Применение шаблона проектирования Read/Write Lock для управления доступом к функциональности сервера приложений

П.П. Олейник

Аннотация. Рассмотрено практическое применение шаблона проектирования Read/Write Lock для разграничения доступа к функциональности сервера приложений, поддерживающего динамическую компиляцию модулей расширений.

Введение

Современные информационные системы, манипулирующие большими массивами данных, строятся на основе трёхзвенной архитектуры. Ключевым звеном данной архитектуры является сервер приложений (СП) [1]. Как правило, СП представляет собой многопоточное приложение, написанное на объектно-ориентированном языке программирования. Ключевой проблемой при проектировании многопоточного СП является синхронизация доступа к ресурсу, запрашиваемого многими клиентами. Выбор оптимального алгоритма блокировки ресурса влияет как на производительность полученной реализации, так и на возможности модификации и расширения функциональности системы.

В статье рассмотрено практическое применение шаблона проектирования (design pattern) Read/Write Lock (Блокировка чтения/записи) [2] для разграничения доступа к функциональности сервера приложений. Обоснование выбора алгоритма базируется на анализе потребностей различных групп приложений, использующих функционал СП.

Рассмотрим современные подходы, применяемые при реализации СП. В литературе рассматриваются два основных пути решения задачи:

1. Разработка множества независимых серверных объектов, каждый из которых решает какую-либо конкретную задачу [3,4];

2. Разработка одного загрузчика объектов, который управляет созданием и управлением бизнес-объектов. В настоящее время данный подход используется не только для создания сервера приложений, но и при разработке других программ. Эта реализация называется архитектурой с поддержкой плагинов (pluggins) или архитектурой с поддержкой модулей расширений [5-8].

Проанализируем достоинства и недостатки данных подходов. Основным достоинством первого подхода является быстрая разработка и развёртывание, так как нет необходимости реализовывать какую-либо инфраструктуру. К недостатку можно отнести усложнение доработки системы, так как в ней существуют множество не связанных друг с другом классов.

Основным достоинством второго подхода является единообразная архитектура, которая

присуща всем объектам. Благодаря этому происходит централизованное управление серверными объектами и возможность динамического расширения функциональности сервера приложений. Слабой стороной подхода является необходимость тщательного проектирования интерфейса сервера приложений, так как последующее расширение функциональности интерфейса может потребовать доработки уже созданных классов серверных объектов. Но, как правило, этот недостаток не является существенным.

В работах [5-8] в качестве плагинов используются Dll библиотеки (Dynamic-link library - динамически подключаемая библиотека). Это позволяет разработать класс на определённом языке программирования и скомпилировать его с помощью соответствующего компилятора. Полученную библиотеку можно использовать в качестве плагина. Основным достоинством данного подхода является простота реализации объекта-загрузчика плагинов. Но, к сожалению, имеется и существенный недостаток. В момент загрузки Dll библиотеки происходит блокировка файла. После неё невозможно заменить Dll библиотеку без перезапуска сервера приложений. То есть невозможно динамически изменить функциональность СП без физического перезапуска. Учитывая тот факт, что современные информационные системы обслуживают несколько сотен (а иногда и тысяч) пользователей, перезагрузка СП приведёт к простоям в работе. Этот недостаток является существенным. Целью данной статьи является разработка и реализация архитектуры СП, лишенной отмеченного недостатка.

Алгоритм доступа в виде шаблона проектирования

По нашему мнению идеальным вариантом реализации архитектуры СП, является загрузка модулей приложений, реализованных в виде исходного текста классов и компиляция кода в момент выполнения СП. В этом случае отсутствует необходимость в перезагрузке приложения, и, тем самым, устраняются недостатки, отмеченные выше. Данный подход был применён при разработке многопоточного сервера приложений, поддерживающего динамическую компиляцию программного кода (SharpArchitect Application Server) [9]. Основной особенностью архитектуры СП является наличие возможности динамической компиляции модулей расширений (плагинов) в момент выполнения. Для анализа потребностей пользователей и выбора алгоритма синхронизации доступа к ресурсам, были выделены две группы приложений, использующих функционал СП (Табл. 1).

Табл. 1. – Группы приложений, использующие SharpArchitect Application Server

№	Группа	Выполняемые функции
1	Клиентское приложение	Создание объекта, класс которого расположен в плагине. Вызов метода созданного объекта, передача параметров, получение результата.
2	Приложение администратора	Обновление (перезагрузка) плагинов. Динамическая компиляция программного кода.

Как видно основной задачей клиентского приложения является использование функционала, имеющегося на СП. В свою очередь приложение администратора (разработчика) необходимо для расширения бизнес-логики.

Будем рассматривать процесс создания объекта и последующий вызов его методов как чтение, а перезагрузку плагинов как запись. Синхронизация доступа к SharpArchitect Application Server требует применения шаблона проектирования Read/Write Lock.

