из html формы.

- method - метод запроса, указанный как строка: `"PUT"`, `"GET"`, `"OPTIONS"` и. т. д. Почему му передаем строку, а не определяем перечислимый тип - для возможности расширения функциональности `HTTP` следуя дизайну Apache. Так, например, с помощью `ModCbroker` можно организовать прием `WebDav` запросов.

2.1.1 Информация о именах передаваемых файлов

Отдельного рассмотрения заслуживает случай передачи файла на WWW сервер с помошью WWW формы.

В этом случае в `binary_parameters.name` записывается имя тега `INUT` элемента формы, при помощи которой был загружен файл.

Например, если файл загружен при помощи

```
<INPUT TYPE=FILE NAME="pics">
```

то имя тега - "pics". Сам файл передается в поле `binary_parameters.value`, а полное имя файла можно получить из обычного параметра запроса с именем `name+"_filename"`. В нашем случае это был бы параметр `"pics_filename"`.

2.2 HTTPStream

Этот обьект предоставляет API для вывода в клиентский браузер, фактически экспортируя для этой цели API apache [**?**].

Вот, к примеру, фрагмент кода, выводящий HTML страницу "Hello, World":

```
httpStream.set_content_type("text/html");
httpStream.send_http_header();
httpStream.puts("<HTML><HEAD><TITLE>Hello, World</TITLE><HEAD>"
                "<BODY><CENTER>Hello, World</CENTER></BODY></HTML>");
```

Рассмотрим этот интерфейс подробнее:

```
interface HTTPStream
{
  void set_content_type(in string content_type);
  void set_http_header(in string name, in string value);
  void set_cookie(in string name, in string value,
                  in unsigned long expire_time);

  void send_http_header();
  void send_http_header_ex(in ReplyHeaderInfo header_info);

  string get_http_header(string name);
```

```
  void puts(in string str)
              raises(StreamClosedException);

  unsigned long send_buffer(in CORBA::OctetSeq buffer)
              raises(StreamClosedException);

  void flush()
              raises(StreamClosedException);

};
```

- `void set_content_type(in string content_type);` Эта функция устанавливает тип контента ответа. Ее аргументом должна быть строка MIME типа. (напримех "text/html" или "image/gif")

- `void set_http_header(in string name, in string value);` Эта функция устанавливает переменную заголовка HTTP ответа.

- `set_cookie` – устанавливает cookie, которая будет послана клиенту. Значения параметров очевидны. В следующем запросе сервера эта cookie будет прочитано и передано в параметрах запроса.

- `void send_http_header()` – послать клиенту заголовок ответа. Функции, работающие с заголовкам ответа (`set_content_type`, `set_http_header`, `set_cookie`) должны быть вызваны перед `send_http_header`, а функции, работающие с телом запроса – соответственно после.

- `void send_http_header_ex(in ReplyHeaderInfo reply_info)` – установить переменные заголовка ответа и послать его клиенту. Этот метод является композицией предыдущих четырех и позволяет передать всю информацию, необходимую для генерации заголовка ответа одним вызовом. Структура `ReplyHeaderInfo` выглядит следующим образом:

```
struct ReplyHeadeInfo
{
 string content_type;
 ParameterSequence fields;
 ParameterSequencr cookies;
 unsigned long     cookie_expire_time;
};
```

  Как видим здесь просто собраны тип контента, значения заголовков и cookie-s. Единственное отличие - всем посланным таким способоб cookies присваивается одинаковое время жизни, заданное `cookie_expire_time` (в секундах).

- `string get_http_header() -` возвратить значение заголовка запроса с именем `name`
- `void puts(in string str)` – послать клиенту строку в теле запроса. Эта (как и другие функции работы с телом запроса) функция вызывает StreamClosedException, если по каким-то причинам, соединение клиента с сервером прервалось.
- `unsigned long send_buffer(in sequence<octet> buffer)` – послать клиенту двоичную последовательность байт. Может использоваться для передачи файлов или изображений.
- `void flush()` – форсирует передачу информации, очищая внутренний буфер WWW сервера.

