Сервер приложений C++

Андрей Шетухин, Илья Космодемьянский

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 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, простоте администрирования и эксплуатации.

Сервер приложений имеет интерфейсы для работы с Apache 1.3, Apache 2.X и FastCGI. Поддерживаются платформы: FreeBSD (i386, amd64), Linux 2.6 (i386, amd64), Solaris (i386, amd64, sparc, ultrasparc).

Конфигурация CAS хранится в виде XML и допускает возможность задать большинство параметров “по 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 этом докладе мы расскажем как использовать сервер приложений для разработки сложных высоконагруженных вебпроектов на языке C++.

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:

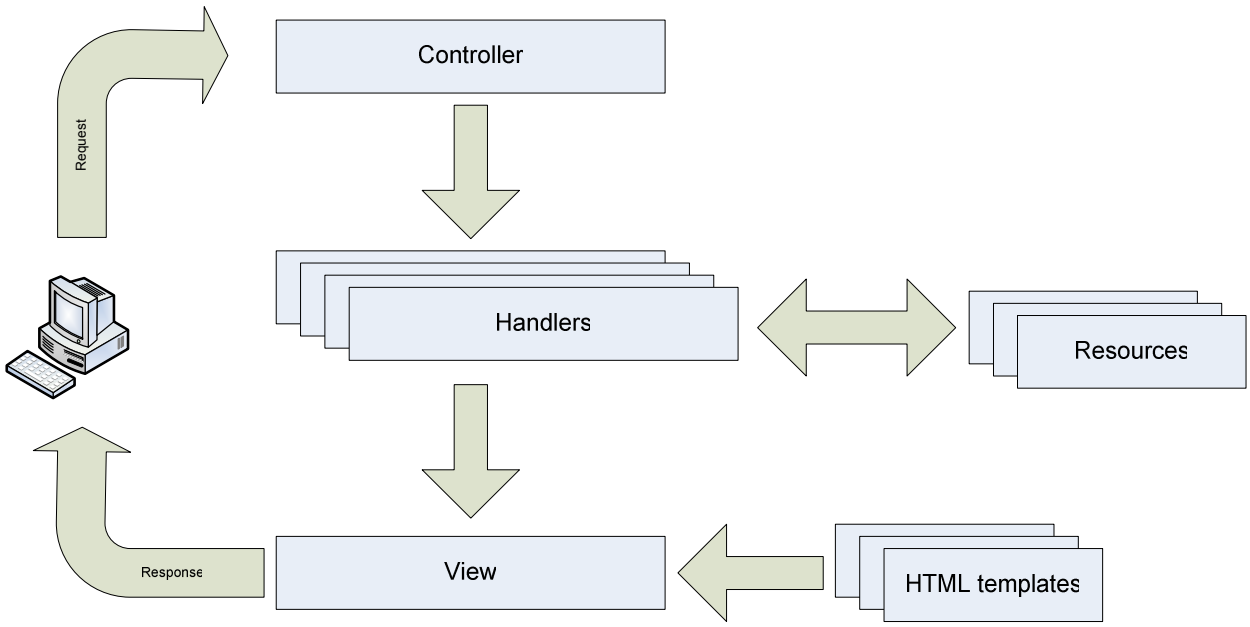

Рис. 1. Общая схема архитектуры MVC

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.

