

# 极客大学 Java 进阶训练营

## 第 10 课

### Java 相关框架 (2)



KimmKing

Apache Dubbo/ShardingSphere PMC

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

## 目录

1. 从 Spring 到 Spring Boot
2. Spring Boot 核心原理\*
3. Spring Boot Starter 详解\*
4. JDBC 与数据库连接池\*
5. ORM-Hibernate/MyBatis\*
6. Spring 集成 ORM/JPA\*
7. Spring Boot 集成 ORM/JPA
8. 第 10 课总结回顾与作业实践

# 1. 从 Spring 到 Spring Boot

# Spring 变得越来越复杂

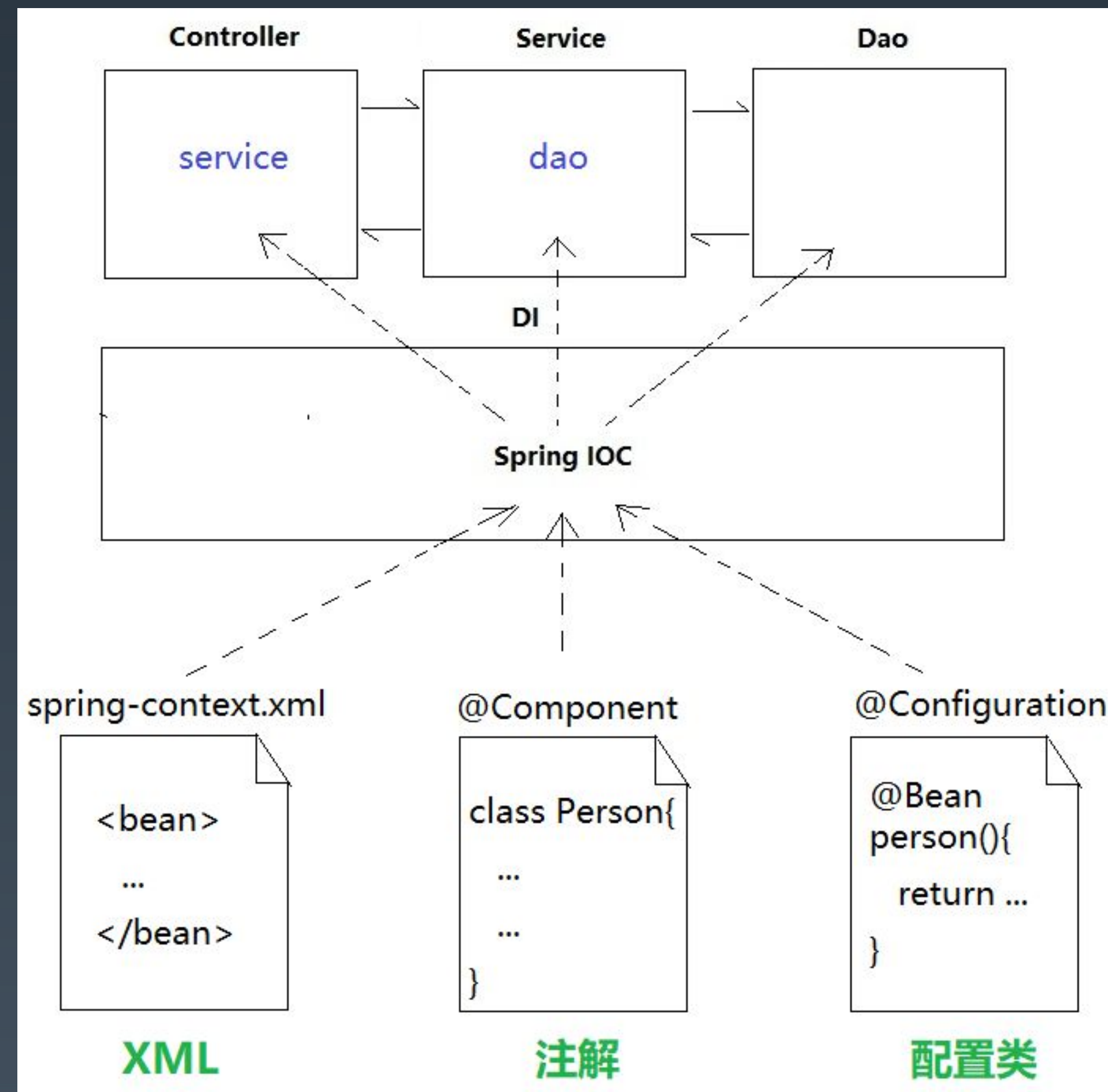
配置的发展方向：

XML--全局

注解--类

配置类--方法

Spring 4 以上的新特性，走向 Spring Boot



功能，使用方式太复杂，怎么办？

# Spring Boot 的出发点

Spring 臃肿以后的必然选择。

一切都是为了简化。

- 让开发变简单：
- 让配置变简单：
- 让运行变简单：

限定性框架和非限定性框架？

怎么变简单？关键词：整合

就像是 SSH、SSM，国产的 SpringSide

基于什么变简单：约定大于配置。

# Spring Boot 如何做到简化

为什么能做到简化：

- 1、Spring 本身技术的成熟与完善，各方面第三方组件的成熟集成
- 2、Spring 团队在去 web 容器化等方面的努力
- 3、基于 MAVEN 与 POM 的 Java 生态体系，整合 POM 模板成为可能
- 4、避免大量 maven 导入和各种版本冲突

Spring Boot 是 Spring 的一套快速配置脚手架，关注于自动配置，配置驱动。

什么是脚手架？

# 什么是 Spring Boot



Spring Boot 使创建独立运行、生产级别的 Spring 应用变得容易，你可以直接运行它。我们对 Spring 平台和第三方库采用限定性视角，以此让大家能在最小的成本下上手。大部分 Spring Boot 应用仅仅需要最少量的配置。


## 功能特性

1. 创建独立运行的 Spring 应用
2. 直接嵌入 Tomcat 或 Jetty, Undertow, 无需部署 WAR 包
3. 提供限定性的 starter 依赖简化配置（就是脚手架）
4. 在必要时自动化配置 Spring 和其他三方依赖库
5. 提供生产 production-ready 特性，例如指标度量，健康检查，外部配置等
6. 完全零代码生产和不需要 XML 配置



# 快速构建基础 maven 项目





**Project**

- ☒ Maven Project
- ☐ Gradle Project

**Language**

- ☒ Java ☐ Kotlin
- ☐ Groovy

**Spring Boot**

- ☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M4) ☐ 2.3.5 (SNAPSHOT)
- ☒ 2.3.4 ☐ 2.2.11 (SNAPSHOT) ☐ 2.2.10 ☐ 2.1.18 (SNAPSHOT)
- ☐ 2.1.17

**Project Metadata**

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name



com.example.demo

**Dependencies**

ADD DEPENDENCIES... CTRL + B

No dependency selected

<https://start.spring.io/>



GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

## 2.Spring Boot 核心原理\*

# Spring Boot 两大核心原理

1、自动化配置：简化配置核心  
基于 Configuration, EnableXX, Condition

2、spring-boot-starter：脚手架核心  
整合各种第三方类库，协同工具

# Spring Boot 两大核心原理

application.yaml

Configuration

Bean

前缀

一组配置

Starter 组件

# 为什么要约定大于配置

为什么要约定大于配置？

举例来说，JVM 有1000多个参数，但是我们不需要一个参数，就能 java Hello。

优势在于，开箱即用：

- 一、Maven 的目录结构：默认有 resources 文件夹存放配置文件。默认打包方式为 jar。
- 二、默认的配置文件的 application.properties 或 application.yml 文件
- 三、默认通过 spring.profiles.active 属性来决定运行环境时的配置文件。
- 四、EnableAutoConfiguration 默认对于依赖的 starter 进行自动装载。
- 五、spring-boot-start-web 中默认包含 spring-mvc 相关依赖以及内置的 web 容器，使得构建一个 web 应用更加简单。

什么是脚手架？

# 自动化配置原理

## 自动化配置

```
@EnableAutoConfiguration
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

```
@Configuration
public class WebConfiguration {
    ...
}
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@Import(WebConfiguration.class)
public class WebAutoConfiguration {
}
```

```
# src/main/resources/META-INF/spring.factories
# 自动装配
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.xxxx.WebAutoConfiguration
```

什么是脚手架？

# Spring Boot 自动配置注解

- @SpringBootApplication

SpringBoot 应用标注在某个类上说明这个类是 SpringBoot 的主配置类，SpringBoot 就会运行这个类的 main 方法来启动 SpringBoot 项目。

- @SpringBootConfiguration

- @EnableAutoConfiguration

- @AutoConfigurationPackage

- @Import({AutoConfigurationImportSelector.class})

加载所有 META-INF/spring.factories 中存在的配置类（类似 SpringMVC 中加载所有 converter）

核心启动入口

# 条件化自动配置

@ConditionalOnBean

@ConditionalOnClass

@ConditionalOnMissingBean

@ConditionalOnProperty

@ConditionalOnResource

@ConditionalOnSingleCandidate

@ConditionalOnWebApplication

运行时灵活组装，避免冲突



## 3.Spring Boot Starter 详解\*

# 以一个实际项目讲解 Starter