2.3 Handler

Хорошо, теперь мы знаем как пользоваться API вывода, а где его применять? Для этой цели разработчику предоставляется интерфейс `Handler`.

```
interface RequestHandler
{
  void  handle(in RequestInfo request_info, in HTTPStream stream)
                     raises(ExternalException, StreamClosedException,
                            Redirect);
  void  destroy();
};
```

Разработчик должен его реализовать как CORBA объект, и поместить обработку запроса в методе `handle`.

handle – Реализовать обработку запроса.

- request_info – информация о запросе.
- stream – HTML поток, предоставляющий API работы с WWW браузером клиента.

Функция может бросить исключение "ExternalException", если ей необходимо зафикисировать сообщение об ошибке в log файле Apache.

```
exception ExternalException
{
  short http_code;
  string reason;
};
```

Как видим из определения, программист может задать HTTP код возврата и строку, описывающую ситуацию, которая будет напечатана на экране пользователя и в log файле WWW сервера.

Напомним, что исключение StreamClosedException вызывается в API HTTPStream. Оно помещено в декларацию интерфейса для того, чтобы если программист его не обработал при возникновении, то WWW сервер сам произвел бы все необходимые действия.

Исключение `Redirect` используется для перенаправления запроса по другому `URL`, его удобно использовать для интеграции ModCbroker и друших скрипт-приложений.

```
exception Redirect
{
 string url; // url to redirect.
 ParameterSequence parameters; // parameters.
};
```

Реакция Web сервера на такое исключение - генерация GET запроса по указанному URL и с параметрами, указанными как аргумент.

destroy – Деактивировать CORBA объект в соответствии с правилами ORB. Подробнее с разработкой програм в CORBA окружении можно ознакомиться по [?], [?], [?] Также ясно, что если этот обработчик статический, то с ним ничего делать не надо.

Вот, к примеру, Handler для вывода HelloWorld:

```
class HelloWorldHandler: POA_HTTP::RequestHandler
{
   PortableServer::POA_var poa_;
 public:
   HelloWorldHandler(PortableServer::POA_ptr poa)
     :poa_(PortableServer::_duplicate(poa)
   {}

   void handle(const RequestInfo& request_info, HTTPStream_ptr httpStream)
   {
     httpStream.set_content_type("text/html");
     httpStream.send_http_header();
     httpStream.puts("<HTML><HEAD><TITLE>Hello, World</TITLE><HEAD>"
                  "<BODY><CENTER>Hello, World</CENTER></BODY></HTML>");
   }

   void destroy()
   {
     PortableServer::ObjectId_var oid = poa_->servant_to_id(this);
```

```
    poa_->deactivate_object(oid);
    _remove_ref();
  }

};
```

(Тут для простоты предположенно, что мы используем полностью совместимую с CORBA-2.3 ORB)

2.4 Servlet

Теперь рассмотрим интерфейс Servlet (Сервлет). Этот интерфейс тоже должен быть реализован разработчиком конечного приложения; Главные задачи, решаемые этим интерфейсом, следующие:

- Авторизация: сервлет должен решить - запросы к каким обработчикам мы реализируем и как именно?
- Порождение обработчиков. (т.е. в терминах модного нынче языка паттернов, сервлет является фабрикой обработчиков). Заметим, что для случая статического набора обработчиков, можно не возиться с удалением/созданием объектов на уровне языка реализации (т. е. Handler::destroy может ничего не делать).

Смотрим на интерфейс:

```
interface Servlet
{

   RequestHandler create_handler(in string handlerName,
                                 in ClientInfo client_info,
                                 in ServerInfo server_info,
                                 in string passwd)
                                     raises(NoSuchHandlerException,
                                            RequireAuthInfo,
                                            AccessDenied,
                                            ExternalException,
                                            Redirect);

};
```

Метод createHandler совмещает в себе авторизацию и создание обработчика для запроса; впоследствии cbroker вызывет для него метод `handle`, а после этого удалит обработчик. Если username или passwd не проходит авторизации, то возбуждается исключение RequireAuthInfo. Если авторизация прошла неуспешно по другим причинам, то возбуждается исключение AccessDenied.

