Сервер приложений C++

полное руководство

Оглавление

Введение

Попытки создания удобных и одновременно высокопроизводительных веб-инструментариев регулярно предпринимаются еще со времен использования первых CGI-сценариев. К сожалению, из-за высокой сложности разработки подобных систем лишь немногие их них доводятся до состояния, пригодного для коммерческого использования.

Прототип CAS, проект Bartertown, был разработан в ноябре 2003 года в компании “НетБридж Сервисез” (Mail.ru). Представлял он собой монолитное приложение, для каждой модификации требующее перекомпиляции кода проекта. Тем не менее, в нем уже присутствовали все основные компоненты современного сервера приложений: класс представления (библиотека CTPP версий 1.X), классы моделей, реализующие конкретный интерфейс, наследованный от абстрактного класса и контроллер в виде массива объектов, содержащих имя обрабатываемого URL и набора исполняемых объектов-моделей.

Очевидные недостатки подобной схемы, а именно: требование пересборки всего проекта при изменении любого класса модели, негибкая организация объекта контроллера и единственный возможный шаблонизатор продиктовали необходимость разработки новой современной удобной версии.

Дальнейшее развитие этой системы привело к созданию сервера приложений C++ (C++ Application Server, CAS) – инструментария, лишенного недостатков предыдущей версии, обладающего широкими возможностями конфигурирования и расширения функционала и в то же время способного обрабатывать сотни и тысячи запросов в секунду.

Сейчас CAS – полноценный сервер приложений, имеющий в своей основе архитектуру MVC и поддерживающий ее дополнительные расширения. При проектировании и разработке этого продукта особое внимание уделялось удобству работы программистов с его API и простоте администрирования в эксплуатации.

Конфигурация CAS хранится в виде XML и допускает возможность задать большинство параметров “умолчанию”, а наиболее часто используемые секции конфигурации вынести в отдельные файлы. Выбор XML как языка описания конфигурации позволяет пользоваться стандартными утилитами для проверки их синтаксиса и редактирования.

Классы моделей-обработчиков и других функциональных объектов можно загружать как виде отдельных модулей, так и как монолитные библиотеки. В случае необходимости возможна разработка собственных вариантов классов контроллера и представления. Таким образом, функциональность CAS можно изменять практически произвольно.

В этой книге рассказано как использовать сервер приложений для разработки сложных высоконагруженных вебпроектов на языке C++. Информация сгруппирована по принципу “от простого – к сложному”, фактически, являясь учебником по применению технологии CAS.

Книга содержит шесть глав, посвященных следующим темам:

- архитектуре CAS
- разработке простейшего класса “Hello, World”
- списку компонент сервера и их предназначению
- установке и первоначальной настройке сервера
- работе с внешними ресурсами, такими как подключения к базам данных или системе сессий

- интеграции с популярными вебсерверами

Все приведенные примеры выполнены в едином стиле и проверены в реальных условиях. Если какой-либо пример кажется непонятным, достаточно изучить предыдущий. Если не помогло и это, можно задать вопрос в списке рассылки опытным пользователям или непосредственно разработчикам. Контактная информация, а также информация о списках рассылки и официальных сайтах размещена в конце книги.

Архитектура CAS и парадигма MVC

Сервер приложений CAS построен по многокомпонентной схеме, отделяющей сущности модели MVC друг от друга. Каждая из сущностей представляет собой отдельный загружаемый модуль и может быть заменена независимо от остальных.

Классическая модель MVC выглядит следующим образом:

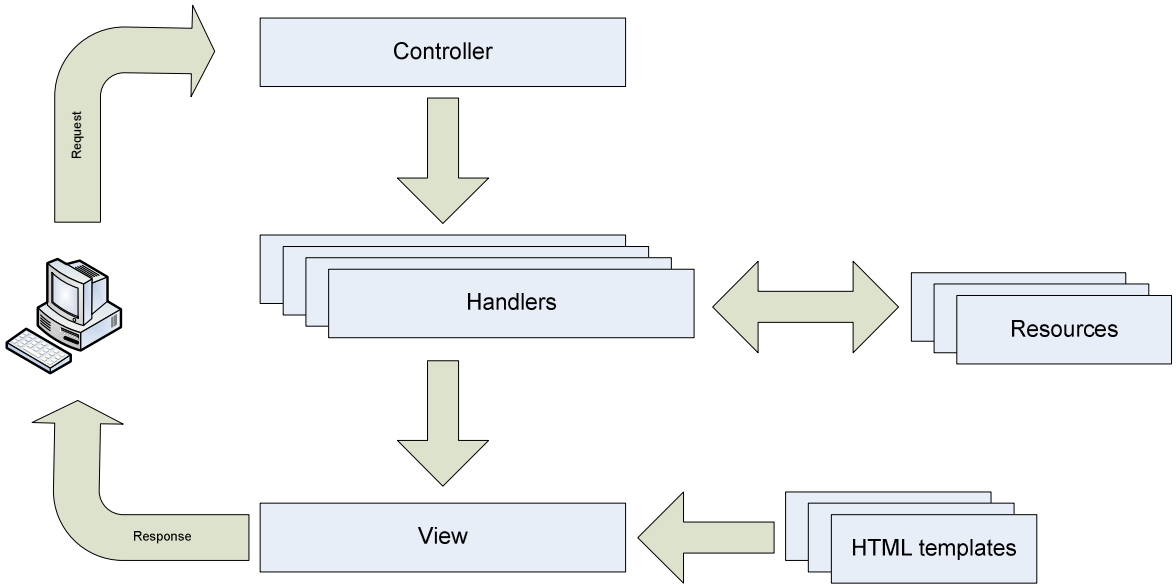


Рис. 1. Общая схема архитектуры MVC

Пользователь отправляет запрос к серверу приложений. Контроллер выбирает необходимый набор моделей и, в зависимости от конфигурации, последовательно или параллельно исполняет их. Модели, взаимодействуя с внешними ресурсами, например, с различными (Р)СУБД создают набор данных, передаваемых в объект представления. Представление формирует требуемый ответ на основе переданного набора данных и загруженных шаблонов страниц и отправляет его пользователю.

В зависимости от конфигурации, для разных URL, обрабатываемых сервером, могут быть использованы различные классы контроллеров, моделей и представления, что дает возможность сконфигурировать CAS для выполнения задач разных типов. К примеру, используя один и тот же сервер можно одновременно построить портал для посетителей, пользующихся как обычными обозревателями (Microsoft Internet Explorer, Firefox или Opera), так и WAP-обозревателями из мобильных телефонов.

Архитектуру сервера приложений проще всего описать на примере использования CAS как модуля вебсервера Apache. Реализации CAS, поддерживающие интерфейсы FastCGI и CGI имеют несущественные архитектурные различия.

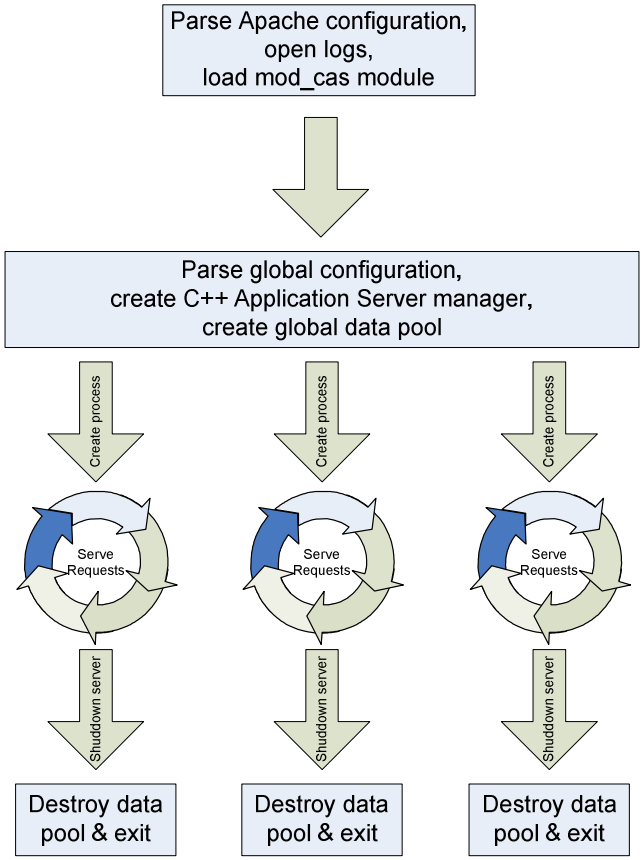


Рис. 2. Схема загрузки и инициализации CAS.

