Data Access Layer

Матвей Попов

План

- Что такое DAL?
- Проблематика DAL
- Как с этими данными можно взаимодействовать?
- Какие инструменты и подходы существуют?
- Livecoding

Что такое Data Access Layer?

DAL - Data access layer

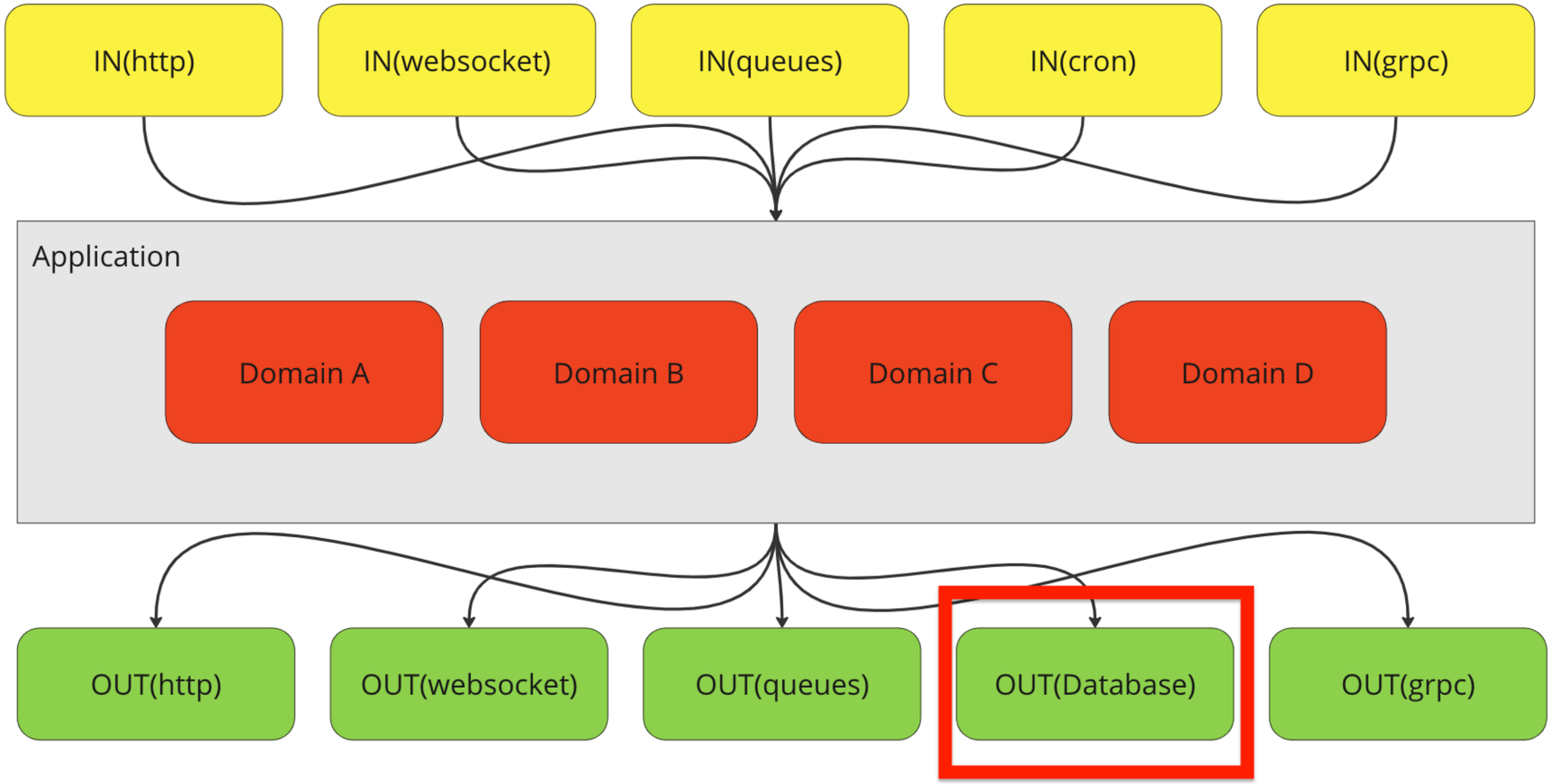
Это слой компьютерной программы, который предоставляет упрощенный доступ к данным, хранимым в постоянном хранилище какого-либо типа.