В качестве примера, напишем сервлет на C++, который запускает наш HelloWorld, в ответ на запрос Обработчика "helllo" только для пользователя с именем "admin" и паролем "qq888ikd930" с хоста 200.200.220.10.

```
class HelloWorldServlet: public POA_HTTP::Servlet
{
 PortableServer::POA_var poa_;
public:

 HelloWorldServlet(PortableServer::POA_ptr poa)
  :poa_(PortableServer::_duplicate(poa))
 {}

 HTTP::RequestHandler_ptr create_handler(in string handlerName
                                         const HTTP::ClientInfo& clientInfo,
                                         const HTTP::ServerInfo& serverInfo,
                                         const char* passwd)
 {
  if (strcmp(handlerName,"hello")!=0) throw NoSuchHandlerException();

  // разрешаем только admin-у с хоста 200.200.220.10 и с паролем qq888ikd930
  if (strcmp(clientInfo.user_name,"admin")!=0) throw RequireAuthInfo();
  if (strcmp(clientInfo.ip,"200.200.220.10")!=0) throw AccessDenied();
  if (strcmp(passwd,"qq888ikd930")!=0) throw RequireAuthInfo();
  if (strcmp(handlerName,"hello")!=0) throw NoSuchHandlerException();

  HelloWorldHandler* handler_impl = new HelloWorldHandler(poa_);
  HTTP::RequestHandler_var retval = handler_impl->_this();
  return retval._retn();
 }

};
```

Или напишем сервлет, который не генерирует обработчиков, но перенаправляет пользователя на свою персонализированную страницу:

```
class CheckServlet: public POA_HTTP::Servlet
{
 UAKGQuery2::QueryManager_var queryManager_;
 public:

   CheckServlet(UAKGQuery2::QueryManager_ptr queryManager)
    :queryManager(UAKGQuery2::QueryManager::_duplicate(queryManager))
   {}

   HTTP::RequestHandler_ptr createHandler(in string handlerName,
                                         const HTTP::ClientInfo& clientInfo,
```

```
                                        const HTTP::ServerInfo& serverInfo,
                                        const char* passwd)
   {
    if (!strcmp(clientInfo.username,"unknown")) {
      throw RequireAuthInfo();
    }
    UAKQUery2::RecordSet_var params=UAKGQuery2::RecordSet::create(2,1);
    params.setColumnName(0,"login");
    params.setColumnName(1,"passwd");
    params.setString(0,0,clientInfo.username.in());
    params.setString(0,1,passwd);
    UAKQUery2::RecordSet_var rs = queryManager_->evaluate(
                                   "select home_url from users where "
                                             "login = :login "
                                             " and "
                                             "password = :password "
                                  );
    if (rs.getNRows()==0) {
           throw AccessDenied();
    }
    String_var home_url=rs.getAsString(0,0);
    Redirect redirect;
    redirect.url=CORBA::string_dup(home_url.in());
    throw Redirect();
   }

};
```

3 Собираем все вместе

Последний вопрос: допустим мы все реализовали, как `cbroker` найдет сервлет и имя обработчика по URL?

Ответ: С помощью CORBA Name Service [?]. Т. е. если path часть URL запроса выглядит следующим образом:

$$http://host.com/cbroker-dir/ServletName/HandlerName$$

(тут cbroker-dir – имя виртуальной директории, устанавливаемой в конфигурации WWW сервера), то `ModCBroker` будет искать сервлет в установленном по умолчанию NamingService по имени HTTPServ/ServletName.

Соответственно, разработчик должен при инициализации сервлета связать его с именем HTTPServ/ServletName. Примеры можно найти в директории demo дистрибутива `ModCBroker`.

Наконец, напомним что программма, использующая cbroker API должна быть скомпилированна с клиентской библиотекой (clcbroker для C++, или

clcbroker.jar) для Java, и что процесс настройки www сервера для работы с `ModCBroker` описан в [?]