Данный паттерн разрешает осуществлять параллельный доступ к объекту для операций чтения, а монопольный доступ для операции записи. При этом блокировки чтения и записи не содержат никакой информации, поэтому их не нужно представлять в виде отдельных объектов. Разные варианты реализации данного шаблона отдают предпочтение блокировке чтения. Это означает, что если некоторый вызов ожидает получение блокировки записи, поскольку существуют невыполненные блокировки чтения, то любые запросы, желающие получить дополнительные блокировки чтения, в таких обстоятельствах будут удовлетворены немедленно. Подобная методика уместна для многих приложений. Но для нашего она неуме-

стна, так как при её применении операция перезагрузки плагинов может ожидать начала своего выполнения бесконечно долго. Это связано с тем, что невозможно заранее предсказать активность клиентских приложений и количество запросов, поступающих от них.

На временной диаграмме, представленной на Рис. 1, изображена оптимальная последовательность обработки вызовов методов серверного объекта, поступающих как от клиентского приложения, так и от приложения администратора. Предполагается, что $t_i > t_{i-1}$.

На рисунке под вызовом метода понимается непосредственная обработка исполняющей среды метода на клиентском приложении или на приложении администратора. Под началом и окончанием метода полагается непосредственное выполнение метода на сервере приложений.

Как видно на Рис. 1 несколько вызовов метода создания объекта могут выполняться параллельно. В момент времени t_1 оба клиентских приложения начали вызывать описанный метод. При этом в момент t_2 вызов, поступивший из второго клиентского приложения, начал непосредственное выполнение на сервере приложений.

Вызов метода перезагрузки плагинов, поступившего в момент t_3, заблокирован до тех пор, пока не закончат своё выполнение все ранее вызванные методы, поступившие от клиентских приложений. В момент времени t_6 поступает новый вызов метода создания объекта. Он ставится в очередь и ожидает окончания перезагрузки плагинов. Такая последовательность вызова и обработки методов соответствует оптимальной реализации шаблона проектирования Read/Write Lock, которая использована в SharpArchitect Application Server.

Отметим, что при разработке SharpArchitect Application Server использован язык программирования C#, поэтому были применены классы управления синхронизацией потоков, входящие в платформу .Net Framework. В частности, применён класс ReaderWriterLock, который реализует описанный шаблон проектирования [10] и введена статическая перемен-

