# Создание многозвенного программного комплекса для интерактивного обучения
**Воронин О. Д.**

*Воронин Олег Дмитриевич / Voronin Oleg Dmitrievich – студент магистратуры,*
*кафедра информатики и методики преподавания информатики,*
*факультет информатики и вычислительной техники,*
*Нижневартовский государственный университет, г. Нижневартовск*

***Аннотация:*** *в контексте данной статьи рассмотрен принцип работы и разработки многозвенной архитектуры. А также приведены преимущества некоторых технологий реализации многозвенного программного комплекса для интерактивного обучения.*

***Ключевые слова:*** *информационная система, многозвенная архитектура, разработка сервиса, сервлет, REST, Spring, Java EE.*

С каждым днём средства интерактивного обучения требуют все больше производительности. Вызвано это большими требованиями к интеграции различных технологий, помогающих визуализировать предоставляемую информацию. Создание средства интерактивного обучения подразумевает наличие не

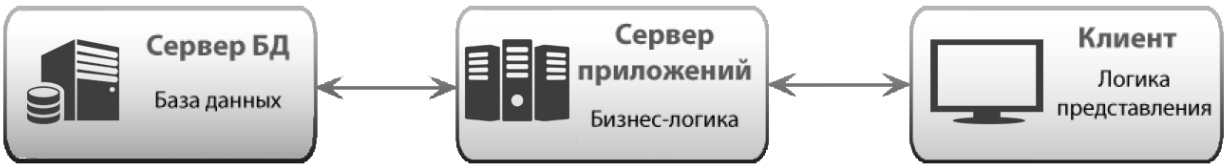


*Рис. 1. Схема многозвенной архитектуры*

только текстов, но и медиа данных, будь то видео или аудио. Проектный анализ требует определения типов информации и её размещения на носителях таким образом, чтобы оптимизировать скорость доступа при условии одновременного использования множества типов. Также влияние на проектирование архитектуры оказывают такие вещи как: объем аудитории, использующей программный продукт, возможность обновления программного продукта. Если программное обеспечение имеет множество медиа данных, которые требуют обновления, то существует потребность запуска на множестве систем с приемлемой производительностью, и это следует учитывать при проектировании многозвенного программного комплекса.

В данной статье рассмотрен принцип работы и разработки многозвенной архитектуры, описание связи между звеньями. Примеры кода представлены на языке программирования Java.

Многозвенная архитектура предполагает наличие следующих компонентов приложения (Рис. 1): клиентское приложение (клиент), сервер приложений и сервер базы данных [3].

Сервер базы данных обеспечивает хранение данных и вынесен на первый уровень. Он представляет собой базу данных вместе с хранимыми процедурами, триггерами и схемой, описывающей приложение в терминах реляционной модели [3].

Сервер приложений расположен на среднем уровне, где сосредоточена большая часть бизнес-логики [3].

Клиентское приложение – это интерфейсный компонент, который представляет последний уровень, предназначенный для конечного пользователя. Клиентское приложение не имеет прямых связей с базой данных (по требованиям безопасности) и не нагружен бизнес-логикой (по требованиям масштабируемости) [3].

Преимущества многозвенной архитектуры: увеличение производительности (так как бизнес-логика выполняется не на клиентском компьютере), легкая масштабируемость программного продукта, отказоустойчивость при высоких нагрузках.

Примером таких систем в среде интерактивного обучения может являться продукт «Lingualeo,» – образовательная платформа для изучения и практики иностранного языка. Клиентским приложением у продукта является, прежде всего, браузер, а также приложения для мобильных платформ.

Многозвенным приложением может являться и веб сервис, так как клиентом является веб браузер, сервером приложений может являться сервлет (Рис. 2) контейнера JavaEE, а вместо сервера баз данных может быть файловое хранилище, к которому обращается сервлет.

Рассмотрим основные функции клиента и сервера приложений.

Функции приложения-клиента [1]:

1. Посылка запросов серверу.
2. Интерпретация результатов запросов, полученных от сервера.
3. Представление результатов пользователю в некоторой форме (интерфейс пользователя).

