электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.4

Архитектура клиент-серверных мобильных приложений

***Устимов А.И.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н, доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gapyu@bmstu.ru

Введение

Разработка клиент-серверных мобильных приложений во многом построена на компромиссах. При решении этой задачи необходимо выбрать формат передачи данных и способ оперирования полученной информацией. Требуется выяснить, должно ли приложение иметь возможность работы при отсутствии интернет-соединения и, в случае положительного решения, определить способы хранения данных и спектр вычислений на стороне клиента.

Рассматриваемая архитектура подразумевает наличие некоторого веб-сервера, с которым программа на мобильном устройстве будет обмениваться информацией. В контексте данной статьи, реализация сервера не так важна. Это может быть как популярный веб-фреймворк, так и статическая HTML страница. Важно понимать, что в последнем случае сервер не сможет производить каких-либо вычислений по запросу пользователя.

По приведённому выше описанию можно построить некоторую обобщённую схему взаимодействия компонентов реализуемой системы (рис. 1), декомпозиция которой будет выполняться на протяжении всей статьи.

Рис. 1. Обобщённая схема взаимодействия компонентов системы

В целом, данная схема ясна. Мобильное устройство передаёт запрос на сервер, а в ответ получает необходимые данные. Но она не подсказывает решения ни на одну из поставленных выше задач. Данная статья преследует цель предложить некоторые варианты решения, используемые автором.

Формат передачи данных

Во-первых, для однозначного трактования передаваемых данных необходимо определиться с форматом их обмена. Можно выделить как минимум три основных принципиально отличающихся варианта представления пересылаемой информации.

1. Страница в формате HTML.
2. Двоичные форматы обмена данными (Protocol Buffers, BSON и т. д.).
3. Текстовые форматы обмена данными (JSON, XML и т. д.).

Первый вариант требует синтаксического разбора HTML-страниц, отдаваемых сервером. Данный способ является наиболее прямолинейным и годится только в тех случаях, когда серверная часть разработана третьими лицами и к исходному коду нет доступа.

В этом случае, задача решается загрузкой на мобильное устройство полной HTML-страницы и выполнением её синтаксического разбора с целью извлечения необходимой информации. Данный подход имеет серьёзные недостатки и его стоит использовать только в крайних случаях, когда невозможно применить другие представленные решения. Остальные описываемые подходы к проектированию архитектуры клиент-серверных мобильных приложений предполагают возможность изменения реализации серверной части системы.

Основные недостатки передачи данных с помощью HTML-страниц:

- требуется загрузка большого объёма данных (по сравнению с остальными представленными методами);
- небольшое изменение в структуре или вёрстке веб-сайта может привести к отказу всего мобильного приложения;
- реализация алгоритмов синтаксического разбора (даже с учётом использования готовых библиотек), ощутимо сложнее использования специализированных форматов для обмена данными.

Второй вариант специально предназначен для решения задач передачи информации. Двоичные форматы обмена данными обладают высокой скоростью сериализации и десериализации, а полученные в результате первой операции строки очень

компактны по сравнению с текстовыми форматами (некоторые форматы представляют собой исключения) [2]. Выбор в пользу данного варианта стоит сделать в случае реализации клиент-серверной архитектуры с очень высокой нагрузкой. К сожалению, данные сериализованные в бинарный формат не поддаются чтению человеком.

Третий вариант отлично подходит для веб-приложений с средней и высокой нагрузкой, к которым можно отнести крупные интернет-магазины. Текстовые форматы обмена данными являются более компактными по сравнению с HTML, но, в то же время, более объёмными по сравнению с двоичными форматами обмена данными. Тем не менее, текстовые форматы обладают важным свойством — человек может прочитать их и понять содержимое, что значительно упрощает отладку приложения.