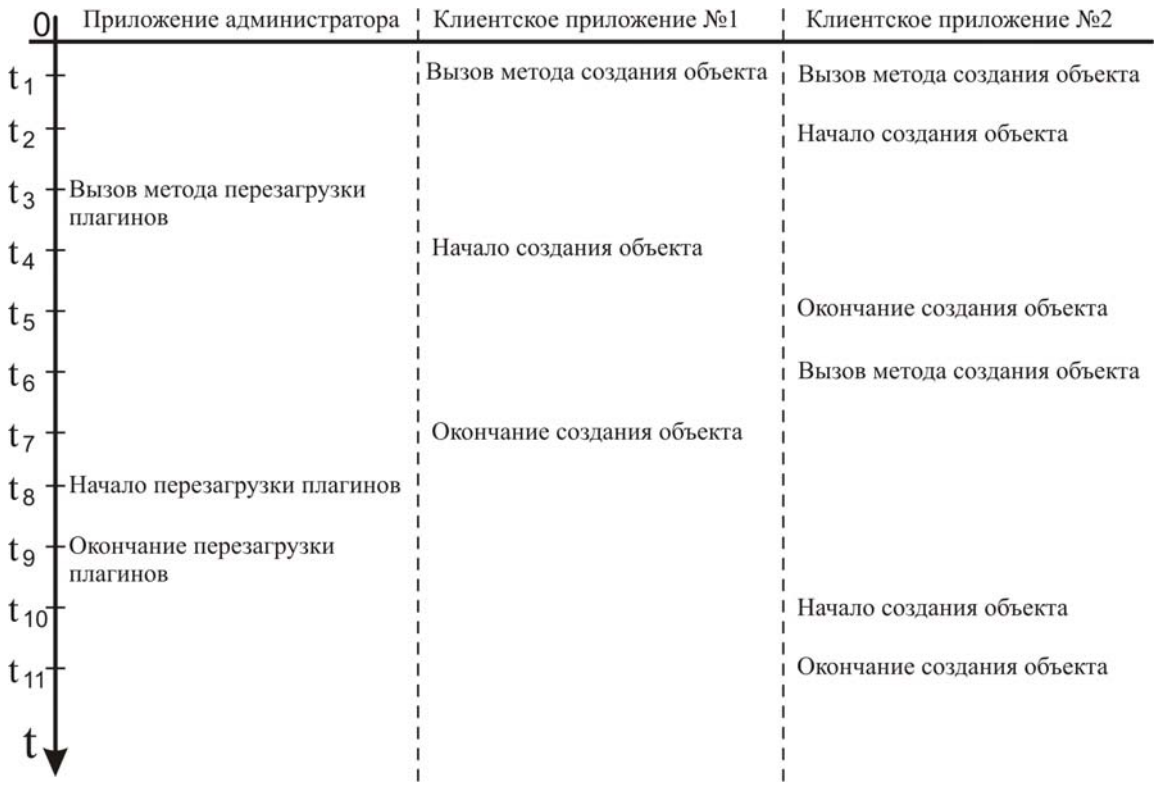

Рис. 1. Оптимальная последовательность обработки вызовов на СП

ная данного классе, которая управляет синхронизацией доступа к функциональности. Также разработана иерархия интерфейсов и классов, которые позволяют выполнять запросы к БД и возвращать результат в виде коллекции объектов [11-13]. Разработаны вспомогательные классы, экземпляры которых сохраняют информацию о местоположении файлов с исходным кодом плагинов, а также о вспомогательных библиотеках, необходимых для компиляции. Имеется возможность сохранения результатов выполнения компиляции программного кода, отображения ошибок и диагностических сообщений. Тестирование полученной реализации сервера приложений показало, что она соответствует оптимальной.

Заключение

Данный алгоритм синхронизации доступа к функциональности СП, реализованный в виде шаблона проектирования может быть повторен практически на любом языке программирования. При этом необходимо реализовать оптимальный алгоритм без предпочтений блокировкам какого-либо конкретного вида. В будущем имеет смысл рассмотреть возможность подключения нескольких компиляторов для различных языков программирования. При этом возникает проблема распознания языка программирования по исходному коду плагина и последующий выбор соответствующего компилятора.

Литература

1. Флауер М., Архитектура корпоративных программных приложений, Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 544 с.: ил. – Парал. тит. англ.
2. Гранд М., Шаблоны проектирования в Java, Пер. с англ. С. Беликовой. – М.: Новое знание, 2004. – 559с.: ил.
3. Лейнекер Р., COM+. Энциклопедия программиста, Пер. с англ.: СПб.: ООО «ДиаСофтЮП», 2002. – 656с.
4. Фаронов В.В., Шумаков П.В., Delphi 5. Руководство разработчика баз данных, М.: «Нолидж», 2001. – 640., ил.
5. Рихтер Дж., Программирование на платформе Microsoft .Net Framework, Пер. с англ. – 2-е изд., испр. – М.: Издательско-торговый дом «Русская редакция», 2003 – 512 стр.: ил.
6. Osherove R., Creating a Plug-In Framework, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/pluginframework.asp
7. Boland D., Pluggable Remote Object Hosting, http://www.codeproject.com/csharp/dbremoteplugins.asp
8. Straughan A., Dynamically Loading an Assembly, http://www.csharpcorner.com/Code/2002/April/LoadingAssemblyInfo.asp
9. Многопоточный сервер приложений, поддерживающий динамическую компиляцию модулей расширений (SharpArchitect Application Server), Свидетельство об официальной регистрации программы для ЭВМ № 2006613503, 9 октября 2006 г.
10. Microsoft Corp., ReaderWriterLock - класс, http://msdn.microsoft.com/library/rus/default.asp?url=/library/RUS/cpref/html/frlrfSystemThreadingReaderWriterLockClassTopic.asp
11. Олейник П.П., Практические аспекты реализации многопоточного сервера приложений // Информационные технологии, № 1, 2007.
12. Олейник П.П., Внутренняя архитектура корпоративного сервера приложений // Теория, методы проектирования, программно-техническая платформа корпоративных информационных систем: Материалы IV Междунар. науч.-практ. конф., г. Новочеркасск, 26 мая 2006 г. / Юж.-Рос. гос. техн. ун-т (НПИ). – Новочеркасск: ЮРГТУ, 2006. – 142 с.
13. Олейник П.П., Об одном способе реализации трёхзвенного приложения // Компьютерные технологии в науке, производстве, социальных и экономических процессах: Материалы VII Междунар. науч.-практ. конф., г. Новочеркасск, 17 нояб. 2006 г.: В 3 ч. / Юж.-Рос. гос. техн. ун-т (НПИ). – Новочеркасск: ООО НПО «Темп», 2006. – Ч.2. - 80 с.

Олейник Павел Петрович. Родился в 1983 г. Окончил Шахтинский институт, филиал Южно-Российского государственного технического университета в 2004 г. Имеет 6 публикаций и 11 докладов. Область интересов: объектно-ориентированные системы, компонентное программирование, алгоритмы, системы построения пользовательского интерфейса, ERP – системы, объектно-реляционные преобразования, базы данных, шаблоны проектирования, распределённые системы. Инженер-программист ООО «Волшебный край».