При запуске Apache происходит загрузка CAS и его инициализация. После успешной инициализации, включающей в себя чтение и обработку файла глобальной конфигурации, загрузку и инициализацию модулей, чтение и инициализацию конфигураций виртуальных серверов, Apache создает пул процессов или нитей, обрабатывающих запросы от пользователей. До тех пор, пока работа CAS не окончена, в цикле производится обработка запросов и выдача результатов пользователям. По окончании работы модули выгружаются, освобождается использованная память и процессы или нити завершаются.

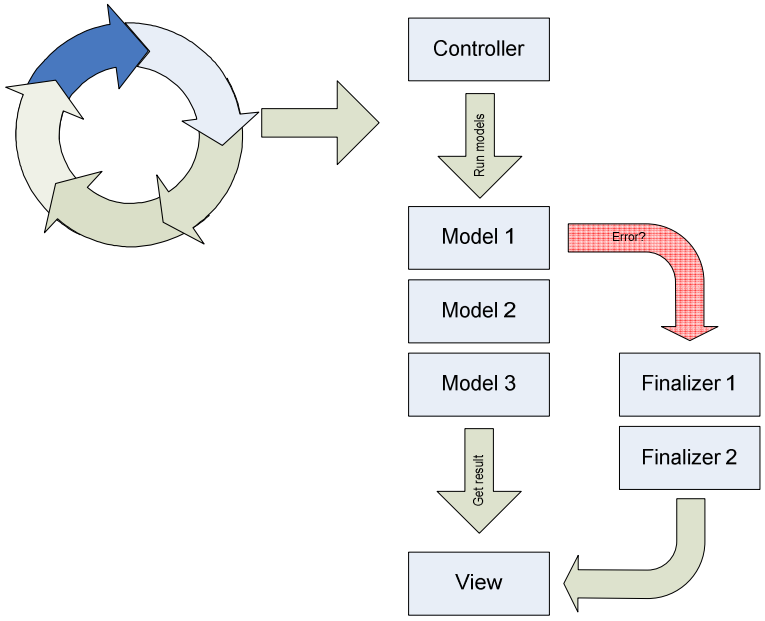


Рис. 3. Цикл обработки запроса

Обработка каждого запроса начинается с проверки URL. Если URL не обрабатывается сервером, управление возвращается без каких-либо действий со стороны CAS. Если URL обрабатывается, то производится поиск локации, ответственной за обработку URL. Для найденной локации выбираются объекты, подлежащие исполнению: контроллер, список моделей, и представление.

Контроллер, если присутствует в описании локации, исполняется первым. Основное предназначение контроллера – изменение конфигурации локации в зависимости от полученных от пользователя данных. Например, если язык страницы зависит от географического положения пользователя, контроллер может вычислить язык по IP-адресу клиента.

Объекты *моделей* вызываются последовательно, в порядке, указанном в файле конфигурации и выполняют работу по получению требуемого набора данных для отображения запроса. При возникновении ошибки выполнение моделей прекращается, и запускается исполнение цепочки финализаторов.

Финализатор – объект, назначение которого состоит в коррекции возникшей в моделе ошибки. В случае возникновения ошибки в объекте модели, финализатор может предпринять определенные действия для ее исправления или, если ошибку исправить нельзя, вывести диагностическое сообщение пользователю.

Визуализация результатов производится в объекте представления. *Представление* производит наложение данных на шаблон и генерирует требуемый документ. Представление всегда выполняется последним.

Кратко остановимся на понятии локации. В идеологии CAS, *локация* – единица конфигурации сервера, включающая в себя список обрабатываемых URL, описание используемого контроллера, моделей, финализаторов и представления.

Каждая локация имеет уникальное имя. Выбор локации производится по совпадению запрашиваемого URL с URL, обрабатываемых локацией. Возможно как полное совпадение, так и совпадение с заданным регулярным выражением.

Конфигурация и интерфейсы объектов сервера CAS описаны в следующей главе.

Простейшее приложение CAS

При первом знакомстве с CAS мы не будем строить чего-либо сложного, а остановимся на широко известной новичкам программе “Hello, World!”.

Чтобы разработать ее, нам потребуется:

- любой компьютер под управлением FreeBSD, Linux или Solaris
- установленный сервер приложений
- компилятор C++
- программа cmake
- программа make
- 15 минут свободного времени

Первое, что следует сделать – создать проект модуля CAS. Для этого служит программа **`cas-xt`** – CAS eXtension Tool. Синтаксис ее ключей во многом повторяет синтаксис широкоизвестной программы apxs. Более подробную справку о **`cas-xt`** вы можете получить в главе, посвященной установке и настройке сервера приложений.

Чтобы создать модуль, необходимо подать следующую команду:

```
cas-xt -t handler -g -n Hello

Using templates from directory "/usr/local/share/cas/xt"
Output directory is ""
Creating [DIR]  Hello
Creating [DIR]  Hello/include
Creating [DIR]  Hello/src
Creating [FILE] Hello/src/Hello.cpp
Creating [FILE] Hello/CMakeLists.txt
```

Смысл ключей следующий:

- **`-t`** задает тип создаваемого ресурса, в нашем случае это **“handler”**
- **`-g`** указывает о необходимости сгенерировать набор шаблонов
- **`-n`** указывает имя создаваемого модуля, у нас – **“Hello”**

Если все прошло успешно, будет создан каталог **`Hello`**, а в нем – два файла: **`CmakeLists.txt`** и **`Hello.cpp`**.

Все, что следует сделать – это открыть на редактирование файл **`Hello.cpp`**. и вписать в метод **`Handler`** вывод строки **“Hello, World”**:

```
//
// Initialize handler
//
INT_32 Hello::Handler(CTPP::CDT   & oData,
                      ASRequest   & oRequest,
                      ASResponse  & oResponse,
                      ASPool      & oGlobalPool,
                      ASPool      & oVhostPool,
                      ASPool      & oRequestPool,
                      CTPP::CDT   & oLocationConfig,
```

```
                        CTPP::CDT   & oIMC,
                        ASLogger    & oLogger)
{
     DEBUG_HELPER(&oLogger, "Hello::Handler");

     // Put your code here
     oData["hello"] = "Hello, World!";

     // 200 OK
     oResponse.SetHTTPCode(200);

     // Set HTTP header
     oResponse.SetHeader("X-Module", "Hello");

return HANDLER_OK;
}
```

Для сборки модуля следует перейти в каталог **Hello** и выполнить команду **cmake . && make install**:

```
cd Hello
cmake . && make install

-- The C compiler identification is GNU
......
-- Configuring done
-- Generating done
-- Build files have been written to: /home/stellar/Hello
......
Scanning dependencies of target mod_hello
......
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/libexec/cas/mod_hello.so
```

После автоматической сборки и установки нам потребуется создать шаблон для вывода данных и отредактировать файлы конфигурации.

Шаблон генерируемой страницы, который необходимо будет сохранить в файле **/home/www/example.com/tmpl/login.tmpl**, будет очень простым:

```
<html>
<head>
     <title>My first example</title>
</head>
<body>
     <TMPL_var hello>
</body>
</html>
```

Теперь осталось только настроить CAS для обработки HTTP запросов.