Рассмотрим небольшой класс на языке программирования C#, представляющий собой абстракцию товара. Данный класс содержит следующие поля: наименование, категория и цена.

```
public class Product
{
	public string name;
	public string category;
	public int cost;

	public Product()
	{
		this.name = "bicycle";
		this.category = "sport";
		this.cost = 1200;
	}
}
```

При создании объекта данного класса, его поля автоматически инициализируются значениями, заданными в конструкторе, т. е. bicycle для наименования, sport для категории и 1200 для цены.

Теперь рассмотрим что получится при сериализации объекта данного класса. Ниже представлен текст, сгенерированный сериализатором XML.

```
<?xml version="1.0" encoding="cp866"?>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <name>bicycle</name>
  <category>sport</category>
  <cost>1200</cost>
```

```
</Product>
```

Аналогичный текст для JSON следует далее.

```
{
	"name": "bicycle",
	"category": "sport",
	"cost": 1200
}
```

Хотя оба формата хорошо читаются, заметно, что JSON более компактен. Поэтому, по мнению автора статьи, данный текстовый формат наиболее удобен для использования в реализации клиент-серверных мобильных приложений и далее будет предполагаться, что для обмена информацией выбран именно он. Тем не менее, существует множество других форматов обмена данными, и веб-приложение может поддерживать одновременно несколько из них, отдавая нужные данные в зависимости от параметров запроса.

Оперирование данными

Для удобного оперирования полученной информацией, на мобильном устройстве должны быть описаны такие же модели данных, как и в веб-приложении. Описание выполняется в виде классов, подобных представленному ранее Product.

Для сериализации объектов класса на сервере и десереализации на клиенте существуют различные инструменты, разработанные практически для каждого языка программирования. В качестве примера, рассмотрим библиотеку Json.NET для C#.

Допустим, в ответ на некоторый запрос мобильного приложения, от сервера пришёл текст, представленный ниже.

```
[
	{ "name": "bicycle", "category": "sport", "cost": 1200 },
	{ "name": "tent", "category": "camping", "cost": 800 },
	{ "name": "kayak", "category": "sport", "cost": 1500 }
]
```

С использованием упомянутой выше библиотеки, преобразование этого текста в формате JSON, представляющего собой массив товаров, в список объектов класса Product выполняется в одну строку кода [3], следующую далее.

```
List<Product> p = JsonConvert.DeserializeObject<List<Product>>(data);
```

После этого, объектами из списка можно манипулировать всеми способами, предоставляемыми языком программирования.

Таким образом, взаимодействие сервера и клиента можно описать следующим образом. Клиент отправляет на сервер запрос с необходимыми параметрами. Сервер, исходя из параметров запроса, выбирает необходимую информацию из базы данных и возвращает её в сериализованном виде в качестве ответа. Клиент получает данные в виде строки и десериализует её, отображая на модель данных. В результате отображения, образуются объекты используемого языка программирования, над которыми выполняются дальнейшие операции.

Работа без интернет-соединения

Следующая дилемма, которую необходимо решить при разработке клиент-серверного мобильного приложения — ответить на вопрос о том, должно ли оно работать без соединения с интернетом.

Решение в пользу отказа от поддержки такой возможности предпочтительней в случаях, когда информация на сервере очень динамична и отображение неактуальных данных может оказать пагубное влияние на удовлетворение потребностей пользователей. Например, это могут быть приложения для отслеживания курса валют или состояния фондовых бирж.

С другой стороны, возможность работы в отсутствие связи с сервером подходит для приложений, в которых информация обновляется не слишком часто. В качестве дополнительного преимущества от возможности работать с приложением без интернет-соединения стоит отметить возможность повышения конверсии. Так, запустив единожды мобильный клиент интернет-магазина и загрузив каталоги, пользователь позднее неоднократно может просматривать их оффлайн и делать заказы, например, с помощью телефонного звонка. Конечно, в данном примере есть и отрицательные стороны, такие как отсутствие актуальной информации о ценах и наличии товаров, но данные вопросы предполагается уточнить при разговоре с оператором.