Другими словами, это:

- Более высокий уровень абстракции
- Сокрытие сложности реализации хранилища

Разберемся глубже с проблематикой

Про что именно мы говорим?

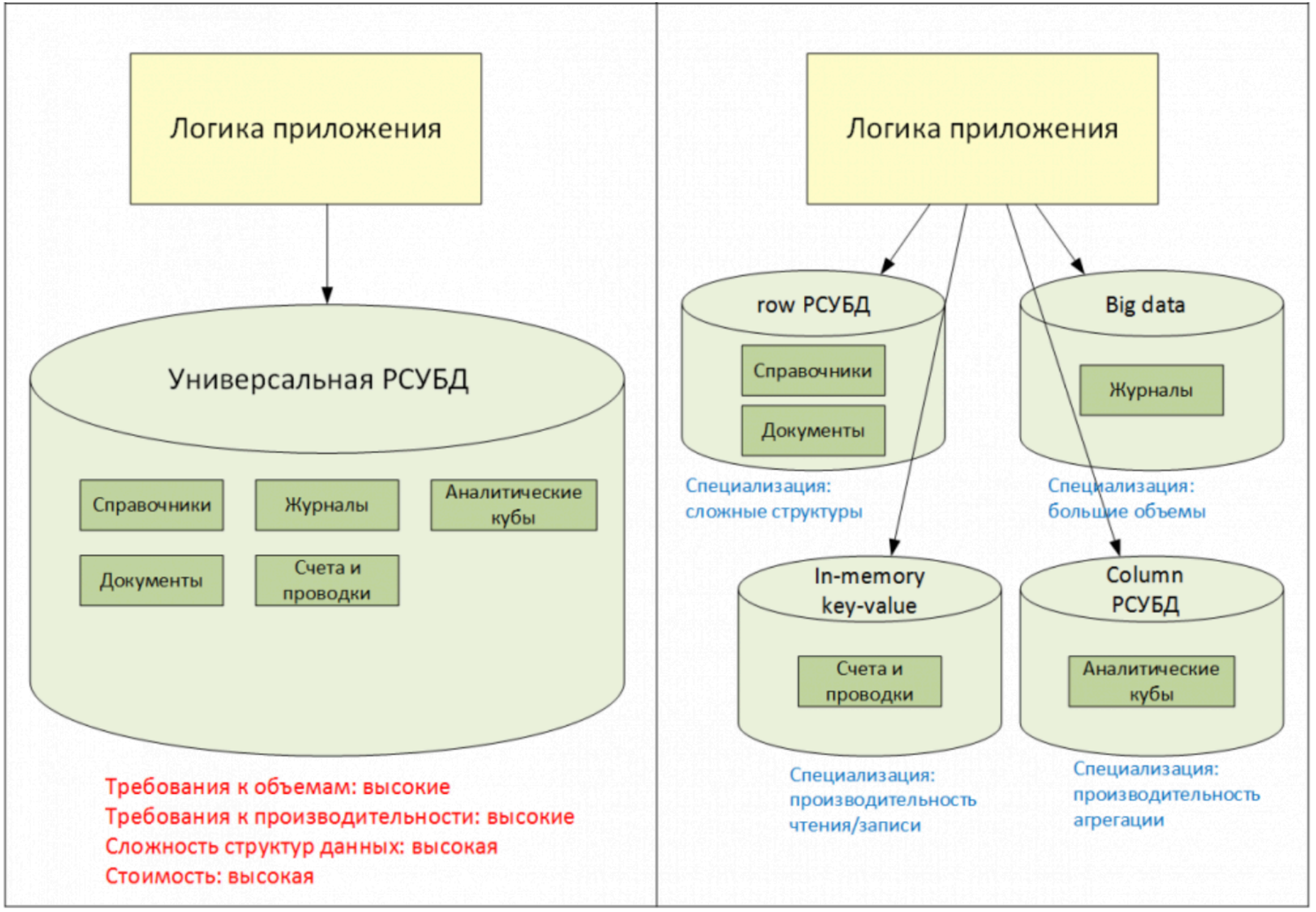

IN(http)
IN(websocket)
IN(queues)
IN(cron)
IN(grpc)
Application
Domain A
Domain B
Domain C
Domain D
OUT(http)
OUT(websocket)
OUT(queues)
OUT(Database)
OUT(grpc)

Для чего нам нужны данные?

Для реализации бизнес процессов

А где эти данные могут быть?

