Трехуровневая архитектура

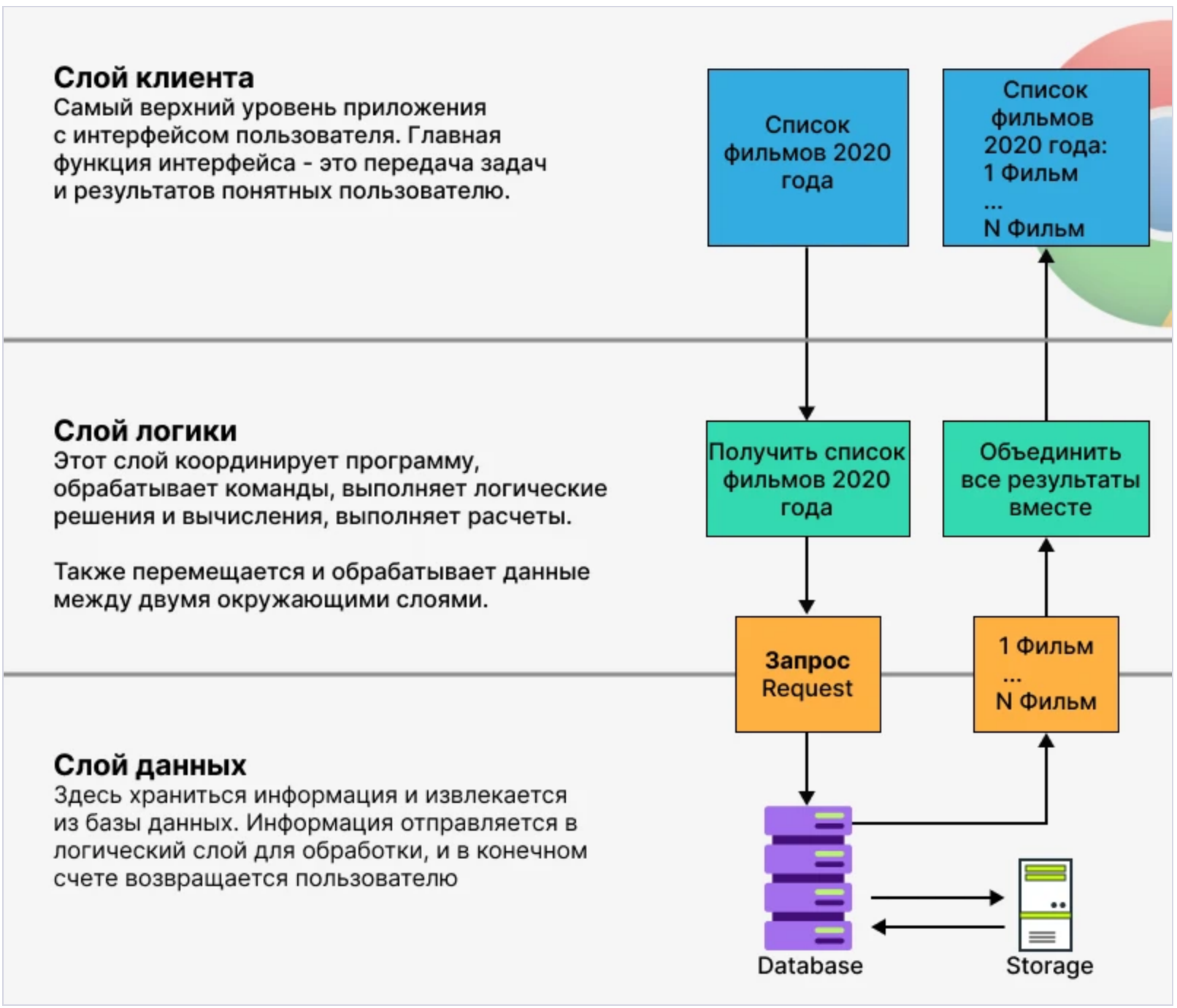
JSP & Servlets
14 уровень, 1 лекция

ОТКРЫТА

Знакомство с трехуровневой архитектурой

Трехуровневая архитектура — это самая распространенная архитектура взаимодействия в интернете. Она появилась, когда серверную часть двухуровневой разделили на две части: **слой логики** и **слой данных**.

Выглядеть это стало примерно так:



Слой клиента — это часть «распределенного приложения», которая отвечает за взаимодействие с пользователем. Этот слой не должен содержать бизнес-логики и не должен хранить критически важные данные. Также он не должен взаимодействовать со слоем базы данных напрямую, а только через слой бизнес-логики.