CAS использует два файла конфигурации: глобальный и конфигурации виртуального сервера. Глобальный файл, **global-config.xml**, в зависимости от операционной системы, обычно размещается в каталогах **/etc/cas**, **/usr/local/etc/cas** или **/opt/CASWserver/conf**.

Чтобы добавить модуль, в глобальный файл конфигурации, в секцию **“Modules”** следует внести строку с описанием имени модуля, его типа и файла динамической библиотеки, в которой он содержится:

```
<Modules>
     ....
     <Module Name="Hello" Library="mod_hello.so"
     ModuleType="Handler"/>
     ....
</Modules>
```

Следующий шаг – указать, как и где будет использоваться подключенный нами обработчик. Делается это путем изменения файла конфигурации виртуального сервера CAS. Его имя задается в конфигурации сервера Apache директивой **CASConfigFile**. По умолчанию сервер приложений отключен, поэтому чтобы разрешить работу CAS с указанным виртуальным сервером Apache, требуется также выставить флаг **CASEnable** в положение **On**. Пример для виртуального сервера дан ниже:

```
<VirtualHost *:80>
     ServerName    example.com
     CASEnable     On
     CASConfigFile /home/www/example.com/example.xml
     DocumentRoot  /home/www/example.com
</VirtualHost>
```

В файле **/home/www/example.com/example.xml** в секции **“Locations”** указываем URL и имя модуля:

```
<Locations>
     ....
     <Location name="HelloExample">
          <URIList>
               <URI type="plain">/hello.html</URI>
          <URIList>
          <Templates>
               <Template type="plain">/home/www/example.com
/tmpl/login.tmpl</Template>
          </Templates>
          <Handlers>
               <Handler name="Hello"/>
          </Handlers>
          <View>
               <Handler name="CTPP2View"/>
```

```
        </View>
    </Location>
    ....
</Locations>
```

Проверить правильность настройки сервера можно при помощи команд **cas-globalconf** и **cas-hostconf**. Для файла глобальной конфигурации вывод должен быть следующим:

```
cas-globalconf /etc/cas/global-config.xml

  Libexec dirs:
      .
      /usr/lib/cas
      /usr/local/libexec/cas
      /opt/REKIWcas/libexec

  Modules:
      Name:         Hello
      Type:         Handler
      Library file: /usr/local/libexec/cas/mod_hello.so

      Name:         CTPP2View
      Type:         View
      Library file: /usr/local/libexec/cas/mod_ctpp2_view.so
      Configuration:
                  LibexecDirs =>
                        LibexecDir =>
                              0 : "."
                              1 : "/usr/local/libexec/ctpp"
                              2 : "/usr/libexec/ctpp"
                              3 : "/opt/REKIWcas/libexec"
                  MaxFunctions => "1024"
```

Для файла конфигурации виртуального сервера:

```
cas-hostconf /home/www/example.com/tmpl/login.tmpl

  Server name:           www.example.com
  Temp. files directory: /tmp
  Template include directories:
      /home/ashetuhin
      /var/www/cas/tmpl
      /usr/share/cas/tmpl
      /usr/local/share/cas/tmpl

  Per-server modules configuration:

  Locations:
      Name: Hello
```

```
      Default HTTP Response code: 200
      Default HTTP content type:  text/html
      URI(s):
          /hello.html

      Handlers(s):
          Hello

      View: CTPP2View
```

Если ошибок нет, можно перезапустить Apache.

```
apachectl restart

/usr/local/sbin/apachectl restart: httpd restarted
```

Вот и все. Настало время насладиться результатами работы:

```
lynx -mime_header http://localhost/hello.html

HTTP/1.1 200 OK
Date: Fri, 27 Nov 2009 15:35:23 GMT
Server: Apache/2.2.13 (FreeBSD) DAV/2 mod_cas/3.2.1(Flora)
X-Powered-By: C++ Application Server v3.2.1(Flora)
X-Module: Hello
Connection: close
Content-Type: text/html

<html>
<head>
        <title>My first example</title>
</head>
<body>
        Hello, World!
</body>
</html>
```

Как мы видим, разработка модулей для CAS не представляет особой сложности: для реализации базового функционала достаточно выполнения нескольких простых действий. Разумеется, для построения больших и сложных систем, безусловно, потребуются глубокие знания архитектуры и реализации CAS.

В следующих главах книги мы постараемся досконально разобраться во всех возникших вопросах и детально рассмотрим архитектуру и API сервера приложений, а также – процессы сборки, установки и развертывания CAS.

Компоненты сервера

Как уже было описано в предыдущей главе, основные классы сервера приложений повторяют функционал компонент, присутствующих в архитектуре MVC. Однако, классы моделей, представления и контроллера – не единственные объекты, которые можно использовать в CAS: кроме них существует множество других классов, которые может быть полезно использовать при разработке под CAS.

Для понимания архитектуры CAS следует начать изучение с основ построения API.

Объект

Базовым классом любого приложения сервера является класс **`ASObject`**. Он предназначен для реализации загрузки классов приложений из динамических библиотек посредством универсального загрузчика – класса **`ASObjectLoader`**. К классу **`ASObject`** восходят все возможные пользовательские объекты: модули, обработчики, финализаторы, контроллеры, ресурсы и представления.

```
//
// Базовый объект
//
class ASObject
{
public:
	// Тип объекта
	virtual CCHAR_P GetObjectType() const = 0;

	// Имя объекта
	virtual CCHAR_P GetObjectName() const = 0;

	// Виртуальный деструктор
	virtual ~ASObject() throw() { ;; }
};
```

Класс имеет два чисто виртуальных метода: **`GetObjectType`** и **`GetObjectName`**. первый метод возвращает тип объекта, второй – human-readable имя. Имя должно быть уникальным, поскольку в CAS доступ ко всем объектам ведется через методы типа **`GetObjectByName`**, а сами объекты хранятся в виде словаря "ключ => значение". Тип объекта необходим для дополнительного контроля правильности выборки объекта из фабрики.

Модуль.

Класс **`ASModule`** – следующий по иерархии объект сервера. Предназначен для подготовки загруженных объектов к работе.
Каждое приложение (контроллер, модель, представление) является загружаемым из разделяемой библиотеки объектом. Класс модуля имеет четыре метода-зацепки (hook-methods), исполняемых при инициализации модуля, старте сервера, окончании работы сервера и выгрузке модуля. Следует отметить, что на некоторых архитектурах реальной выгрузки модуля не происходит, тем не менее, метод вызывается всегда.

```
//
// Модуль
//
```

```
class ASModule:
  public ASObject
{
public:
      // Инициализация модуля

      virtual INT_32 InitModule(CTPP::CDT  & oConfiguration,
                                ASPool     & oModulesPool,
                                ASPool     & oGlobalPool,
                                ASLogger   & oLogger);

      // Инициализация сервера
      virtual INT_32 InitServer(CTPP::CDT  & oConfiguration,
                                ASPool     & oGlobalPool,
                                ASPool     & oServerPool,
                                ASLogger   & oLogger);

      // Останов сервера
      virtual INT_32 ShutdownServer(CTPP::CDT  & oConfiguration,
                                    ASPool     & oGlobalPool,
                                    ASPool     & oServerPool,
                                    ASLogger   & oLogger);

      // Выгрузка модуля
      virtual INT_32 ShutdownModule(CTPP::CDT  & oConfiguration,
                                    ASPool     & oModulesPool,
                                    ASPool     & oGlobalPool,
                                    ASLogger   & oLogger);

      // Получение типа модуля
      virtual CCHAR_P GetModuleType() const = 0;

      // Виртуальный деструктор
      virtual ~ASModule() throw() { ;; }
};
```

Обязательным к реализации является только метод **GetModuleType**. Все другие методы могут использоваться по необходимости, не загромождая собой без необходимости код проекта.