- В одной специализированной БД
- В разных специализированных БД
- В разных БД с разной специализацией


Логика приложения
Универсальная РСУБД
Справочники
Журналы
Аналитические кубы
Документы
Счета и проводки
Требования к объемам: высокие
Требования к производительности: высокие
Сложность структур данных: высокая
Стоимость: высокая
Логика приложения
row РСУБД
Справочники
Документы
Специализация: сложные структуры
Big data
Журналы
Специализация: большие объемы
In-memory key-value
Счета и проводки
Специализация: производительность чтения/записи
Column РСУБД
Аналитические кубы
Специализация: производительность агрегации

Как с данными можно взаимодействовать?

- Писать свой некоторый клиент к каждой базе данных
- Использовать как можно больше хранимых процедур
- Использовать специализированные SDK баз
- Использовать ORM
- Использовать Query DSL

Как с данными нужно взаимодействовать?

Необходимо максимально
абстрагироваться от конкретной БД

Почему?

- Ценовые или иные политики поставщиков хранилищ данных меняются
- С ростом сервиса существующие решения могут переставать удовлетворять требованиям
- Технологии хранения данных развиваются и предоставляют новые средства
- Появляются Open Source проекты, которые предоставляют бесплатные альтернативы

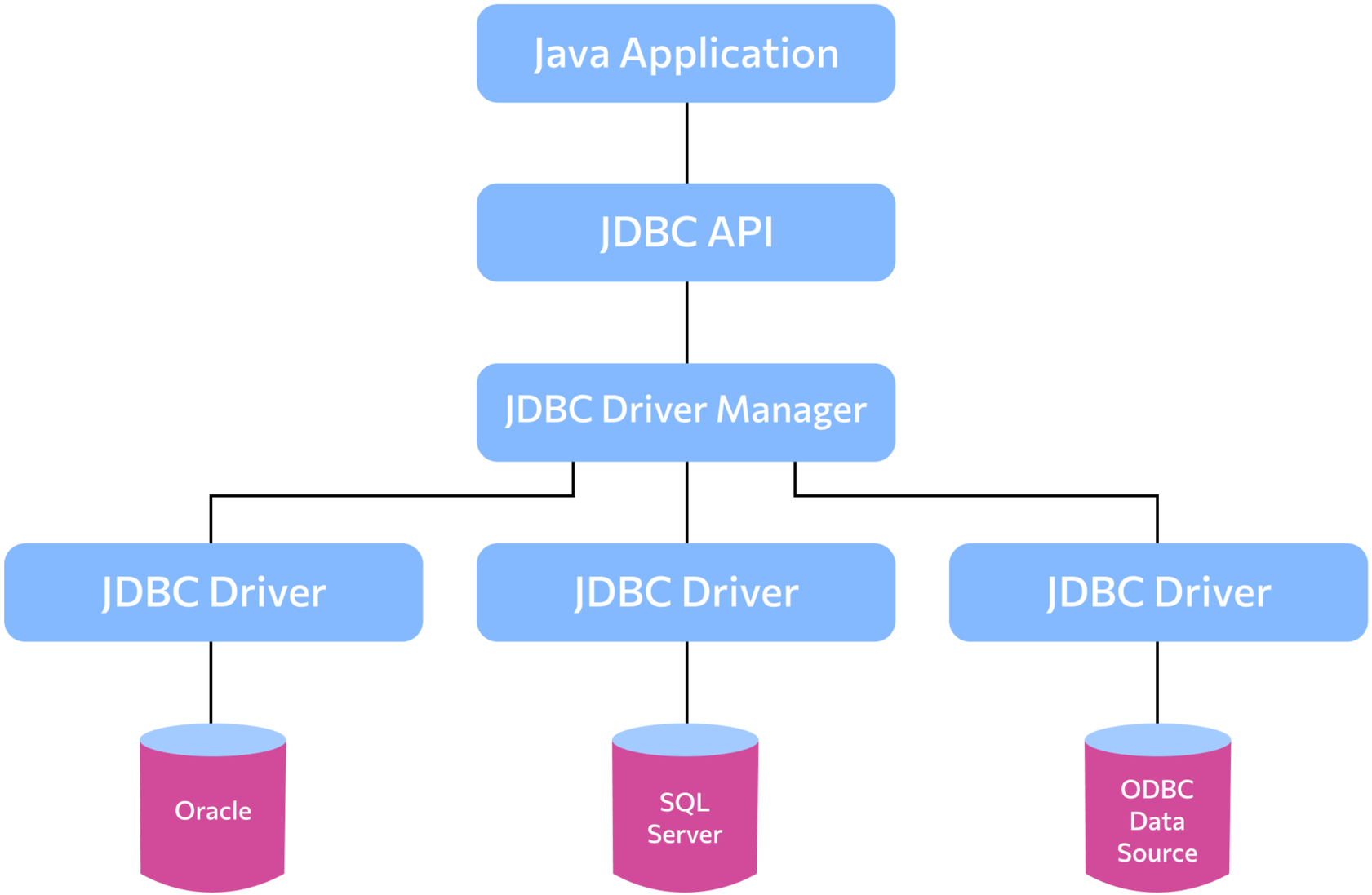
Какие инструменты существуют?