Однако какая-то логика тут все же есть. Во-первых, это взаимодействие с пользователем через интерфейс, валидация вводимых им данных, работа с локальными файлами. Еще сюда можно отнести все, что касается авторизации пользователя и шифрование данных при работе с сервером.

Во-вторых, это несложная бизнес логика. Например, прислал интернет-магазин список товаров, на стороне клиента мы их можем отсортировать, отфильтровать. И примитивное хранение данных тут тоже есть: кэширование, куки залогиненого пользователя и тому подобное.

Слой бизнес-логики располагается на втором уровне, на нем сосредоточена бóльшая часть бизнес-логики. Вне его остаются только фрагменты, экспортируемые на клиента, а также элементы логики, погруженные в базу данных (хранимые процедуры и триггеры).

Очень часть сервера бизнес-логики содержат не только эту самую логику, но и решают задачи масштабирования: код разбивается на части и разносится по различным серверам. Некоторые особо востребованные сервисы могут запускаться на десятках серверов. Нагрузкой между ними управляет load balancer.

Серверные приложения обычно проектируются таким образом, чтобы легко можно было запустить еще одну копию сервера и она начала работать в кооперации с другими его копиями. То есть даже в процессе написания серверного кода у тебя никогда не будет гарантий, что твой статический класс запущен в единственном экземпляре.

Слой данных обеспечивает хранение данных и выносится на отдельный уровень, реализуется, как правило, средствами систем управления базами данных (СУБД), подключение к этому компоненту обеспечивается только с уровня сервера приложений.

Причины отделения слоя данных

Отделение слоя данных в полноценный третий слой произошло по многим причинам, но самая главная — это возросшая нагрузка на сервер.

Во-первых, **базы данных стали требовать много памяти** и процессорного времени на обработку данных. Поэтому их повсеместно стали выносить на отдельные сервера.

Бэкенд при возросшей нагрузке можно было легко дублировать и поднять десять копий одного сервера, а дублировать базу данных было нельзя — база все еще оставалась единым и неделимым компонентом системы.

Во-вторых, **базы данных стали умными** — у них появилась собственная бизнес-логика. Они стали поддерживать хранимые процедуры, триггеры, собственные языки типа PLSQL. И даже появились программисты, которые стали писать код, выполняемый внутри СУБД.

Всю логику, которая была не завязана на данные, выносили в бэкенд и параллельно запускали на десятках серверов. Все критично завязанное на данных оставалось внутри СУБД и там уже проблемы выросшей нагрузки приходилось решать своими методами:

- Кластер базы данных — группа серверов БД, которые хранят одни и те же данные и синхронизируют их по определенному протоколу.
- Шардирование — данные дробятся на логические блоки и разносятся по разным серверам БД. Очень сложно поддерживать изменения БД при таком подходе.
- Подход NoSQL — хранение данных в БД, которые построены для хранения огромного количества данных. Это часто даже и не базы, а специфические файловые хранилища. Очень бедный функционал по сравнению с реляционными базами данных.
- Кэширование данных. Вместо простого кэша на уровне базы данных появились целые кеширующие СУБД, которые хранили результат только в памяти.

Понятно, что для управления этим зоопарком серверных технологий нужны были отдельные люди и/или целые команды, что и привело к вынесению слоя данных в отдельный слой.

Важно! Все передовые технологии рождаются в недрах крупных ИТ-корпораций, когда старые подходы перестают справляться с новыми вызовами. Вынесение баз данных в отдельный слой придумал не какой-нибудь программист, а группа инженеров где-нибудь в недрах Oracle или IBM.