Модель-Контроллер-Представление

В сервере приложений архитектура MVC представлена двумя базовыми классами: классом **ASHandler**, к которому восходит контроллер и все модели и классом **ASView**, использующимся для реализации классов-представлений.
Единовременно в каждой локации может быть описано произвольное количество моделей, не более одного контроллера и одно представление. Иными словами, для каждой локации объект представления является обязательным, а контроллер и модели – нет.
Разумеется, для каждой локации можно указать свой контроллер и представление, никаких ограничений на использование разных классов MVC в рамках одного проекта нет.

Класс представления

Класс представления **ASView** предназначен для формирования HTTP-ответа пользователю. Является потомком класса **ASModule**, и предоставляет два дополнительных метода: зацепку для инициализации локации и метод вывода данных.

```
//
// Представление объект
```

```
//
class ASView:
  public ASModule
{
public:

    // Инициализация локации
    virtual INT_32 InitLocation(CTPP::CDT  & oLocationConfig,
                                ASLogger   & oLogger);

    // Вывод данных
    virtual INT_32 WriteResponse(STLW::vector<ASTemplate *>  & vTemplates,
                                 CTPP::CDT                   & oData,
                                 ASResponse                  & oResponse,
                                 ASResponseWriter            & pRespWriter,
                                 CTPP::CDT                   & oIMC,
                                 ASLogger                    & oLogger) = 0;

    // Виртуальный деструктор
    virtual ~ASView () throw() { ;; }
};
```

Класс обработчика

Каждое исполняемое сервером приложение строится на основе базового класса приложения – **`ASHandler`**. Под понятием приложения подразумеваются модели и/или контроллер. Класс **`ASHandler`** наследуется от класса модели **`ASModel`** и содержит в себе три дополнительных метода: **`InitLocation`**, **`Handler`** и **`Fixup`**. Обязательным к реализации является только метод **`Handler`**.

```
//
// Обработчик объект
//
class ASHandler:
  public ASObject
{
public:
    // Инициализация локации
    virtual INT_32 InitLocation(CTPP::CDT  & oLocationConfig,
                                ASLogger   & oLogger);

    // Обработчик
    virtual INT_32 Handler(CTPP::CDT   & oData,
                           ASRequest   & oRequest,
                           ASResponse  & oResponse,
                           ASPool      & oGlobalPool,
                           ASPool      & oVhostPool,
                           ASPool      & oRequestPool,
                           CTPP::CDT   & oLocationConfig,
                           CTPP::CDT   & oIMC,
                           ASLogger    & oLogger) = 0;
    // Финализатор
    virtual INT_32 Fixup(ASPool      & oGlobalPool,
                         ASPool      & oVhostPool,
                         ASPool      & oRequestPool,
                         CTPP::CDT   & oLocationConfig,
                         CTPP::CDT   & oIMC,
                         ASLogger    & oLogger);

    // Виртуальный деструктор
```

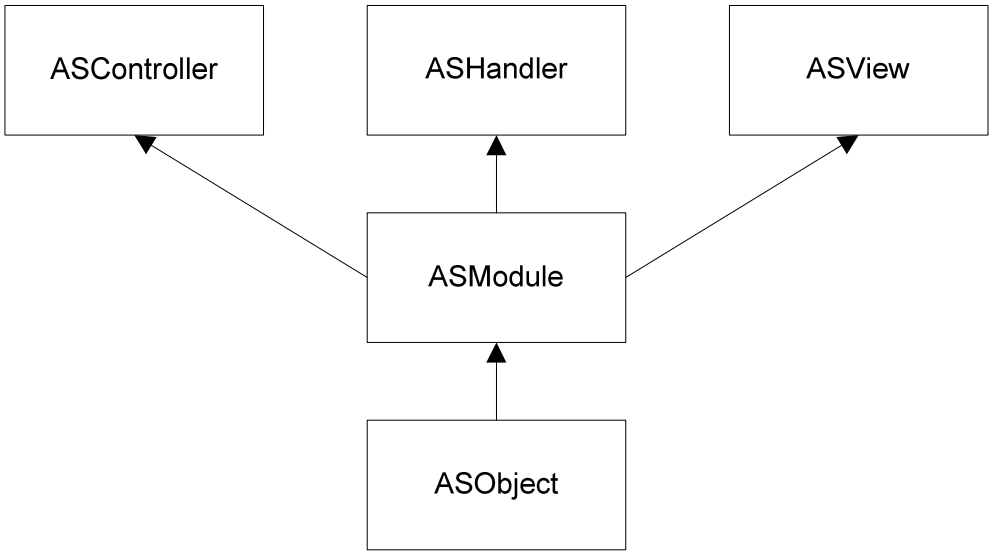
```
    virtual ~ASHandler() throw() { ;; }
};
```

Класс контроллера

Класс контроллера **ASController** во многом подобен классу обработчика **ASHandler**: он также является потомком класса **ASModule**, и предоставляет два дополнительных метода: зацепку для инициализации локации и метод обработки данных **Handler**. . Обязательным к реализации является только метод **Handler**.

Диаграмма наследования классов

Резюмируя все вышеописанное можно построить диаграмму наследования для классов **ASObject**, **ASModule**, **ASController**, **ASHandler** и **ASView**.



Вспомогательные классы и структуры данных

Кроме перечисленных классов, предоставляющих пользовательское API сервера приложений, следует обратить внимание на вспомогательные классы, реализующие поддержку пулов данных (**ASPool**), логгеров (**ASLogger**), HTTP запроса (**ASRequest**), HTTP ответа (**ASResponse**) и транспорта данных (**CDT**).

Логгер

Логгер предназначен для вывода различных диагностических сообщений в стандартное устройство файл протоколирования. Для вебсервера Apache таковым является файл, описанный в директиве ErrorLog конфигурации виртуального сервера.

Класс содержит в себе методы:

```
    // Установка минимального приоритета для вывода
    // Если сообщение имеет приритет ниже указанного, оно не выводится
    void SetPriority(const UINT_32 & iNewPriority);

    // Запись строки сообщения
    virtual INT_32 WriteLog(const UINT_32  & iPriority,
```

```
                          CCHAR_P          szString) = 0;

    // Запись строки сообщения с форматированием параметров,
    // поведение функции аналогично стандартной sprintf
    virtual INT_32 LogMessage(const UINT_32  & iPriority,
                              CCHAR_P          szFormat, ...);

    // Запись сообщения с наивысшим приоритетом
    // Особо важное сообщение, требующее немедленного вмешательства
    // в работу сервера
    INT_32 Emerg(CCHAR_P szFormat, ...);

    // Важное сообщение
    INT_32 Alert(CCHAR_P szFormat, ...);

    // Критическая ошибка
    INT_32 Crit(CCHAR_P szFormat, ...);

    // Ошибка
    INT_32 Err(CCHAR_P szFormat, ...);

    // Ошибка, дополнительный синоним для функции Err
    INT_32 Error(CCHAR_P szFormat, ...);

    // Предупреждение
    INT_32 Warn(CCHAR_P szFormat, ...);

    // Предупреждение, дополнительный синоним для функции Warn
    INT_32 Warning(CCHAR_P szFormat, ...);

    // Уведомление
    INT_32 Notice(CCHAR_P szFormat, ...);

    // Информация к сведению
    INT_32 Info(CCHAR_P szFormat, ...);

    // Отладочное сообщение
    INT_32 Debug(CCHAR_P szFormat, ...);
```

Пул объектов

Пул – хранилище объектов сервера приложений. Все пулы оперируют объектами типа **`ASObject`**. Для регистрации объекта в пуле требуется указатель на объект и идентификатор имени (строка STL). При регистрации объекта в пуле выдается уникальный числовой идентификатор. Для получения объекта из пула можно воспользоваться как именем, данным при регистрации объекта, так и уникальным идентификатором.
В сервере приложений существует три типа пулов:

- пул глобальных объектов
- пул объектов виртуального хоста
- пул объектов, создающихся на время текущего запроса
-

Важно понимать, что объекты только *хранятся* в пуле, и **во избежание утечек памяти программисту следует самостоятельно заботиться о своевременном создании и удалении объектов**.