В итоге, при выборе модели клиента требующего наличие интернет-соединения, при каждом запуске приложения выполняется запрос к серверу для получения актуальных данных. В случае отсутствия соединения, приложение не может отобразить никакой полезной информации, поэтому единственный выход — уведомить пользователя и завершить работу. В аналогичной ситуации, приложение работающее без интернет-соединения может принести пользу пользователю и без доступа к серверу.

Хранение информации

При выборе модели клиента не требующего интернет-соединения, возникает проблема хранения данных. Необходимо ответить на вопросы о том, где и как сохранить полученную от сервера информацию, чтобы она не была утеряна при закрытии приложения и была загружена при повторном его открытии.

Рассмотрим решение данной задачи на примере операционной системы для мобильных устройств Android. Данная операционная система поддерживает пять способов хранения данных [4]:

- пары ключ-зачение;
- внутренняя память;
- внешняя память (флеш-карта);
- реляционная база данных SQLite;
- сторонний веб-сервер.

Очевидно, что вариант со сторонним веб-сервером противоречит условию задачи, т. к. предполагается отсутствие интернет-соединения.

Первый вариант не подходит, т. к. позволяет сохранять только примитивные типы данных, такие как: булевы значения, числа с плавающей точкой, целые числа и строки. В то время как требуется сохранять сложные структуры данных и массивы из этих структур.

Оставшиеся претенденты на роль хранилища данных удовлетворяют всем требованиям и будут рассмотрены подробнее.

Принципы работы с данными записанными во внешнюю и внутреннюю память устройства аналогичны. Стоит лишь упомянуть, что при наличии флеш-карты предпочтительней сохранять данные на неё, нежели во внутреннюю память. Это объясняется тем, что, как правило, внутренняя память имеет меньший объём и используется системой.

При работе с данными видами памяти, полученную от сервера информацию предполагается сохранять в обычных текстовых файлах в сериализованном формате без дополнительных преобразований. Таким образом, при последующем запуске приложения, данные загружаются из файлов и десереализуются в объекты, так же как и при получении с сервера. Например, для интернет-магазина это могут быть каталоги продукции. Изображения и другие мультимедийные данные сохраняются в виде файлов соответствующих форматов.

Рассмотренный выше подход будет хорошо работать на данных относительно

небольшого объёма. При увеличении количества информации возникает вопрос о том, загружать ли несколько тысяч десериализованных объектов в оперативную память приложения, либо разбивать эти несколько тысяч на более мелкие файлы и загружать по отдельности. Оба подхода далеки от идеала. Как раз для подобных случаев в Android предусмотрена реляционная база данных SQLite. Таким образом, все получаемые данные можно записывать в базу, а потом, по мере необходимости, извлекать точно требуемое количество информации с помощью SQL запросов. Но в данном случае возникают новые сложности в виде необходимости написания дополнительного кода для извлечения данных из объектов с целью записи в таблицы, а также для обратного преобразования.

Распределение вычислений

Следующей проблемой, которая возникает особенно остро в случае поддержки приложением работы без интернет-соединения, является определение того, какие вычисления нужно производить на стороне клиента.

В случае, когда приложение не может работать без интернет-соединения, трудно с полной уверенностью выделить какие-либо аспекты, требующие вычислений на мобильном устройстве. Поэтому ответ на поставленный вопрос с данной точки зрения лучше всего дать, опираясь на показатели производительности того или иного решения.

С другой стороны, если приложение задумано с возможностью работы оффлайн, то пользователь скорее всего будет ожидать, что оно может произвести некоторые вычисления без наличия интернет-соединения. Поэтому необходимо определить спектр данных вычислений и по возможности реализовать их в приложении. Для уже рассмотренного ранее примера интернет-магазина, реализация логики работы корзины на клиенте может стать значительным преимуществом. Пусть пользователь и не сможет сделать заказ напрямую из приложения, но ему будет значительно удобнее заполнить корзину в приложении и получить вычисленную сумму заказа, нежели выписывать необходимые товары на бумаге, а после чего высчитывать общую стоимость.