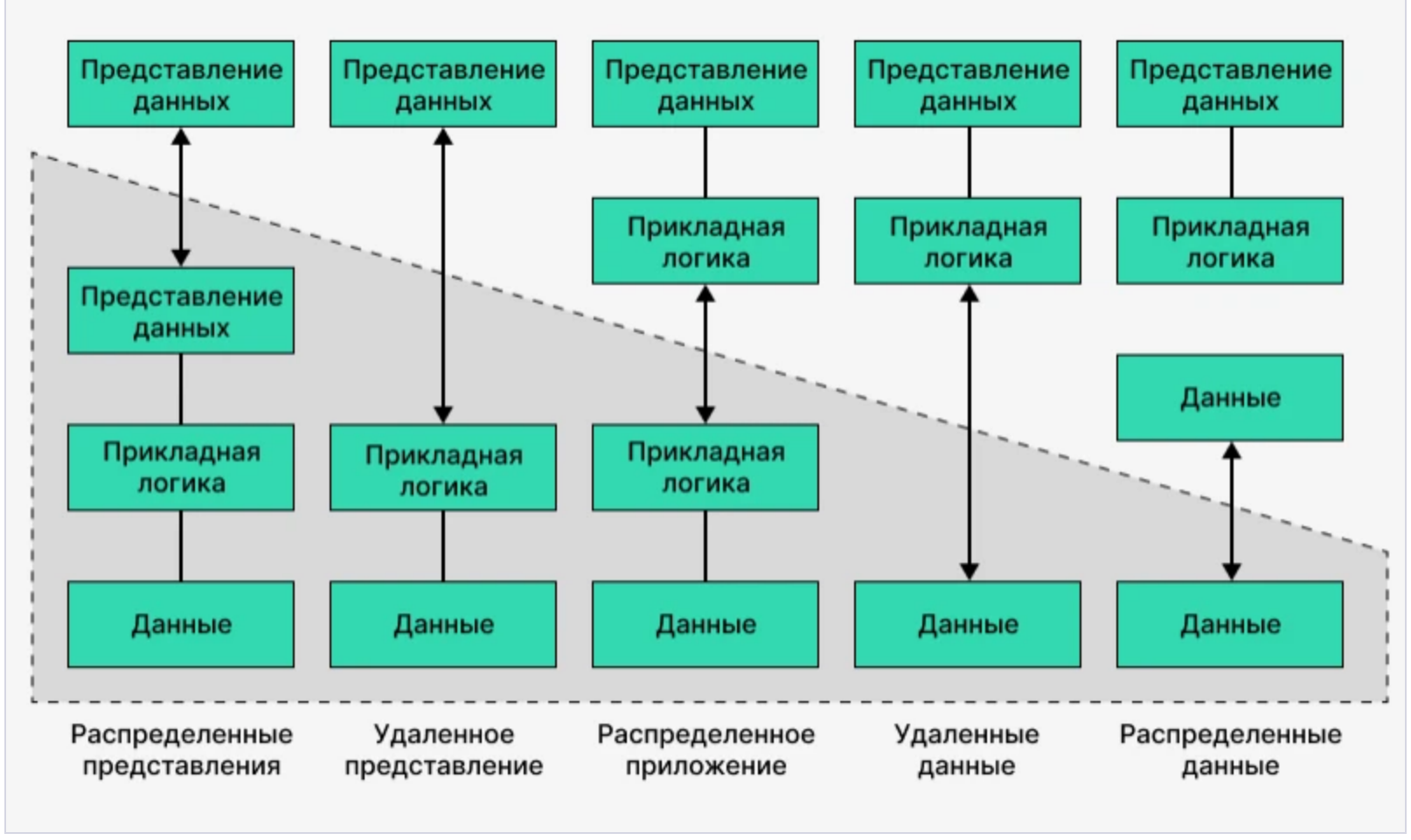
Интересно! Когда Цукерберг начинал писать Facebook, то он работал просто на PHP+MySQL. Когда пользователей стало миллионы, то написали специальный транслятор, который перевел весь PHP код в C++ и скомпилировали его в нативный машинный код.

Также MySQL не способен хранить данные миллиардов пользователей, поэтому Facebook разработал концепцию NoSQL-баз данных и написал мощную серверную NoSQL-СУБД – Cassandra. Кстати, она полностью написана на Java.

Неоднозначность расположения логики приложения

И хотя трехуровневая архитектура практически однозначно распределяет роли между ее частями, не всегда можно правильно определить, в какое именно место системы нужно добавить новую часть бизнес-логики (новый код).

Пример такой неоднозначности представлен на картинке ниже:



Серым фоном залита серверная часть, белым — клиентская. Хороший пример последнего подхода (крайний правый вариант) — это современные мобильные приложения. На стороне клиента (на телефоне) содержится представление (отображение), логика и данные. И только иногда эти данные синхронизируются с сервером.

Пример крайнего левого варианта — это типичный PHP-сервер, у которого вся логика находится на сервере, и клиенту он отдает уже статический HTML. Где-то между этими двумя крайними вариантами и будет находиться ваш проект.

В начале работы, после того, как ты ознакомишься с проектом, тебе нужно будет советоваться с работающими над ним программистами, насчет мест, где тебе лучше реализовать логику очередного задания.

Не стесняйся так делать. Во-первых, в чужой монастырь со своим уставом не лезут. Во-вторых, всем будет проще (и тебе тоже) находить нужный вам код в том месте, где ты ожидаешь его найти.

‹ Предыдущая лекция

Следующая лекция ›

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