Будем идти от “старого” к “новому”

JDBC - Java Database Connectivity API

- Используют абсолютно все библиотеки
- Java реализация стандартного подхода к работе с базами данных
- Доступна в JDK
- Использует драйвера
- Существует с 1997 года

JDBC Architecture



Преимущества и недостатки

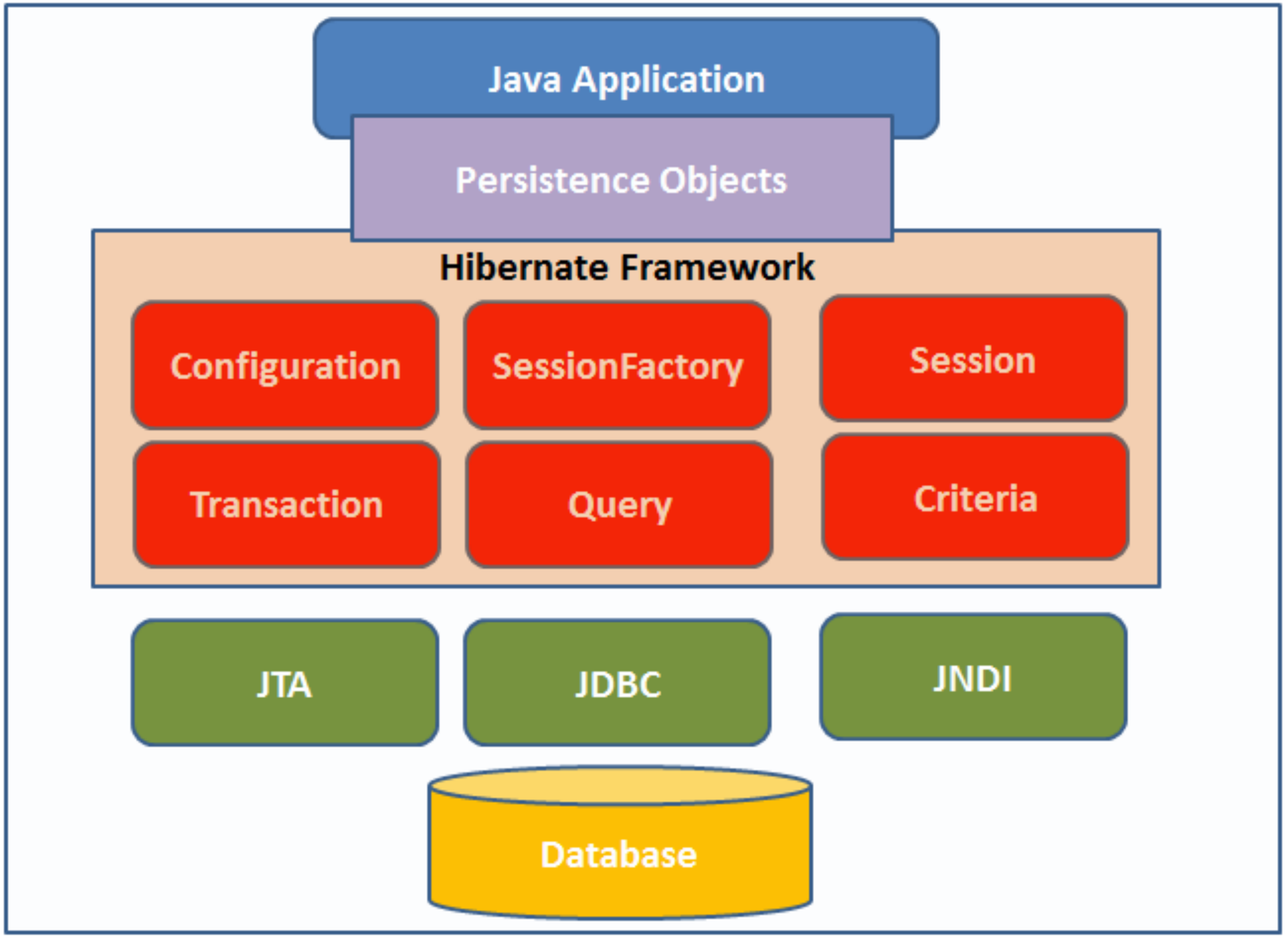
- Гибкий и проверенный временем подход
- Не зависим от реляционной базы данных
- Однако зависим от диалекта SQL
- Не очень удобно соотносить объекты и схемы в таблицах
- Необходимо писать избыточно много кода