4 Встраивание сервлетов в Web приложения

В ряде случаев нам хочется отделить генерирацию контента от выбора содержания: скажем некоторые части web приложения (критичные по времени либо связанны с back-end функциональностью) должны быть оформлены как CORBA сервлеты, но с другой стороны сам Web интерфейс контролируется дизайнером сайта.

Такое разделение контента и содержания может быть реализовано как со стороны сервлета (CASL предоставляет похожую функциональность) так и со стороны Apache. На практике оказалось, что последнее также достаточно удобно.

В `cbroker` встроенна поддержка такой возможности в виде фильтра и в виде API взаимодействия со скрипт языками.

Использование фильтров позволяет встраивать вызовы сервлетов ModCbroker в Web страницы: выражение вида

```
<?cbroker ServletName HandlerName p1="v1" ... p2="vN" ?>
```

заменяется на результат вызова обработчика `handlerName` сервлета `ServletName` с параметрами p_i, v_i. В структуре `RequestInfo` метод устанавливается в строку `'CBROKER-INCLUDE'`

Для этого достаточно просто настроить обработку текста HTML фильтром `Cbroker`.

К примеру, пусть у нас есть сервлет, выводящий в окно браузера список внутренних телефонов работников предприятия, который называется, скажем: `NSI/emp`.

В традиционной технологии программирования, при каждой смене дизайна сайта надо обращаться к программисту, чтобы он модифицировал соответствующий сервлет. С появлением `ModCbroker-3.1.0` эта необходимость отпала, теперь мы:

- Пишем файл с расширением, скажем `cbhtml` и содержанием:

  ```
  <HTML>
  <HEAD> <TITLE> Список работников </TITLE> </HEAD>
  <BODY>
   <img src="images/OurGreatImageOnTop.gif" /><br></br>
    <?cbroker NSI Emp ?>
   <img src="images/OurGreatImageOnBottom.gif" /><br></br>
  </BODY>
  ```

- Указываем Apache применять этот фильтр к файлам с расширением `cbhtml`, добавляя следующую команду в конфигурацию Apache:

```
AddOutputFilterByType CBROKER cbhtml
```

- Модифицируем программу так, чтобы она выводила только данные.

Все, теперь наш программист может заниматься только семантикой, а дизайнер - дизайном.

4.1 Передача параметров

Теперь, допустим у нас еще есть обработчик, который ищет человека по фамилии и выводит подробную информацию. Скажем, `NSI/emp_search`. К примеру, на домашней странице пользователя scott[1]tiger есть следующий блок:

```
<?cbroker NSI emp_search name="jon" family="scott" ?>
```

Это хорошо, если у нас страница статичная, а если мы хотим чтобы имя и фамилия человека, которого надо искать, передавадась в параметрах запроса?

Для этого есть специальный параметр: `passRequestParameters`.

```
<?cbroker NSI emp_search passRequestParameters="1" ?>
```

Можно одновременно передавать параметры из текста и из запроса.

```
<?cbroker NSI emp_search family="scott" passRequestParameters="1" ?>
```

В этом случае, если имя какого-то параметра в запросе совпадает с именем параметра в строке вызова то приоритет отдается второму. Почему так? Чтобы мы могли писать в тексте `html` тексты SQL выражений и не бояться их подмены.

Кстати, для параметров запроса существует несколько разных способов приема:

- `passRequestParameters` - обработать параметры запроса, независимо от метода.
- `passQueryArgs` - передать параметры GET запроса.
- `passPostAndPutParameters` - использовать параметры, передающиеся в потоке `POST` и `PUT` методов.

4.2 Комбинирование фильтров

Фильтры можно комбинировать друг с другом. Например, использовать `mod_include` для генерации заголовков и окончаний страниц:

- Добавить для типа scbhtml комбинацию cbroker и include, соответствующая директива конфигурации Apache:

[1] /

```
AddOutputFilter CBROKER;INCLUDE .scbhtml
```

Тогда наш файл будет сначала обрабатываться с помощью CBROKER, а потом резудьтат этого преобразования будет обработан фильтром INCLUDE.