Пул объектов реализован в виде класса и содержит следующие методы:

```
    // Удаление объекта из пула ресурсов по имени
    // sResourceName - имя ресурса
```

```
    // Возвращает 0, если удаление прошло успешно и -1 - если возникла ошибка
    template <typename T>
        INT_32 RemoveResource(const STLW::string & sResourceName);

    // Регистрация объекта в пуле ресурсов
    // pResource - объект
    // sResourceName - имя ресурса
    // Возвращает числовой идентификатор ресурса, если регистрация
    // прошло успешно и -1 - если возникла ошибка
    template <typename T>
        INT_32 RegisterResource(T                     * pResource,
                                const STLW::string  & sResourceName);

    // Получение объекта по имени
    // sResourceName - имя ресурса
    // Возвращает указательна ресурс или NULL, если ресурс не найден
    template <typename T>
        T * GetResourceByName(const STLW::string & sResourceName) const;

    // Получение объекта по числовому идентификатору
    // iResourceId - идентификатор
    // Возвращает указательна ресурс или NULL, если ресурс не найден
    template <typename T>
        T * GetResource(const UINT_64 & iResourceId) const;
```

Объект запроса

Запрос к серверу приложений хранится в структуре **ASRequest**. Эта структура содержит 13 полей, список которых представлен ниже.

- location_name – имя обрабатываемой локации ("MyLocation").
- request_method – метод запроса (GET, POST, HEAD …)
- uri – URI запроса ("/foo/bar.html")
- host – имя сервера ("www.example.com")
- port – порт сервера (80, 8080, 443)
- remote_ip – IP адрес клиента (10.10.1.11)
- uri_components – список компонентов URI.
- headers – заголовки HTTP запроса
- cookies – список переданных cookies
- arguments – список обработанных аргументов запроса
- user – имя пользователя, если была Basic Authorization (RFC 2617)
- password – пароль, если была Basic Authorization (RFC 2617)
- files – список загруженных файлов (RFC 1867)

Отдельно следует остановиться на списке компонентов URI. Если в конфигурации локации указано регулярное выражение, найденные компоненты содержатся в этом поле. Например, если задано регулярное выражение **`/foo/([0-9]+)/bar/([a-z]+)`** и имя URI - **`/location/foo/123/bar/baz/index.html`**, список будет выглядеть следующим образом:

```
"$0" : "/foo/123/bar/baz"
"$1" : "123"
"$2" : "baz"

"postmatch": "/index.html"
```

```
"prematch" : "/location"
```

Иными словами, список по своей структуре полностью повторяет концепцию переменных в регулярных выражениях языка Perl.

Поддержка регулярных выражений

Поддержка регулярных выражений реализована посредством библиотеки PCRE (http://www.pcre.org).
Для упрощения работы с библиотекой используется класс-враппер **ASPCRE**.
Класс содержит следующие методы:

```
// Результат применения регулярного выражения
struct Match
{
    // Начальная позиция найденной подстроки
    INT_32    match_start;
    // Конечная позиция найденной подстроки
    INT_32    match_end;
};

// Конструктор
// sRe - регулярное выражение
// iMaxMatches - максимальное количество элементов поиска
PCRE(CCHAR_P sRE, const UINT_32 & iMaxMatches = 10);

// Исполнение регулярного выражения
// szString - строка, в которой ведется поиск
// iStringLength - длина строки
// Возвращает количество совпадений или 0, если совпадений не найдено
INT_32 Exec(CCHAR_P         szString,
            const UINT_32 & iStringLength);

// Получение строки перед первым найденным элементом
Match PreMatch();

// Получение строки после последнего найденного элемента
Match PostMatch();

// Получение подстроки
// iMatchNum - номер элемента
Match GetMatch(const UINT_32 & iMatchNum);

// Получение подстроки
static STLW::string ExtractMatch(const STLW::string  & sText,
                                 const Match         & oMatch);
```

Объект ответа

Объект ответа предназначен для формирования заголовочной части HTTP ответа на запрос. Объект хранит в себе следующие данные: числовой код (200, 302, 404, 500 и т.п.), тип содержимого (Content-Type: text/plain, text/html и т.п.), набор выводимых заголовков и набор данных cookies.
Для облегчения работы класс предоставляет методы-аксессоры типа **`GetXXX/SetXXX`**, а также – методы для автоматического формирования ответа, например, **`Redirect`** или **`InternalServerError`**.

Поддержка Cookies

Для создания Cookies используется класс ASCookie.
Класс содержит следующие методы:

```
// Конструктор
// sName - Имя cookie
// sValue - Значение cookie
// iExpires - Время истечения срока хранения
// sDomain - Домен
// sPath - Путь внутри домена
// bHTTPOnly - Флаг "Только для HTTP"
ASCookie(const STLW::string  & sName,
         const STLW::string  & sValue,
         const UINT_64       & iExpires,
         const STLW::string  & sDomain,
         const STLW::string  & sPath = "/",
         const bool            bHTTPOnly = false);

// Получение строки cookie
CCHAR_P GetCookie();
```

Транспорт данных

Под транспортом данных подразумевается их передача между различными компонентами сервера приложений. В качестве такого транспорта используется класс CDT проекта CTPP. Класс позволяет передавать числа (целые и с плавающей точкой), строки, указатели, массивы простые и ассоциативные.
Предусмотрено два объекта-транспортера данных: внутренний транспорт, использующийся для передачи сообщений между контроллером, моделями и представлением – объект IMC(Inter-Module Communicator) и внешний транспорт, используемый для передачи только тех данных, которые следует выводить во View.

Алгоритм работы CAS

В этой главе мы детально рассмотрим все стадии работы сервера приложений: запуск, разбор конфигурации, загрузку и инициализацию модулей, обслуживание запросов, обработку ошибок, остановку и завершение работы.
К данной главе прилагаются блок-схемы, расположенные на отдельных вкладках.

Общая информация

Сервер приложений может работать в трех режимах: как CGI приложение, запускаемое на каждый запрос, как модуль Apache 1.3 и 2.X и как FastCGI сервер. Несмотря на наличие трех вариантов работы алгоритм остается практически неизменным, отличаясь только в несущественных деталях.

Инициализация

Инициализация сервера приложений происходит при запуске FastCGI сервера или сервера Apache (один раз при старте) или на каждый запрос, если сервер используется как CGI приложение. Фаза инициализации заключается в создании объекта класса **`ASServerManager`**. Для любого количества обслуживаемых виртуальных хостов требуется данный объект. При инициализации требуется задать объект логгера для ведения протокола ошибок.

Разбор конфигурации

Разбор глобальной конфигурации сервера – вторая фаза после инициализации. Для этого считывается файл глобальной конфигурации, содержащий в себе список загружаемых приложений (имена приложений, их типы, конфигурацию). Для разбора конфигурации используется универсальный класс обработки потока XML **`ASXMLParser`**, которому передается экземпляр класса **`ASGlobalConfigParser`**. Формат файла глобальной конфигурации приведен в приложении. В случае возникновения ошибок выбрасывается исключение.

Загрузка модулей

Если фаза разбора конфигурации прошла успешно, происходит загрузка всех перечисленных модулей (классы **`ASLoader`** и **`ASLoadableObject`**). Поиск файлов ведется последовательным перебором каталогов, указанных в секции конфигурации **`LibexecDirs`**. Указанные файлы читаются с диска функцией *dlopen(3)*, а создание объектов – путем получения указателя (*dlsym(3)*) и запуска функции, описанной в макросе **`EXPORT_HANDLER`**. Если функция вернула не NULL, считается, что получен указатель на созданный объект класса.