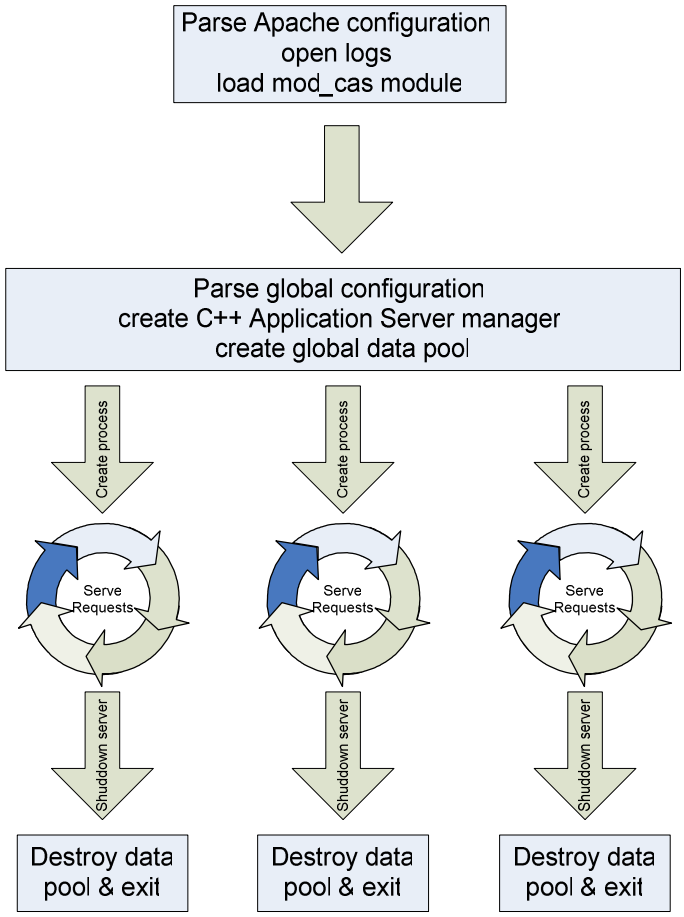

Рис. 2. Схема загрузки и инициализации CAS.

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.

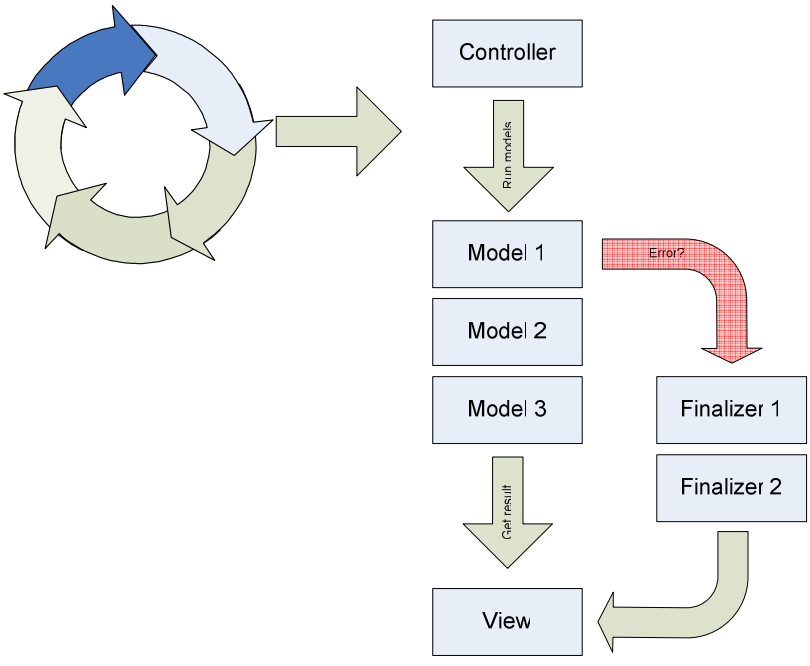

Рис. 3. Цикл обработки запроса

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.

Чтобы сказанное было более понятно, рассмотрим пример создания простейшего приложения.

Простейшее приложение CAS

В нашем приложении 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
- 10 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**:

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>/home/example.com/tmpl/hello.tmpl</Template>
     </Templates>
     <Handlers>
        <Handler name="Hello"/>
     </Handlers>
     <View>
        <Handler name="CTPP2View"/>
     </View>
  </Location>
  ....
</Locations>
```

После этого 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: Wed, 30 Jul 2008 12:00:41 GMT
Server: Apache/1.3.41 (Unix) mod_cas/3.1.4(Feather)
X-Powered-By: C++ Application Server v 3.1.4(Feather)
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.

Интеграция с популярными технологиями

Разумеется, мы не призываем полностью отказаться от использования традиционных веб-технологий. Используя CAS вместе с библиотекой CTPP для Perl или PHP вы можете создавать проекты, высоконагруженная часть которых реализована на технологии сервера приложений, а низконагруженная – на классических Perl или PHP.

Подобный подход даст возможность очень быстро создать прототип проекта, найти самые нагруженные его части и провести их портирование на CAS. При этом шаблоны страниц и набор данных для их отображения останутся неизменными, что позволит произвести максимально плавную миграцию серверной части.

Выводы

- сервер приложений C++ – зрелый, высокотехнологичный OpenSource продукт, готовый к коммерческого использования.
- производительность CAS является наиболее высокой среди популярных вебтехнологий, а потребление памяти и процессорного времени – самым низким, что в совокупности своей позволяет использовать CAS в высоконагруженных проектах любой сложности
- модульность системы дает возможность параллельной разработки проекта большим коллективом программистов
- продемонстрированные примеры свидетельствуют о простоте разработки, настройки и обслуживания CAS
- совместное использование с другими средами (Perl, PHP, Python) допускает быстрое прототипирование и эффективную технологическую миграцию проекта
- использованный язык шаблонов содержит в своей основе синтаксические соглашения широкоизвестных модулей Perl, а значит – прост в понимании и освоении верстальщиками HTML