- Преобразовать наш файл следующим образом:

```
<HTML>
<HEAD> <TITLE> Список работников </TITLE> </HEAD>
<BODY>
 <--#include virtual "/topOfOurPage/" --><br></br>
  <?cbroker NSI Emp ?>
 <--#include virtual "/bottomOfOurPage/" --><br></br>
</BODY>
```

Точно так же можно комбинировать использование CBROKER и XSLT или PHP. Впрочем, для PHP лучше использовать специальный интерфейс работы со скриптами так как комбинация простой подстановки текста и языка программирования выглядит не очень естественно.

4.3 Авторизация

При выводе html страниц с включениями тегов cbroker Вам необходимо отдельно реализовать авторизацию: ведь сервер должен знать имя пользователя до начала вывода страницы.

Для этой цели предназнаячена директива CbrokerAddAuthByLocation, которая позволяет задать авторизирующий сервлет для определенной директории. К примеру:

```
CbrokerAddAuthByLocation  /cbf /Billing /auth
```

Это означает, что доступ ко всем файлам расположенным в `/cbf` (и в подкаталогах) регулируется с помощью сервлета `Billing`, вызывая там обработчик `auth`. Протокол взаимодействия очены простой - сервлет при запросе авторизации ведет себя точно так-же, как и при любом другом запросе - возвращает значение либо генерирует исключительную ситуацию. Единственное отличие - нас не интересует возвращаемое значение, т. е. мы абсолютно спокойно можем возвратить `NIL`. К примеру, посмотрим на следуйщий блок кода:

```
class HelloWorldServlet: public POA_HTTP::Servlet
{
 PortableServer::POA_var poa_;
public:
 HelloWorldServlet(PortableServer::POA_ptr poa)
  :poa_(PortableServer::_duplicate(poa))
```

```
 {}

 HTTP::RequestHandler_ptr create_handler(in string handlerName
                                         const HTTP::ClientInfo& clientInfo,
                                         const HTTP::ServerInfo& serverInfo,
                                         const char* passwd)
 {
  if (strcmp(handlerName,"auth")==0) {
    // разрешаем только admin-у с хоста 200.200.220.10 и с паролем qq888ikd930
    if (strcmp(clientInfo.user_name,"admin")!=0) throw RequireAuthInfo();
    if (strcmp(clientInfo.ip,"200.200.220.10")!=0) throw AccessDenied();
    if (strcmp(passwd,"qq888ikd930")!=0) throw RequireAuthInfo();

    return HTTP::RequestHandler::_nil();
  }
 }

};
```

Естественно, в конфигурации Apache также надо задать способ авторизации для соответствующего местоположения. Перед использованием директивы CbrokerAddAuthByLocation, пожалуйста, прочитайте соответствующий раздел в руководстве администратора.

5 API взаимодействия со скрипт языками

5.1 WebSh

Вы можете вызывать сервлеты `ModCbroker` из скриптов websh [?]. Синтаксис вызова:

```
cbroker servletName handlerName {param-list}
```

В параметрах должны быть последовательности пар имя/значение.
Например:

```
[cbroker Billing AccountState { username $username userid $userid }]
```

Для этого в конфигурации websh скрипта Вам нужно загрузить библиотеку webshcbroker.

Пример вызова нашего `Hello, World` из первой главы:

```
#
# Demo for interactuion of ModCbroker with Tcl
#
web::initializer{
  # load cbroker interaction library
  load /usr/local/lib/libweshcbroker83.so.1
```

```
}

web::command default {

 web::put {
   "<HTML><HEAD>  Hello World demo </HEAD>"
   "<BODY>"
   "<center>"
   "    [cbroker Hello World {} ] "
   "</center>"
   "</BODY>"
   "</HTML>"
 }

}

web::dispatch
```

6 API взаимодействия с другими модулями Apache

Этот раздел предназначен для авторов (и соавторов) других модулей Apache. Если вы хотите вызвать сервлет `ModCbroker` из другого модуля (например, для доступа к CORBA объектам из своего любимого скрипт языка), то ModCbroker предоставляет Вам функцию:

```
APR_DECLARE_OPTIONAL_FN(int,ap_cbroker_call,
            (request_r* where,
             const char* servletName,
             const char* handlerName,
             const char*  method,
             apr_table_t* parameters,
             bool useOutputBuffer,
             char** outputBuffer,
             apr_size_t* outputLen
            ))
```

Которую можно вызывать из других модулей. Механизм такого вызова описан в [?].