Создание глобального пула данных

Для хранения объектов используемых одновременно в нескольких виртуальных хостах используется глобальный пул данных. Такими объектами могут служить, например, постоянные подключения к СУБД (persistent database connections). Пул создается сразу после загрузки модулей, непосредственно перед их инициализацией.

Инициализация модулей

Сразу после создания глобального пула данных происходит инициализация загруженных модулей. Инициализация, также как и загрузка ведется последовательно, в порядке указания записей в файле конфигурации. Этот момент может оказаться важным в том случае, если некоторые модули совместно используют объекты из пула: прежде чем использовать объект из пула, его требуется создать.
После успешной инициализации всех модулей сервер приложений готов к работе.

Инициализация виртуального хоста

Для обработки запросов по протоколу HTTP требуется создание как минимум одного виртуального хоста. Процесс инициализации виртуального хоста выглядит следующим образом:

- загружается и обрабатывается конфигурация
- создается пул объектов сервера
- вызываются зацепки всех указанных в конфигурации модулей

В случае отсутствия ошибок считается что виртуальный хост инициализирован и можно принимать подключения к данному ресурсу.

Обслуживание запросов

Обслуживание запросов разбито на 4 фазы:

1. получению объекта виртуального хоста по имени сервера из HTTP запроса
2. определения, обрабатывается ли заданный location сервером
3. чтения всего тела POST запроса (если есть)
4. создания пула объектов запроса, исполнения набора приложений указанных в конфигурации данного location и вывода результата пользователю

Определение, обрабатывается ли заданный location сервером происходит путем последовательной проверки всех указанных для данного виртуального сервера локаций. В случае если локация обрабатывается, происходит обслуживание запроса. Если нет – управление отдается вызывающему проверку коду. Так достигается возможность параллельной работы сервера приложений с такими технологиями как mod_php или mod_perl.

Для чтения GET и POST запросов применен конечный автомат с запоминанием текущего состояния. Для упрощения кода и ускорения данной фазы обработки запроса использована схема на GOTO-таблицах. Поскольку CAS оперирует только готовыми объектами запроса и ответа, функционал предварительной разборки данных вынесен в отдельные программные модули (каталог **`src/SAPI/util/`**) и построен в процедурном, а не объектном стиле. Необходимость процедурного стиля диктуется тем соображением, что подавляющее большинство интерфейсов к вебсерверам реализованы на языке C.

Далее происходит обработка запроса: вызов контроллера (если есть), исполнение моделей (если указаны) и вызов представления для формирования ответа пользователю.

Исполнение набора моделей происходит последовательно, в порядке, указанном в конфигурации.

По окончании исполнения моделей вызывается представление и ответ отправляется клиенту.

Обработка ошибок и исключительных ситуаций

В случае возникновения ошибки при исполнении запроса вызываются *финализаторы*. Финализатор – метод обработчика, используемый для чистки "мусора", возможно возникшего при обработке запроса.

Важный момент: последовательность вызова финализаторов – обратна по отношенияю к последовательности вызова обработчиков. Кроме того, вызываются только финализаторы тех обработчиков, которые были исполнены. Таким образом, если в конфигурации были указаны обработчики приложений A, B, C и D, а ошибка возникла в C, будут вызваны финализаторы в порядке C, B и A.

В случае возникновения ошибки в финализаторе считается, что произошла фатальная ошибка и управление отдается вызывающему процессу.

Завершение работы сервера

При завершении работы сервера происходит последовательный вызов завершающих методов для всех загруженных приложений. Порядок завершения – обратный, то есть модули, которые стартовали раньше, завершаются позже. Таким образом при завершении гарантируется, что объекты, использующие данные из пулов, могут к ним обращаться с гарантией правильного функционирования.

Стандартные компоненты

Модули расширения

Модули за исключением тестовых, не входят в стандартную поставку дистрибутива сервера. В случае, если требуется тот или иной модуль расширения, разработчик или системный администратор должны доустановить его самостоятельно.
В зависимости от используемого модуля может также возникнуть необходимость установить сторонние библиотеки, например, для SQLConnector потребуется библиотека SQLayer.
Список официально поддерживаемых модулей дан ниже:

- MemcachedConnector – интерфейс для работы с сервером memcached
- SQLProxy – прокси-сервер для SQL СУБД
- SQLConnector – ADOdb-подобный коннектор к СУБД.
- Dumper – дампер данных

MemcachedConnector

Модуль предназначен для работы с сервером memcached.

SQLProxy

Модуль предназначен для работы с сервером SQLProxy

SQLConnector

Модуль предназначен для работы с СУБД MySQL, PostgreSQL и Oracle.

Dumper

Модуль предназначен для дампа данных, содержащихся в объектах зфпроса (ASRequest) и межмодульного взаимодействия (IMC).
Модуль не требует какой-либо настройки.

XMLRPCClient

Модуль реализует клиента XML RPC.

Представления

Сервер приложений содержит в стандартной поставке 6 модулей представления (View):

- CTPP2View
- PlainView – непосредственный вывод данных
- JSONView – сериализация в JSON
- XMLRPCView – поддержка спецификации XML RPC
- SOAPView – поддержка стандарта SOAP
- XMLView – сериализация в XML

Как правило, компоненты представления не нуждаются в предварительной настройке и могут использоваться с параметрами “по умолчанию”.

CTPP2View

Представление производит рендеринг шаблона CTPP.

PlainView

Прдставление предназначено для непосредственного вывода данных без какой-либо трансформации.
Типичное применение – вывод содержимого captcha или двоичного файла данных.

JSONView

Представление производит сериализацию переданных данных в формат JSON.
Типичное применение – AJAX автоматизация и экспорт данных.

XMLRPCView

Представление производит сериализацию данных согласно спецификации XML RPC.
Типичное применение – реализация совместно с обработчиком POST запроса XML RPC сервера приложений XML RPC.

SOAPView

В разработке.

XMLView

В разработке.
Представление производит сериализацию переданных данных в формат JSON.
Типичное применение – AJAX автоматизация, экспорт данных, различные варианты RPC.

Установка и развертывание CAS

Есть два варианта установки сервера приложений: из готовых пакетов и сборка из исходных кодов. На данный момент готовые пакеты существуют для следующих операционных систем:

- Solaris, ultrasparc и amd64
- FreeBSD, i386 и amd64
- CentOS Linux, i386 и amd64
- Debian GNU/Linux, i386 и amd64

Если ваша система присутствует в этом списке, достаточно воспользоваться стандартным менеджером пакетов и установить требуемые компоненты. В этом случае следующую часть главы можно пропустить и рассмотреть только вопросы конфигурирования сервера.

Сборка из исходных кодов

Для сборки из исходных кодов требуется:

- компьютер под управлением ОС Solaris, FreeBSD, OpenBSD или Linux,
- ANSI C++ компилятор (GNU g++, Sun CC, icc)
- библиотека CTPP версии 2.3.4 и старше
- вебсервер Apache 1.3 или 2.X

Процесс сборки не представляет сложностей:

```
gunzip cas-3.2.1.tar.gz | tar -xf
cd cas-3.2.1
cmake .
make
make install
```

По умолчанию CAS собирается с поддержкой Apache 1.3, без вывода отладочной информации и с установкой файлов примеров.

