**организация веб-сервисов в распределённых приложениях, построенных на модульной архитектуре**

**Д. В. Цветцих, А. В. Цветцих**

*Сибирский Федеральный Университет, Красноярск, Россия*

Появление веб-сервисов открыло новый этап в истории разработки распределённых приложений. Благодаря использованию открытых стандартов и протоколов, а также передаче данных в текстовом формате XML, веб-сервисы сделали возможным взаимодействия любых программных систем, в независимости от их платформы. Количество инструментов для разработки, а также серверов приложений для развёртывания веб-сервисов быстро увеличивалось, вместе с ним увеличивалось количество распределённых приложений, разрабатываемых на основе веб-сервисов. Однако из-за отсутствия эффективных практик использования сервисов, построения модульных распределённых систем на основе сервис – ориентированной архитектуры, поддержка многих приложений, разработанных на основе веб-сервисов, оказывалась очень сложной и трудоёмкой задачей. В данной статье мы рассмотрим способы построения модульной архитектуры распределённых приложений, а также проанализируем их на предмет эффективности при решении задач расширения и сопровождения приложений.

**Модульная структура ярусов.**

Каждое распределённое приложение состоит из двух независимых ярусов: сервера приложений и клиента, взаимодействующих при помощи веб-сервисов. Однако общие принципы построения модульных приложений одинаковы как на клиенте, так и на сервере (хотя детали их реализации могут различаться). Разделение приложения на модули требует организации модулей в иерархию.

*Стартовый модуль* является главным по отношению к остальным модулям яруса, он не содержит функционала предметной области, но реализует композицию приложения из модулей и одновременной является точкой запуска приложения.

*Функциональный модуль* содержит реализацию блока операций предметной области и не является самостоятельным приложением.

Существует два способа загрузки функциональных модулей стартовым модулем: статический и динамический. В случае статической загрузки стартовый модуль имеет ссылки на все функциональные модули. Данный метод прост и не требует усилий в реализации, однако имеет ряд существенных недостатков при сопровождении:

* Добавление новых модулей возможно лишь после перекомпиляции стартового модуля
* Неконтролируемый рост количества ссылок со стартового модуля на функциональные

Динамический способ подразумевает загрузку функциональных модулей после запуска приложения. Данный способ требует дополнительных трудозатрат, связанных с реализацией загрузчика функциональных модулей, однако обладает значительными преимуществами при сопровождении, так как позволяет гибко настраивать состав модулей приложения без перекомпиляции стартового модуля. Существует два варианта реализации загрузчика модулей. В первом случае загрузчик получает список модулей (например, из конфигурационного файла), после чего производит инициализацию модулей в указанном порядке. Процедура добавления нового модуля состоит из двух частей: копирование модуля в каталог, доступный загрузчику и регистрация модуля в конфигурации загрузчика. Во втором случае загрузчик из параметров конфигурации получает только целевые директории, после чего производит поиск доступных модулей по указанным директориям, определяет порядок их загрузки и производит инициализацию каждого модуля. Данный способ требует реализации алгоритма поиска и сортировки модулей, однако позволяет добавлять новые модули к уже развёрнутому приложению путём копирования модулей в целевой каталог приложения. Поиск и сортировка модулей согласно порядку их загрузки может основываться на конвенции именования модулей.

**Декомпозиция на подсистемы.**

Крупные распределённые приложения допускают разделения на несколько независимых подсистем. Необходимое условие такого разделения – сущности из пространства предметной области должны образовывать несколько полноценных, но независимых друг от друга групп. Таким образом, *подсистемой* мы будем называть блок функциональности (клиент-серверный для распределённых приложений), построенный на основе группы взаимосвязанных сущностей предметной области. В свою очередь, как клиентская, так и серверная часть подсистемы разделяется на модули. Необходимо понимать, что подсистема является крупной логической единицей, объединяющей весь функционал обработки группы сущностей, а модуль - физической единицей, реализующей небольшой, но целостный блок операций. Например, подсистема контроля состояния оборудования может содержать модули хранения информации, аналитических отчётов, визуализации текущего состояния.