Последнее решение может оказать влияние на безопасность приложения, поэтому на реализацию оффлайн функционала стоить обратить особое внимание и отнестись с осторожностью.

Заключение

В соответствии с информацией, изложенной в статье, исходную схему взаимодействия компонентов системы (рис. 1) можно уточнить, как показано на рис. 2.

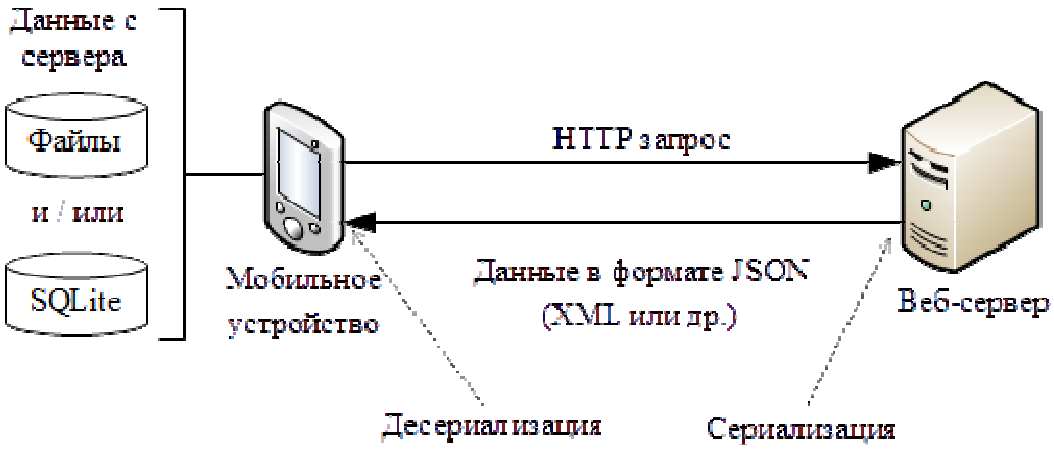

Рис. 2. Уточнённая схема взаимодействия компонентов системы

Выбор в пользу того или иного решения рассмотренных задач будет сильно варьироваться в зависимости от предметной области мобильного приложения.

Двоичные форматы обмена данными отлично подходят в случае высокой нагрузки на сервер и необходимости максимально уменьшить размер передаваемой информации. В остальных случаях, для удобства, можно остановиться на текстовых форматах.

Для однозначной интерпретации передаваемой информации и удобства работы с ней, на клиенте и сервере должны быть описаны одинаковые модели данных. Перед передачей с сервера объекты должны быть сериализованы, а после получения на клиенте - десериализованы.

Для многих клиент-серверных мобильных приложений, может оказаться значительным преимуществом возможность работы без наличия интернет-соединения. Поэтому, необходимо продумать, как наиболее подходящим образом обеспечить долговременное хранение информации полученной от сервера, и определить спектр вычислений, которые желательно произвести на стороне клиента.

Список литературы

1. Gapanyuk Yu., Lakomkin E., Ionkin S., Davtyan M. MVC WEB Framework based on eXist application server and XRX architecture // 7th Spring Researchers Colloquium on Databases and Information Systems (Moscow, 02-03 june 2011): proceedings. Moscow, 2011. P. 19-25.
2. Comparing Protobuf, JSON, BSON, XML with .NET for File streams. Available at: https://damienbod.wordpress.com/2014/01/09/comparing-protobuf-json-bson-xml-with-net-for-file-streams/, accessed 10.03.2015.
3. Serializing Collections. Режим доступа: http://www.newtonsoft.com/json/help/html/

SerializingCollections.htm, accessed 16.03.2015.

4. Storage Options. Available at: http://developer.android.com/guide/topics/data/data-storage.html, accessed 10.03.2015.
5. How to serialize an object to XML by using Visual C#. Available at: https://support.microsoft.com/en-us/kb/815813, accesse 16.03.2015.