В случае необходимости подобное поведение можно изменить при помощи следующих опций:

Опция	Значение по умолчанию	Комментарий
DEBUG_MODE	OFF	Режим отладки.
ENABLE_OPTIMIZATION	ON	Сборка с флагами оптимизациями компиляции
BUILD_APACHE13_MODULE	ON	Поддержка Apache 1.3
BUILD_APACHE2X_MODULE	OFF	Поддержка Apache 2.X
INSTALL_EXAMPLE_MODULES	OFF	Установка файлов примеров
CAS_GLOBAL_CONFIG_FILE	Зависит от ОС	Местоположение файла глобальной конфигурации
CAS_LIBEXEC_DIR	libexec/cas	Каталог модулей CAS
CAS_SHAREDIR	share/cas	Каталог примеров
APACHE_LIBEXEC_DIR	-	Каталог модулей Apache

Для этого требуется либо отредактировать файл CmakeLists.txt, либо указать требуемые значения при запуске cmake:

```
cmake . -DENABLE_OPTIMIZATION=OFF -DDEBUG_MODE=ON
```

Конфигурирование сервера приложений

Конфигурирование сервера приложений сводится к созданию файлов глобальной и локальной конфигурации, а также – к настройке вебсервера.

Глобальная конфигурация

Файл глобальной конфигурации должен находиться в каталоге, указанном в директиве CAS_GLOBAL_CONFIG_FILE. Значения по умолчанию следующие:

Linux: **`/etc/cas/global-config.xml`**

FreeBSD: **`/usr/local/etc/cas/global-config.xml`**

Solaris: **`/opt/CASWserver/conf/global-config.xml`**

Пример конфигурации можно найти в каталоге **`conf/`** исходных кодов проекта.
Начиная с версии 3.2.1 есть возможность переопределить место расположения файла глобальной конфигурации посредством указания переменной окружения CAS_GLOBAL_CONFIG.

Конфигурация виртуального хоста.

Конфигурация виртуального хоста задается в зависимости от используемого интерфейса. Для вебсервера Apache как версий 1.3, так и 2.X следует настроить виртуальный хост.

Пример настройки дан ниже:

```
NameVirtualHost *:80
<VirtualHost *:80>
    DocumentRoot  /home/project/www

    CASEnable     On
    CASConfigFile /home/project/conf/vhost-config.xml
</VirtualHost>
```

Директива **`CASEnable`** включает поддержку сервера приложений для заданного хоста (по умолчанию – выключено), а директива **`CASConfigFile`** – указывает путь к конфигурационному файлу.

cas-globalconf

Утилита предназначена для тестирования или просмотра файла глобальной конфигурации сервера. Может быть запущена без аргументов, в этом случая проверяется файл, указанный в переменной окружения CAS_GLOBAL_CONFIG, а если эта переменная не задана – то файл конфигурации по "умолчанию". При запуске с одни параметром проверяется файл, указанный в аргументе.

Пример:

```
cas-globalconf

Global config not given, using /usr/local/etc/cas/global-config.xml as
DEFAULT
  Libexec dirs:
      .
      /usr/lib/cas
      /usr/local/libexec/cas
      /opt/REKIWcas/libexec

  Modules:
      Name:         ASExampleHandler
      Type:         Handler
      Library file: /usr/local/libexec/cas/mod_example_handler.so

      Name:         ASExampleModule
      Type:         Module
      Library file: /usr/local/libexec/cas/mod_example_module.so

      Name:         ASMemcachedConnector
      Type:         Module
      Library file: /usr/local/libexec/cas/mod_memcached_module.so

      Name:         ASMemcachedTest
      Type:         Handler
      Library file: /usr/local/libexec/cas/mod_memcached_test.so

      Name:         CTPP2View
      Type:         View
      Library file: /usr/local/libexec/cas/mod_ctpp2_view.so

      Name:         PlainView
      Type:         View
      Library file: /usr/local/libexec/cas/mod_plain_view.so

      Name:         JSONView
      Type:         View
      Library file: /usr/local/libexec/cas/mod_json_view.so

      Name:         XMLRPCView
      Type:         View
      Library file: /usr/local/libexec/cas/mod_xmlrpc_view.so
```

cas-hostconf

Утилита предназначена для тестирования или просмотра файла конфигурации виртуального хоста. Единственный аргумент – имя файла конфигурации виртуального хоста. Порядок вывода данных:

- конфигурация виртуального хоста
- список модулей данного виртуального хоста
- список локаций
- имя контроллера локации
- список модулей для каждой локации
- список обработчиков
- представление

Пример:

```
cas-hostconf /usr/local/etc/cas/vhosts/vhost-config.xml
  Server name:           cas.havoc.ru
  Server root:           ./
  Temp. files directory: /tmp
  Template include directories:
      /var/www/cas
      /usr/share/cas/tmpl
      /usr/local/share/cas/tmpl

  Per-server modules configuration:
    Module: ASMemcachedConnector
          Configuration:
                  Memcached =>
                        Connection =>
                              ConnString => "host='127.0.0.1' port='11211'"
                              Name => "SessionConnector"
                              Type => "Global"
  Locations:
      Name: Test
      Allowed methods:            All
      Max. POST request size:     unlimited
      Max. size of uploaded file: unlimited
      Default HTTP Response code: 200
      Default HTTP content type:  text/xml; charset=utf-8
      URI(s):
          /plain-test (Plain)
          /regexp-test([a-zA-Z0-9]+) (Regexp)

      Handlers(s):
          ASTestMemcached
          Configuration: `
                  Memcached =>
                        Connection =>
                               Name => "SessionConnector"
          ASExampleHandler

      View: XMLRPCView
```

cas-regexp

Утилита предназначена для тестирования регулярных выражений, используемых в конфигурации локации. Утилита принимает два параметра: регулярное выражение и строку, требующуюся для проверки: **`cas-regexp regexp string`**.

Пример:

```
cas-regexp '/news/([a-zA-Z0-9]+)' '/news/elections2012/index.html'

Result: HASH {
    $0 => /news/elections2012
```

```
    $1 => elections2012
    postmatch => /index.html
    prematch =>
}
```

cas-xt

CAS eXtension Tool – утилита для создания модулей, представлений и приложений для CAS.
Для упрощения процесса разработки используется утилита cas-xt. Общий синтаксис команды следующий

`cas-xt -t <module type> -g -n <module name>`

Назначение флагов командной строки:
-t – указание типа модуля
-g – создание каталогов и файлов модуля
-n – имя модуля

`<module type>` - тип модуля, может быть **`"handler"`**, **`"module"`**, **`"controller"`** или **`"view"`**

`<module name>` - имя модуля. Допускаются латинские буквы, знак подчеркивания ”_” и цифры. Имя должно начинаться с буквы или символа подчеркивания.

cas-xmlrpc-parser

Утилита может быть использована для обработки файла запроса XML RPC. Принимает единственный параметр: имя файла с запросом XML RPC:

`cas-xmlrpc-parser xmlrpc.xml`

Пример:

```
cas-xmlrpc-parser xmlrpc.xml
Result: HASH {
    method => examples.getStateName
    params => ARRAY [
        0 : 41
        1 : HASH {
            param1 => Value1
            param2 => Value2
        }
    ]
}
```

