shkolanauki.ru

Организация клиент – серверного взаимодействия при проектировании и разработке распределённых систем и приложений

Юрчик Петр Францевич, профессор кафедры АСУ
Андрющенко Всеволод Иванович, студент
Московский Автомобильно-Дорожный Государственный Технический Университет (МАДИ),
Российская Федерация, город Москва

Аннотация. *В данной статье рассматривается проблематика организации клиент – серверного взаимодействия при проектировании распределённых систем информационной поддержки жизненного цикла объектов (СИП ЖЦО), распределенных мобильных и web приложений. С целью успешного проектирования хорошо структурированных, легко масштабируемых и поддерживаемых клиент – серверных приложений предлагается использование архитектуры REST. Рассмотрены общие механизмы работы протокола HTTP/HTTPS, основные понятия и правила реализации REST API в распределенных системах информационной поддержки ЖЦО и клиент – серверных приложениях.*

Введение.

В настоящее время при проектировании и разработке современной системы информационной поддержки ЖЦО, а также практически любого, современного мобильного или web приложения стало необходимым внедрение серверного приложения. Синхронизация данных между приложениями, установленными на различных устройствах одного пользователя, загрузка данных из сети, перенос бизнес - логики приложения на серверную часть, обеспечение взаимодействия с другими пользователями или платформами с очевидностью требуют разделения ответственности между клиентским и серверным приложениями с организацией клиент - серверного взаимодействия по средствам интернет соединения.

Любое клиент - серверное приложение состоит, как минимум, из одного серверного и нескольких клиентских приложений. Клиентские приложения, как правило представляют собой мобильные или детекторные приложения, предназначенные для использования конечными пользователями, причем, именно данные приложения обеспечивают пользовательский интерфейс для формирования запросов к серверному приложению и отображения полученных данных. Серверное приложение обрабатывает полученные запросы и отправляет ответы на каждый из них, обеспечивает взаимодействие с базами данных, отвечает за авторизацию и аутентификацию пользователей.

Однако при проектировании современных распределённых систем информационной поддержки ЖЦО, выполняющих значительный объем различных функций и, как следствие, соответствующих запросов от клиентского приложения к серверному, требует создание определённых программных интерфейсов (API), обеспечивающих возможность работы распределенной системы, (в т.ч., СИП ЖЦО), с ее поддержкой и дальнейшим развитием.

Архитектура REST.

Одна из главных задач при проектировании и разработке любого современного клиент - серверного приложения или системы заключается в создание четко структурированной, надежной и легко масштабируемой архитектуры клиент серверного взаимодействия.

Именно создание **REST API** для обеспечения взаимодействия серверного и клиентских приложений с его дальнейшей реализацией на всех платформах додерживаемыми для использования клиентских приложений является одним из наиболее популярных методов реализации клиент - серверного взаимодействия**.** Данный архитектурный стиль проектирования получил широкое распространение среди разработчиков при проектировании современной клиент - серверной архитектуры.

REST (*Representational State Transfer*) – это архитектурный стиль проектирования различных распределенных систем и приложений, являющийся согласованным набором ограничений, учитываемых при проектировании распределенных систем. Транзакция, сформированная с использованием данного архитектурного стиля, в сети интернет представляет из себя определенный HTTP – запрос (GET, POST и другие), где необходимые данные передаются в качестве параметров запроса. Данная архитектура клиент – серверного взаимодействия имеет следующие особенности:

- Надёжность (за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
- Кэшируемость;
- Масштабируемость;
- Простота и единообразие программных интерфейсов;
- Лёгкость внесения изменений.

При проектировании и разработке распределённой системы информационной поддержки ЖЦО необходимо заранее провести анализ разрабатываемой системы, выявить наиболее востребованные задачи при работе с данной системой или приложением, стандартизировав каждую из задач согласно базовым операциям модели CRUD. Данные базовые операции реализуются в проектируемом REST API и выполняют следующие действия:

- Получение данных в удобном для клиента формате (как правило JSON);

• Передача (создание) новых данных;
• Обновление данных;
• Удаление данных.