Функции серверной части [1]:

1. Прием запросов от приложений-клиентов.
2. Интерпретация запросов.
3. Оптимизация и выполнение запросов к БД.
4. Отправка результатов приложению-клиенту.
5. Обеспечение системы безопасности и разграничение доступа.
6. Управление целостностью БД.
7. Реализация стабильности многопользовательского режима работы.

Классическое взаимодействие между клиентом и сервером осуществляется через протокол TCP/IP, а

```
public class HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
     throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.write("<!DOCTYPE html>\n"+
        "<html>\n" +
        "<head><title> Пример простейшего сервлета</title></head>\n" +
        "<body bgcolor=\"#fdf5e6\">\n" +
        "<p>Эта страница создана с помощью сервлета</p>\n" +
        "</body></html>");
    }
}
```

*Рис. 2. Пример простейшего сервлета*

именно с помощью сокетов. Как известно, для взаимодействия между машинами по протоколу IP используются адреса и порты. Первое на текущий момент представляет из себя 32-битный адрес, наиболее часто его представляют в символьной форме mmm.nnn.ppp.qqq (адрес, разбитый на четыре октета по одному

```
public class SampleSocket {
    public static void main(String[] args) {
        try {
            //открываем сокет и коннектимся к localhost:3128
            //получение сокета сервера
            Socket s = new Socket("localhost", 3128);
            //берем поток вывода и выводим туда первый аргумент
            //заданный при вызове адрес открытого сокета и его порт
            args[0] = args[0] +"\n" + s.getInetAddress().getHostAddress()
                    +":"+s.getLocalPort();
            s.getOutputStream().write(args[0].getBytes());
            //читаем ответ
            byte buf[] = new byte[64*1024];
            int r = s.getInputStream().read(buf);
            String data = new String(buf, 0, r);
            //выводим ответ в консоль
            System.out.println(data);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            System.out.println(e); //вывод исключений
        }
    }
}
```

*Рис. 3. Пример простейшего взаимодействия с помощью сокетов*

байту в октете и разделённый точками). Второе — это номер порта в диапазоне от 0 до 65535. Эта пара и есть сокет («гнездо», соответствующее адресу и порту). В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя (Рис. 4). Например, при обращении к серверу на HTTP порт сокет будет выглядеть так: 127.0.0.1:3128 или localhost:3128 (Рис. 3), а ответ будет поступать на mmm.nnn.ppp.qqq: xxx [5].

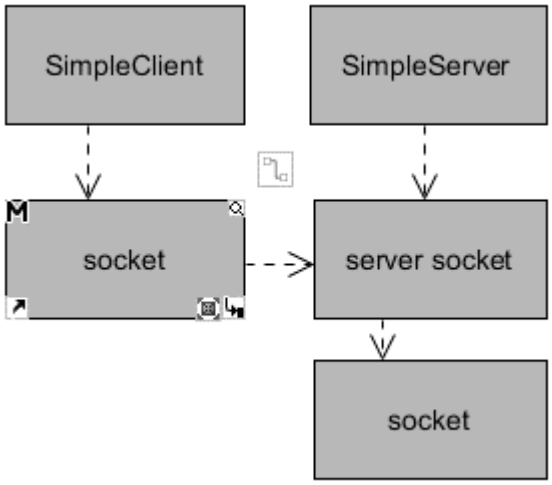


*Рис. 4. Логическая структура работы между клиентом и сервером*

Работа с сокетами подразумевает наличие открытого порта на сервере и клиенте, что может быть не совсем удобно, если порт занят или если у клиента закрыто обращение к некоторым портам. Поэтому лучшим решением для взаимодействия между клиентом и сервером может стать работа по API сервера, используя HTTP. Таким API является REST.