Способ декомпозиции системы на подсистемы имеет много общего с декомпозицией яруса на модули. Существует главная подсистема, которая содержит реализацию функциональности, обязательной для использования во всех остальных подсистемах без изменений. Таковой функциональностью как на клиенте, так и на сервере, является:

* Безопасность приложения (система прав и ролей, авторизация и аутентификация пользователей)
* Обработка ошибок

Кроме того, главная подсистема может содержать интерфейс и базовую реализацию следующей функциональности:

* Логирование
* Кеширование
* Работа с базой данных
* Импорт-экспорт данных

Однако наличие данного функционала в главной подсистеме не является обязательным, а даже при его наличии его использование не является обязательным. Если одна из второстепенных подсистем содержит модуль, функционал которого требует весьма специфического ведения логов, вполне допустим отказ от использования стандартного логирования, предоставляемого главной подсистемой и реализация собственной функциональности логирования.

Стоит заметить, что реализовывать декомпозицию системы на подсистемы имеет смысл только в том случае, когда

**Структура слоя сервисов.**

Каждая подсистема, хотя и выстраивается поверх группы независимых сущностей, содержит ряд одинаковых функций. Эти функции связаны с хранением состояния сущностей на сервере, а также отображения этого состояния на клиенте. Каждая подсистема должна содержать так называемый «модуль справочников», который реализует функционал по созданию, изменению и удалению объектов, причём зачастую реализация этого функционала очень похожа. Каждая подсистема должна содержать сервисы для передачи объектов между клиентом и сервером. Поскольку в рамках подсистемы возможна передача множества разных типов сущностей, существует два способа организации сервисов:

1. Отдельный сервис для каждой сущности
2. Обобщенный сервис в рамках подсистемы для всех сущностей это подсистемы
3. Обобщенный сервис в рамках системы для всех сущностей всех подсистем

Рассмотрим достоинства и недостатки каждого решения с точки зрения удобства его дальнейшего сопровождения.

Решение «Отдельный сервис для каждой сущности» подразумевает создание целой линейки сервисов в рамках подсистемы. Контракт каждого сервиса данной линейки, определяющий его внешний интерфейс, будет очень близок к контрактам других сервисов линейки, зачастую отличаясь только типом обрабатываемых сущностей.

Достоинства данного решения:

* Гибкость (контракт каждого сервиса можно изменять независимо от контракта всех остальных сервисов)
* Простота реализации

Недостатки:

* Большое количество дублирующегося кода сервисов, которые отличаются только типом обрабатываемых сущностей
* В течение сопровождения возможны изменения как в контрактах, так и реализациях сервисов, которые невозможно синхронизировать ввиду независимости сервисов
* В рамках одного вызова сервиса допустима работа толькос одной сущностью

Решение «Обобщенный сервис в рамках подсистемы» подразумевает создание единого сервиса для работы со всеми сущностями подсистемы, список которых заранее известен.

Достоинства:

* Отсутствие дублирующегося кода сервисов в рамках подсистемы
* Поддержка групповых операций над всеми сущностями подсистемы (например, возможно одновременное сохранение двух разных сущностей)
* Упрощение добавления новых сущностей в подсистему, за счёт того, что для неё не нужно разрабатывать сервис

Недостатки:

* Снижение гибкости. Изменение контракта обобщенного сервиса означает изменение интерфейса работы со всеми сущностями подсистемы
* Применение данного решения может потребовать реализации механизма сериализации (преобразования сущностей в текстовый формат) и десериализации (восстановления из текстового формата) для передачи через обобщённый веб-сервис текстовых данных

Решение «Обобщенный сервис в рамках системы» похоже на решение «Обобщенный сервис в рамках подсистемы», но имеет одно отличие: обобщённый сервис создаётся в рамках главной подсистемы, и он используется всеми остальными подсистемами. Достоинства и недостатки данного решения совпадают с аналогичными для решения «Обобщенный сервис в рамках подсистемы». Реализация данного решения связана с дополнительной сложностью, поскольку список объектов, которые можно передавать через обобщённый сервис, неизвестен до момента запуска приложения и загрузки всех модулей.

Список литературы

1. *Гамма, Э.* Приемы объектно - ориентированного проектирования. Паттерны проектирования. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес // Пер. с англ. – СПб: Питер, 2001. – 368 с.
2. *Буч, Г.* Объектно-ориентированное проектирование с примерами применения / Г. Буч; пер. с англ. – М.: Конкорд. – 1992. – 519 с., ил.
3. *Фаулер, М.* Архитектура корпоративных программных приложений / М. Фаулер; пер. с англ. – М.: Вильямс. – 2006. – 544с.