```
shardingsphere-jdbc-core-spring-boot-starter
├── src
│   ├── main
│   │   ├── java
│   │   │   └── org.apache.shardingsphere.spring.boot
│   │   │       ├── prop
│   │   │       └── SpringBootConfiguration
│   └── resources
│       └── META-INF
│           ├── additional-spring-configuration-metadata.json
│           ├── spring.factories
│           └── spring.provides
```

1、spring.provides

2、spring.factories

3、additional--metadata

4、自定义 Configuration 类

# 以一个实际项目讲解 Starter

```
@Configuration
@ComponentScan("org.apache.shardingsphere.spring.boot.converter")
@EnableConfigurationProperties(SpringBootPropertiesConfiguration.class)
@ConditionalOnProperty(prefix = "spring.shardingsphere", name = "enabled", havingValue = "true", matchIfMissing = true)
@AutoConfigureBefore(DataSourceAutoConfiguration.class)
@RequiredArgsConstructor
public class SpringBootConfiguration implements EnvironmentAware {
    ...

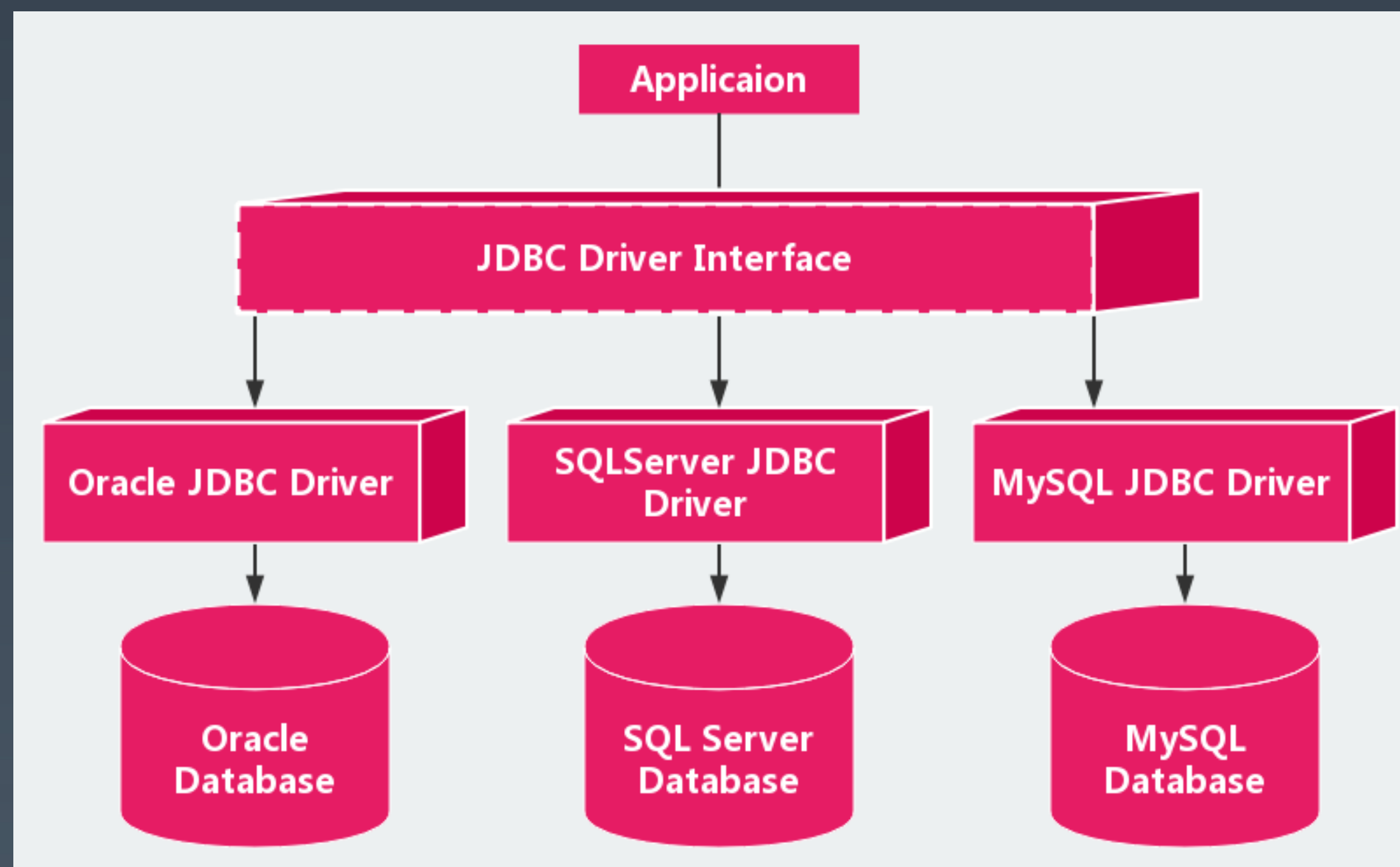
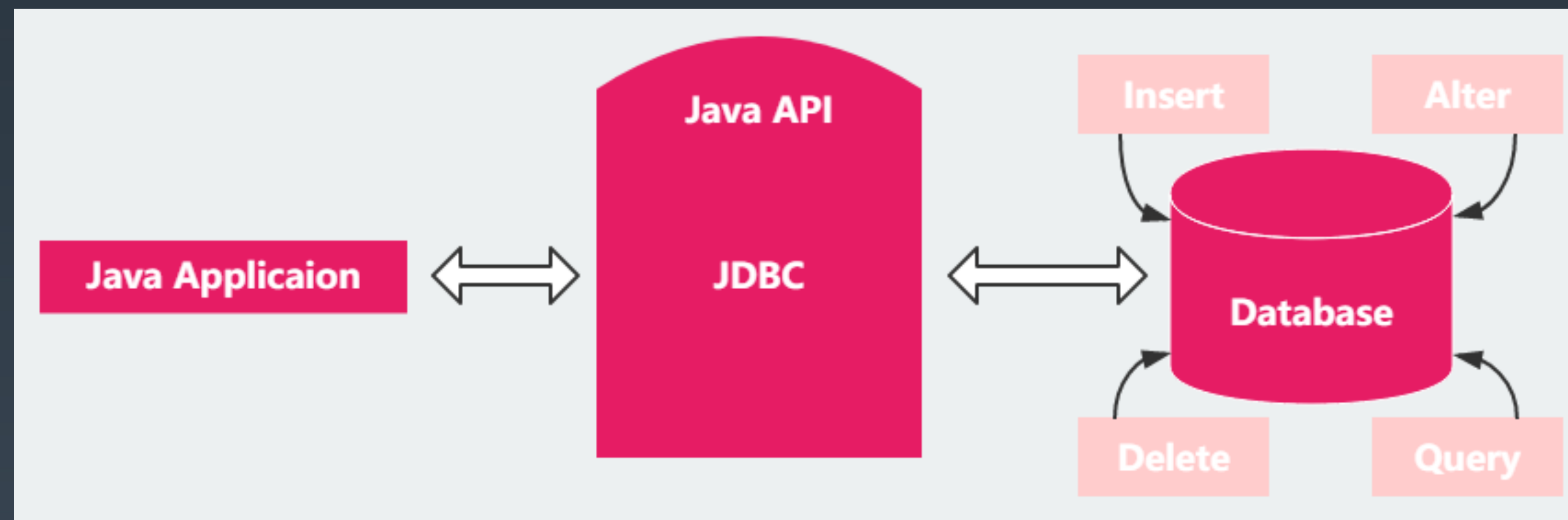
    private final SpringBootPropertiesConfiguration props;
    ...