Файл глобальной конфигурации

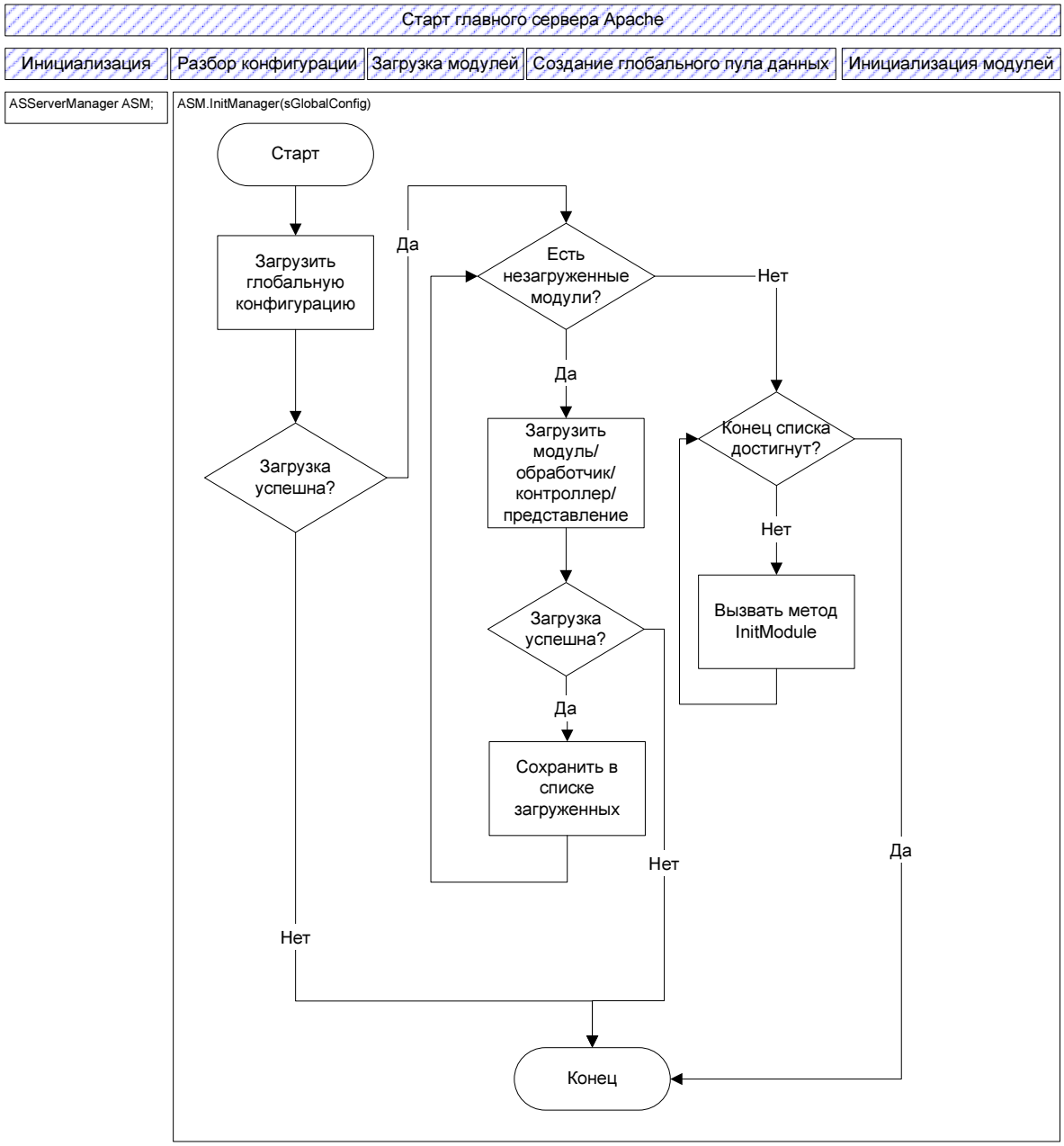
```
<?xml version="1.0" ?>
<CASConfig version="1.0">
    <!-- Список каталогов модулей CAS  -->
    <LibexecDirs>
        <!-- Для Linux -->
        <LibexecDir>/usr/lib/cas</LibexecDir>

        <!-- Для FreeBSD -->
        <LibexecDir>/usr/local/libexec/cas</LibexecDir>

        <!-- Для Solaris -->
        <LibexecDir>/opt/REKIWcas/libexec</LibexecDir>
    </LibexecDirs>

    <!-- Список модулей -->
    <Modules>
        <Module Name="Name_of_module"
                Library="shared_library.so"
                ModuleType="Type_of_module">

           <Parameter>1024</Parameter>
        </Module>
    </Modules>
</CASConfig>
```

Name_of_module – имя модуля, строка
Shared_library.so – имя файла библиотеки, из которой следует загрузить модуль
Type_of_module – тип подуля, например "Module", "Handler" или "View".

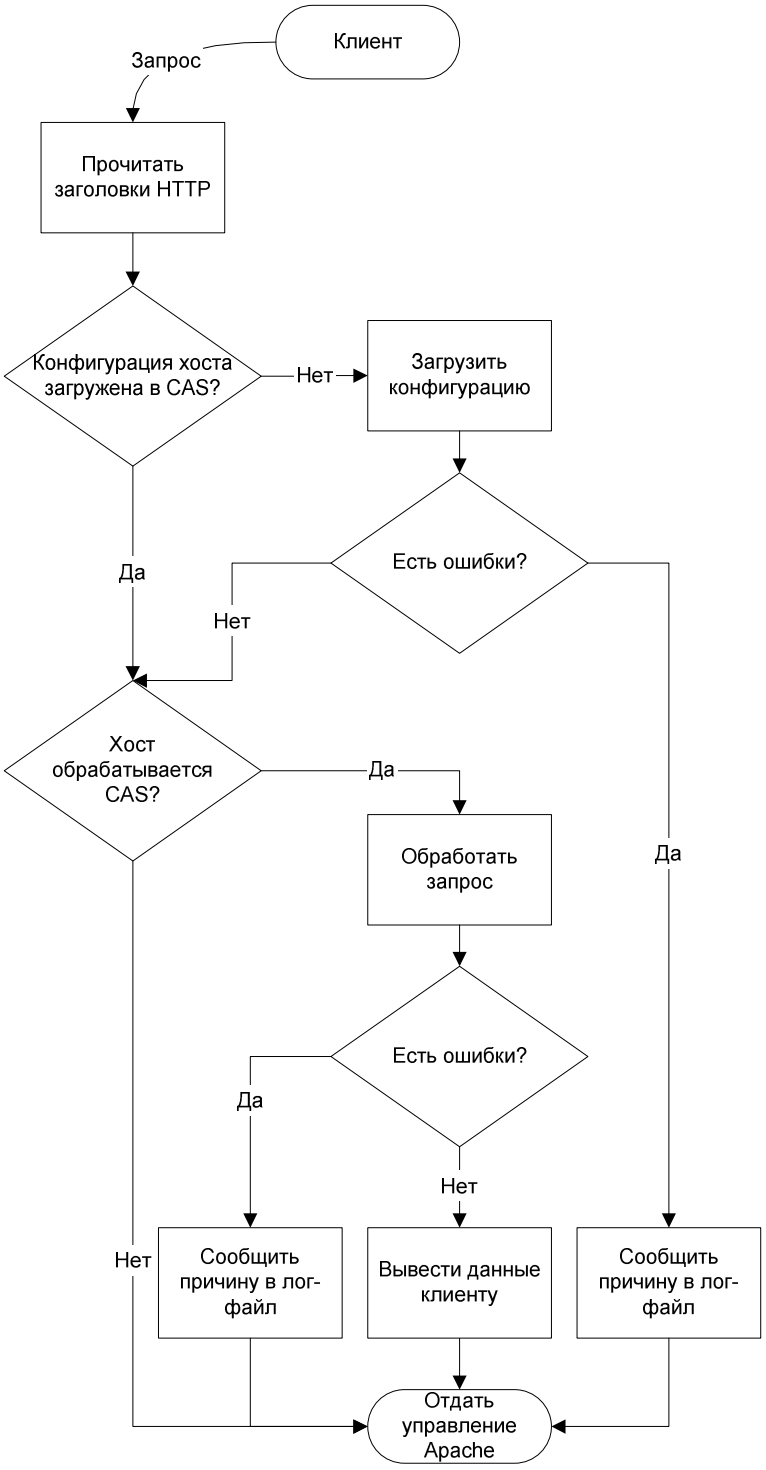
Файл конфигурации виртуального хоста

TBD

Общий алгоритм инициализации сервера



Общий алгоритм исполнения запроса



Процесс обработки данных MVC