В настоящее время существует минимум восемь методов HTTP/HTTPS запросов: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATH. Однако для реализации архитектуры REST на практике применяются следующие методы протокола HTTP, каждый из которых отвечает за свои, ранее оговорённые действия:

• **GET** – получение данных;
• **POST** – создание и передача данных;
• **PUT** – обновление, модификация уже существующих данных;
• **DELETE** – удаление данных.

Таким образом, одна транзакция по такому API будет состоять, как минимум, из следующего:

• **Тип метода запроса**,
• **Путь запроса**,
• **Тело запроса**,
• **Код ответа,**
• **Тело ответа.**

Практическое использование REST архитектуры при проектировании распределенных систем.

Рассмотрев базовые понятия и основные компоненты, из которых состоит каждый из методов проектируемого серверного REST API, необходимо выработать общие ограничения, применяемые при проектировании распределённых систем информационной поддержки ЖЦО, различных мобильных и web приложений, различных распределенных систем. Данные ограничения могут меняться в зависимости от текущего проекта, поэтому важно, чтобы стилистика организации архитектуры клиент-серверного взаимодействия оставалась неизменной, при реализации серверного API всего проекта.

Необходимо обеспечить единую стилистику формирования ответов при обработке клиентских запросов. Например, каждый из ответов должен содержать следующие обязательные поля: статус операции, поле возвращаемых данных, поле ошибки.

Пример данных JSON сформированных с использованием данного правила:

```
{
"status":true,
"result":{ данные возвращаемые при удачном выполнении запроса},
}
```

– при успешном выполнение определенного запроса,

```
{
"status":false,
"error": {"код ошибки или/и сообщение об ошибке"},
}
```

- при возникновении ошибок во время выполнения определённого запроса.

Использование единого формата маршрута каждого из запросов при проектировании системы. Так, запрос по одному адресу в зависимости от его типа (GET, POST и другие) и наличия тех или иных параметров может выполнять различные действия. Пример формирования маршрутов выполняемых запросов приведён в табл. 1:

Таблица 1. Формирование маршрута HTTP запроса.

Адрес запроса: https://mySite.com/api/news			
GET	POST	DELETE	UPDATE
Получить список новостей	Добавить новость	Не используется	Не используется
Адрес запроса: https://mySite.com/api/news/134			
GET	POST	DELETE	UPDATE
Получить подробную информацию по данной новости	Добавить в избранное	Удалить данную новость	Внести изменения в данную новость

При необходимости внесения каких-либо изменений в работу серверного API, уже доступного широкой аудитории проекта, категорически запрещено изменять логику работы ранее добавленных программных интерфейсов для изменения функциональных возможностей. В этом случае необходимо либо создавать новую версию API для всего программного продукта с обеспечением работы всех версий API, либо добавлять отдельные методы в уже созданном API, которые никак не повлияют на работу ранее созданных.

Необходимо обеспечить кэширование данных, запрашиваемых из базы данных на серверном приложении, и, по возможности, обеспечить кэширование данных получаемых клиентскими приложениями.

Резюме.

Опыт показывает, что примеры проектирования и разработки API клиент – серверного взаимодействия с использованием архитектуры REST при проектировании распределенных систем информационной поддержки ЖЦО, мобильных приложении и различных распределенных приложений могут отличаться от проекта к проекту в зависимости от требований каждого проекта.

Необходимо при этом отметить, что отсутствие четких стандартов по проектированию и необходимость формирования списка «ограничений» для каждого из проектов является единственным существенным недостатком данного подхода к проектированию и реализации клиент – серверного взаимодействия.

При этом очевидно, что внедрение REST API, как определённого «стандарта» для каждого из проектов, позволяет четко структурировать все методы клиент – серверного взаимодействия, упростить поддержку и развитие данного проекта, повысить надежность работы разрабатываемой системы в целом.

Фактически, использование REST API может рассматриваться как современное стандартное решение организации клиент – серверного взаимодействия в распределенных системах информационной поддержки ЖЦО, в иных распределенных системах и приложениях, благодаря своей гибкости и простоты реализации.