Именно таким образом реализованно наше API взаимодействия со скрипт-языками.

7 IDL интерфейсы

```
#ifndef __HTTP_IDL
#define __HTTP_IDL
/*
 * Interface for CORBA http servlets.
 * (C) Grad-Soft Ltd, Kiev, Ukraine 2000 - 2003
 * http://www.gradsoft.com.ua
 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>
 *  1999, 2000, 2001, 2002, 2003
 * $Id: ProgrammingGuide.tex,v 1.27 2007-06-14 11:47:14 rssh Exp $
 */


#pragma prefix "gradsoft.kiev.ua"

/**
 * Module for CORBA http servlets.
 **/
module HTTP {

   ///

   ///
   typedef sequence<octet> HTTPOctSeq;
   ///
   typedef sequence<string> HTTPStrSeq;

   /**
    * used for passing binary parameters from
    * users forms
    **/
   struct BinaryParameter
   {
     ///
     string  name;
     ///
     HTTPOctSeq value;
   };
   ///



   /**
    * used for passing parameters from
    * users forms
```

```
 **/
struct Parameter
{
  ///
  string  name;
  ///
  string value;
};


///
typedef sequence<Parameter>  ParameterSequence;
///
typedef sequence<BinaryParameter>  BinaryParameterSequence;

/**
 * what we know about client:
 **/
struct ClientInfo
{
  /**
   * IP adress of client
   **/
  string ip;
  /**
   * hostname of client
   **/
  string hostname;
  /**
   * user name of client.
   * (or 'unknown', if handler does not require authorization)
   **/
  string user_name;
};

/**
 * what we know about web server
 **/
struct ServerInfo
{
  ///
  string hostname;
  ///
  short  port;
};
```

```
/**
 * this struct describe user request
 **/
struct RequestInfo
{
  ///
  ClientInfo        client_info;
  ///
  ServerInfo        server_info;
  ///
  ParameterSequence parameters;
  ///
  ParameterSequence cookies;
  ///
  BinaryParameterSequence binary_parameters;
  ///
  string            method;
};


/**
 *
 **/
struct ReplyHeaderInfo
{
  string            content_type;
  ParameterSequence fields;
  ParameterSequence cookies;
  unsigned long     cookie_expire_time;
};

/**
 * Can be raised by user servlets code.
 **/
exception ExternalException
{
  /// http code, which we want return to client
  short http_code;
  /// string description of error.
  string reason;
};

/**
 * raised by web server, when user output stream is cancelled
 **/
exception StreamClosedException {};
```

```
/**
 * raised when create_handler require authorization.
 **/
exception RequireAuthInfo {};

/**
 * raised by createHandler(), when ClientInfo::ip or ClientInfo::hostname
 * or ServerInfo::hostname or ServerInfo::port have no access to this
 * servlet or handler
 **/
exception AccessDenied {};

/**
 * raised in createHandler() or in handle() to redirect request
 **/
exception Redirect {
    // url to redirect
    string url;

    // parameters for uri
    ParameterSequence parameters;
};

/**
 * Servel comunicate with web server (and users web browser)
 * via this interface, during handling of request.
 **/
interface HTTPStream
{
  ///

  /**
   * set content type of servlet output.
   *@precondition
   *  it must be done before call of send_http_header
   **/
  void set_content_type(in string content_type);

  /**
   * set additional http header name-value pair
   *@precondition
   *  it must be done before call of send_http_header
   **/
  void set_http_header(in string name, in string value);
```

```
  /**
   * get http header value for requested name
   * return value of http header in request.
   **/
  string get_http_header(in string name);

  /**
   * set cookie in output client header.
   *@precondition
   *  it must be done before call of send_http_header
   **/
  void set_cookie(in string name, in string value,
                  in unsigned long expire_time );

  ///
  void send_http_header();

  ///
  void send_http_header_ex(in ReplyHeaderInfo header_info);

  /**
   * send string to user browser
   **/
  void puts(in string str)
                raises(StreamClosedException);

  /**
   * send sequence of octet to user browser.
   **/
  unsigned long send_buffer(in HTTPOctSeq buffer)
                raises(StreamClosedException);

  /**
   * flush output stream.
   **/
  void flush()
                raises(StreamClosedException);

};

/**
 * Application programmer must provide implementation of this
 * interface.
 **/
interface RequestHandler
```

```
{

  ///

  /**
   * handle request and output results to HTTPStream
   **/
  void  handle(in RequestInfo request_info, in HTTPStream stream)
              raises(ExternalException, StreamClosedException, Redirect);

  /**
   * destroy request handler. This method is
   * called by servlet engine after processing of request.
   * programmer must not call this method directly.
   **/
  void  destroy();
};

/**
 * throwed by servlet when we try to call unexistent handler
 **/
exception NoSuchHandlerException {};


/**
 * Application programmer must provide implementation
 * for this interface and bind it to "HTTPServ/Name" in
 * Naming Service
 **/
interface Servlet
{
   /**
    * create handler for request.
    *@exception NoSuchHandlerException when we try to call
    *            unexistent handler
    *@exception RequireAuthInfo  when Servelt require filled client name
    *            and client password structure, but it is not passed with
    *            this call (and ModCbroker popup auth window on next step)
    *@exception AccessDenied when access for this user is denied
    *@exception ExternalException can be throwed by servlet.
    **/
   RequestHandler create_handler(in string handler_name,
                                 in ClientInfo client_info,
                                 in ServerInfo server_info,
                                 in string passwd)
                                    raises(NoSuchHandlerException,
```

```
                                                RequireAuthInfo,
                                                AccessDenied,
                                                ExternalException,
                                                Redirect
                                               );
    };


};

#endif
```