Необходимо сильнее
унифицировать подход

Hibernate и JPA

- Hibernate - реализация ORM модели
- Унифицированный подход
- Использует аннотации для настройки
- Очень много функционала

Hibernate Architecture



```kotlin
package com.ferum_bot.quotesapi.models.entity

import ...

Ferum-bot
@Entity
data class QuoteEntity(

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    var id: Long? = null,

    @Column(name = "text", length = 5000)
    var text: String,

    @ManyToOne
    @JoinColumn(name = "author_entity_id", referencedColumnName = "id")
    var author: AuthorEntity? = null,

    @ManyToOne
    @JoinColumn(name = "tag_entity_id", referencedColumnName = "id")
    var tag: TagEntity? = null,

    @JsonFormat(pattern = "dd.MM.yyyy HH:mm:ss")
    var createdDate: LocalDateTime? = null,

    @JsonFormat(pattern = "dd.MM.yyyy HH:mm:ss")
    var updatedDate: LocalDateTime? = null,
) {

    Ferum-bot
    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other == null || Hibernate.getClass( proxy: this) != Hibernate.getClass(other)) {
            return false
        }
        other as QuoteEntity

        return id != null && id == other.id
    }

    Ferum-bot
    override fun hashCode(): Int = 916666166

```

А что же такое JPA?

JPA - Java Persistance API

- Hibernate был настолько крут, что его решили стандартизировать
- JPA - это спецификация
- Hibernate - имплементация JPA

Плюсы и минусы

- JPA очень продуман и очень мощный
- Предоставляет гигантский API
- Сам генерирует SQL запросы
- Очень просто организовывать DataLayer в приложении
- Очень тяжело контролировать генерируемые запросы
- Объектная модель не всегда соответствует реляционной

Может есть какой-то компромиссный подход?

QueryBuilders

- Библиотека все еще валидирует запросы
- Пользователь сам контролирует получаемые SQL
- Все еще унифицировано
- Нужно писать заметно больше кода

- Хорошо абстрагируемся от конкретной технологии

JOOQ

```kotlin
override fun updateProfileEntityAccess(access: ProfileEntityAccess): ProfileEntityAccess {
    val authorEntityAccessValue = accessTransformer.transformToDatabaseValue(access.data.access.accessForAuthor)
    val responsibleEntityAccessValue = accessTransformer.transformToDatabaseValue(access.data.access.accessForResponsible)
    val followerEntityAccessValue = accessTransformer.transformToDatabaseValue(access.data.access.accessForFollower)
    val noRoleEntityAccessValue = accessTransformer.transformToDatabaseValue(access.data.access.accessForNoRole)

    jooqConfiguration.dsl().transaction { cfg ->
        cfg.dsl() DSLContext
            .insertInto(
                PROFILE_ENTITY_ACCESS,
                PROFILE_ENTITY_ACCESS.ID,
                PROFILE_ENTITY_ACCESS.ENTITY_META_ID,
                PROFILE_ENTITY_ACCESS.SECURITY_PROFILE_ID,
                PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_AUTHOR,
                PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_RESPONSIBLE,
                PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_FOLLOWER,
                PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_NO_ROLE
            ) InsertValuesStep7<ProfileEntityAccessRecord!, UUID?, UUID?, UUID?, Byte?, Byte?, Byte?, Byte?>
            .values(
                access.id,
                access.data.entityMetaId,
                access.data.securityProfileId,
                authorEntityAccessValue,
                responsibleEntityAccessValue,
                followerEntityAccessValue,
                noRoleEntityAccessValue
            )
            .onDuplicateKeyUpdate() InsertOnDuplicateSetStep<ProfileEntityAccessRecord!>
            .set(PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_AUTHOR, authorEntityAccessValue) InsertOnDuplicateSetMoreStep<ProfileEntityAccessRecord!>
            .set(PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_RESPONSIBLE, responsibleEntityAccessValue)
            .set(PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_FOLLOWER, followerEntityAccessValue)
            .set(PROFILE_ENTITY_ACCESS.ACCESS_VALUE_FOR_NO_ROLE, noRoleEntityAccessValue)
            .execute()
    }

    return access
}
```