REST (Representational state transfer) — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В нашем случае это приводит к повышению производительности и упрощению архитектуры. Компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют REST-запрос), а необходимые данные передаются в качестве параметров запроса. Серверы не связаны с интерфейсами клиентов и их состояниями. Клиент отсылает запросы, когда готов совершить транзакцию на изменение состояния. Каждый из клиентов, а также промежуточные узлы между сервером и клиентами, могут кэшировать ответы сервера. Правильное использование кэширования в REST-архитектуре устраняет избыточные клиент-серверные взаимодействия, что улучшает скорость и расширяемость системы [9].

Пример такой работы описан в следующем рисунке (Рис. 5). В нем часть кода приложения, отвечающего за GET запросы, позволяет показывать необходимую нам информацию.

В результате первый метод при переходе в браузере по адресу http://localhost/developers/ покажет нам страницу с надписью «duchess,duke». Второй метод используется, когда нужно передать параметры http://localhost/developer/first-last. В результате этого запроса сервер покажет нам введенные параметры (Рис. 6). Таким образом можно работать с сервером и его бизнес логикой, используя простые и понятные

```
@Path("developer")
public class DevelopersResource {
    @GET
    public String developers(){
        return "duchess,duke";
    }
    @GET
    @Path("{first}-{last}")
    public String developer(@PathParam("first") String first
            ,@PathParam("last") String last){
        return f
    }
}
```

```
<developer>
    <firstname>first</firstname>
    <last>last</last>
</developer>
```

*Рис. 5. Класс, отвечающий на GET запросы*

```
<developer>
    <firstname>first</firstname>
    <last>last</last>
</developer>
```

*Рис. 6. Вывод XML при передаче параметров в запросе*

HTTP запросы, что значительно упрощает взаимодействие.

Говоря о масштабируемости многозвенного приложения стоит упомянуть, что экземпляров сервера приложений может быть множество. Достигается это путем размещения его в облачном сервисе, где каждый сервер приложений находится в своей виртуальной машине. Управление же количеством нагрузки на серверы и перенаправление трафика берет на себя отдельный сервер. Таким образом, в случае наплыва пользователей весь трафик будет равномерно распределен между виртуальными машинами, либо будут созданы новые экземпляры виртуальных машин при недостатке ресурсов у существующих, что обеспечит отказоустойчивость. Такими свойствами обладают все сервисы публикации облачных приложений, такие как: Windows Azure (Рис. 7), и Amazon S3.