8 Изменения

- 13-06-2007 адаптирован для `mod_cbroker-3.2.0`
- 24-01-2003 адаптирован для `mod_cbroker-3.1.0`
- 21-05-2002 адаптирован для `mod_cbroker-3.0.0`
- 26-02-2002 адаптирован для `mod_cbroker-2.0.5`
- 16-01-2002 просмотр.
- 16-08-2001 добавлен подраздел о загрузке файлов.
- 24-04-2001 адаптирован для `mod_cbroker-2.x`
- 17-03-2001 просмотр.
- 30-01-2000 добавлены формальные атрибуты эксплуатационной документации.
- 12-05-2000 создан.

References

[1] Ryan Bloom. Apache modules. *ONLAMP*, 2001. http://www.onlamp.com/pub/a/apache/2001/09/27/apache_2.html.

[2] Ruslan Shevchenko. Anatoliy Doroshenko. A method of mediators for building web interfaces of corba distributed enterprose applications. *Lecture Notes in Informatics V. 4 - Proceeding of Information Systems Technology and its Applications 2001*, 2001. ISBN 3-88579-331-8.

[3] Object Management Group, editor. *Common Object Services Specification*, chapter 3. Naming. OMG, 1997. formal/97-12-10.

[4] Object Management Group, editor. *The Common Object Request Broker: Architecture & Specification*, chapter 11. Portable Object Adapter. OMG, 1999. formal/99-10-07.

[5] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999. ISBN 0201379279.

[6] Ben Laurie and Peter Laurie. *Apache. The Definitive Guide*. O'Reilly, 1997. ISBN 1-56592-250-6.

[7] Apache Software Foundation NetCetera AG. *websh home page*, 2000-2002. http://tcl.apache.org/websh.

[8] Object Oriented Concepts, inc. *ORBacus for C++ and Java*.

[9] Julia Prokopenko Ruslan Shevchenko, Alexandr Yanovec. *mod_cbroker: Administration Guide*, 2000-2001. GradSoft-CBroker-e-AG503-79-10.05.2000-2.0.1.