JPA

```
Ferum-bot
67  override fun createNewQuote(quote: CreateQuoteModel): QuoteEntity {
68      val tagEntity = tagsDataSource.findById(quote.tagId).get()
69      val authorEntity = authorsDataSource.findById(quote.authorId).get()
70      val quoteEntity = QuoteEntity(
71          text = quote.text,
72          author = authorEntity,
73          tag = tagEntity,
74      )
75      val resultEntity = quotesDataSource.saveAndFlush(quoteEntity)
76
77      updateTag(tagEntity, resultEntity)
78      updateAuthor(authorEntity, resultEntity)
79
80      return resultEntity
81  }
82
```

А что будем использовать мы?

EXPOSED

- Написан на Kotlin для Kotlin приложений
- Представляет облегченную ORM
- Можно использовать в виде DSL в SQL
- Поддержка Coroutine из коробки
- Использует JDBC под капотом
- Многомодульная архитектура

Примеры

Описание таблицы

```
8

    Ferum-bot
9   object PostsTable: LongIdTable( name: "posts") {
10
11      val name = text( name: "name", eagerLoading = true)
12
13      val text = text( name: "text", eagerLoading = true)
14
15      val isVisible = bool( name: "is_visible").default( defaultValue: true)
16
17      val isDeleted = bool( name: "is_deleted").default( defaultValue: false)
18
19      val isEdited = bool( name: "is_edited").default( defaultValue: false)
20
21      val author = reference(
22          name = "author_user_id", foreign = UserProfilesTable,
23          onDelete = NO_ACTION, onUpdate = NO_ACTION
24      )
25
26  //    val settings = reference(
27  //        name = "post_settings_id", foreign = PostsSettingsTable,
28  //        onDelete = NO_ACTION, onUpdate = NO_ACTION
29  //    )
30
31      val createdDate = datetime( name: "created_date").clientDefault { localDateTimeNow() }
32
33      val updatedDate = datetime( name: "updated_date").clientDefault { localDateTimeNow() }
34  }
```

Описание сущности

```
14
    Ferum-bot
15  class Post(id: EntityID<Long>): LongEntity(id) {
        Ferum-bot
16      companion object: CallbackLongEntityClass<Post>(PostsTable)
17
18      var name by PostsTable.name
19
20      var text by PostsTable.text
21
22      var author by PostsTable.author
23
24      val comments by PostComment referrersOn PostCommentsTable.post
25
26      val reactions by PostReaction referrersOn PostReactionsTable.post
27
28      var tags by PostTag via PostToTagTable
29
30      var isVisible by PostsTable.isVisible
31
32      var isDeleted by PostsTable.isDeleted
33
34      var isEdited by PostsTable.isEdited
35
36      var createdDate by PostsTable.createdDate
37
38      var updatedDate by PostsTable.updatedDate
39
```

Пример запроса

```
Ferum-bot
override fun getPostsWithAuthorIds(
    authorIds: Collection<EntityID<Long>>, pageNumber: Int, pageSize: Int
): List<Post> {
    val offset = (pageNumber - 1).toLong() * pageSize
    val query = PostsTable
        .select {
            PostsTable.author inList authorIds
        }
        .orderBy(PostsTable.createdDate, SortOrder.DESC_NULLS_LAST)
        .limit(pageSize, offset)
    val result = Post.wrapRows(query)
    return result.toList()
}

Ferum-bot
```

Плюсы и Минусы

- Лаконичен и легковесен
- Использует весь функционал Kotlin
- Можно использовать как ORM так и QueryBuilder
- Поддерживает очень мало баз данных
- Очень молодая библиотека
- Небольшое сообщество
- Очень активно развивается

А теперь подключим Exposed в наш проект

Livecoding: Добавляем хранение данных

Выводы

Kotlin - это все еще очень круто!

Домашнее задание

Нужно добавить хранение данных в ваше приложение

Материалы

Будут в README

Спасибо за внимание!