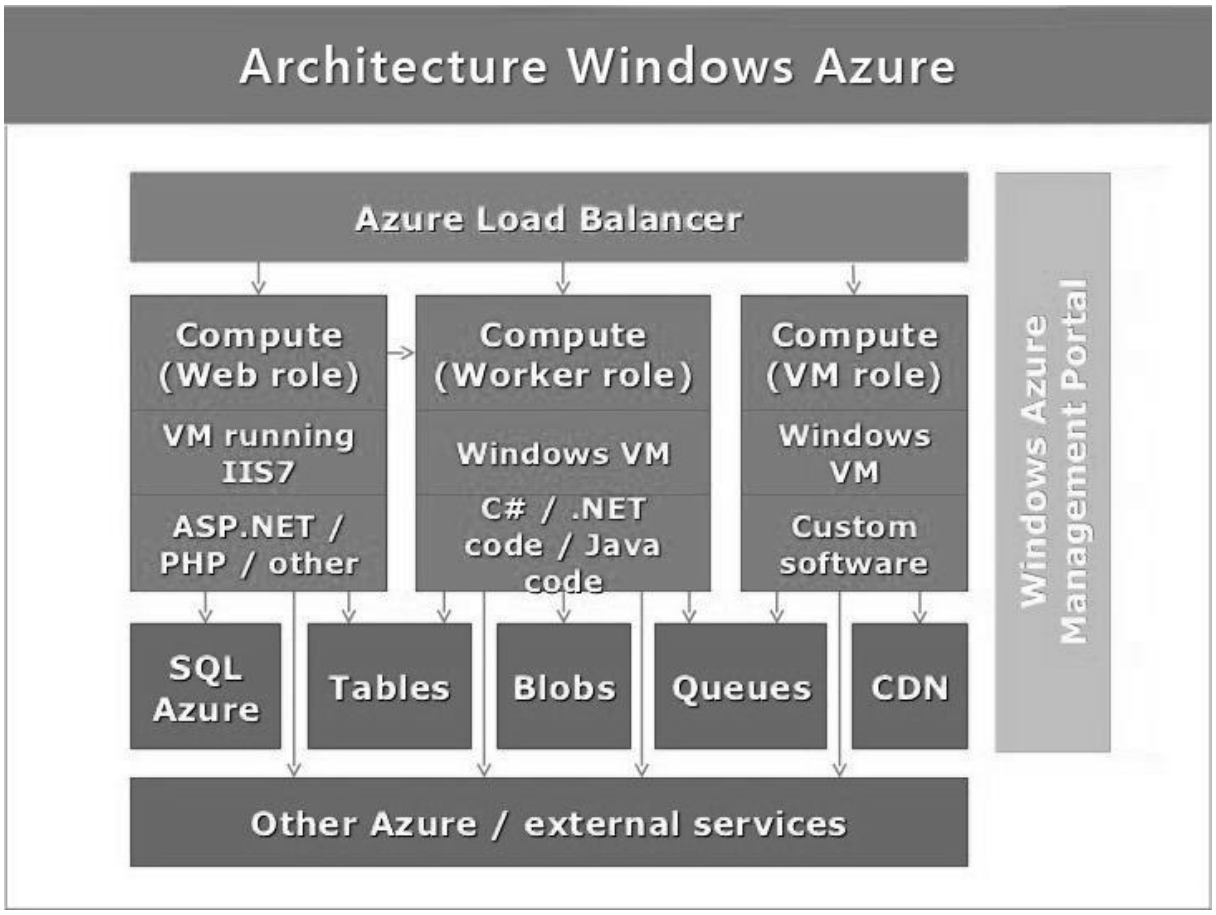


*Рис. 7. Архитектура облачного сервиса Windows Azure*

***Литература***


1. Технология Клиент-Сервер. [Электронный ресурс]: Интуит. URL: http://www.intuit.ru/studies/courses/508/364/lecture/8643?page=2 (дата обращения: 15.02.2016).
2. Начало работы с веб-службами RESTful. [Электронный ресурс]: NetBeans help. URL: https://netbeans.org/kb/docs/websvc/rest_ru.html (дата обращения: 15.02.2016).
3. Клиент/Сервер (Client/Server). [Электронный ресурс]: Cnews. URL: http://cioguru.cnews.ru/glossary/byid/42 (дата обращения: 15.02.2016).

4. Информационная система. [Электронный ресурс]: Википедия. URL: https://ru.wikipedia.org/wiki/Информационная_система (дата обращения: 15.02.2016).
5. Сокет (программный интерфейс). [Электронный ресурс]: Википедия. URL: https://ru.wikipedia.org/wiki/Сокет_(программный_интерфейс) (дата обращения: 15.02.2016).
6. Создание RESTful Web-сервиса. [Электронный ресурс]: Spring Project. URL: http://spring-projects.ru/guides/rest-service/ (дата обращения: 15.02.2016).
7. Описание простейшего сервлета. [Электронный ресурс]: Technerium. URL: http://www.technerium.ru/tehnologiya-java-servlet/kak-napisat-prostoy-servlet-v-ide-eclipse-tomcat-v7 (дата обращения: 15.02.2016).
8. Работа с TCP/IP в Java. Сокеты. [Электронный ресурс]: Javaportal.ru. URL: http://www.javaportal.ru/java/articles/java_http_web/article02.html (дата обращения: 15.02.2016).
9. REST. [Электронный ресурс]: Википедия. URL: https://ru.wikipedia.org/wiki/REST (дата обращения: 15.02.2016).
10. Cloud for Developers: Azure vs. Google App Engine vs. Amazon vs. AppHarbor. [Электронный ресурс]: Slideshare.net. URL: http://www.slideshare.net/nakov/cloud-for-developers-azure-vs-google-app-engine-vs-amazon-vs-appharbor (дата обращения: 15.02.2016).