    private final Map<String, DataSource> dataSourceMap = new LinkedHashMap<>();
    ...

    /**
     * Get ShardingSphere data source bean.
     *
     * @param rules rules configuration
     * @return data source bean
     * @throws SQLException SQL exception
     */
    @Bean
    @Autowired(required = false)
    public DataSource shardingSphereDataSource(final ObjectProvider<List<RuleConfiguration>> rules) throws SQLException {
        Collection<RuleConfiguration> ruleConfigurations = Optional.ofNullable(rules.getIfAvailable()).orElse(Collections.emptyList());
        return ShardingSphereDataSourceFactory.createDataSource(dataSourceMap, ruleConfigurations, props.getProps());
    }
}
```

## 4.JDBC 与数据库连接池

# JDBC



JDBC 定义了数据库交互接口:

DriverManager

Connection

Statement

ResultSet

后来又加了DataSource--Pool

# JDBC 是 Java 里操作数据库的核心

Java 操作数据库的各种类库，都可以看做是在 JDBC 上做的增强实现

为什么可以这么做？

加上 XA 事务--XAConnection

从连接池获取--PooledConnection

MySQL 驱动 JDBC 接口--Connection

# 数据库连接池

C3P0

DBCP--Apache CommonPool

Druid

Hikari

连接池需要哪些功能，如何实现？

## 5.ORM-Hibernate/MyBatis



# Hibernate

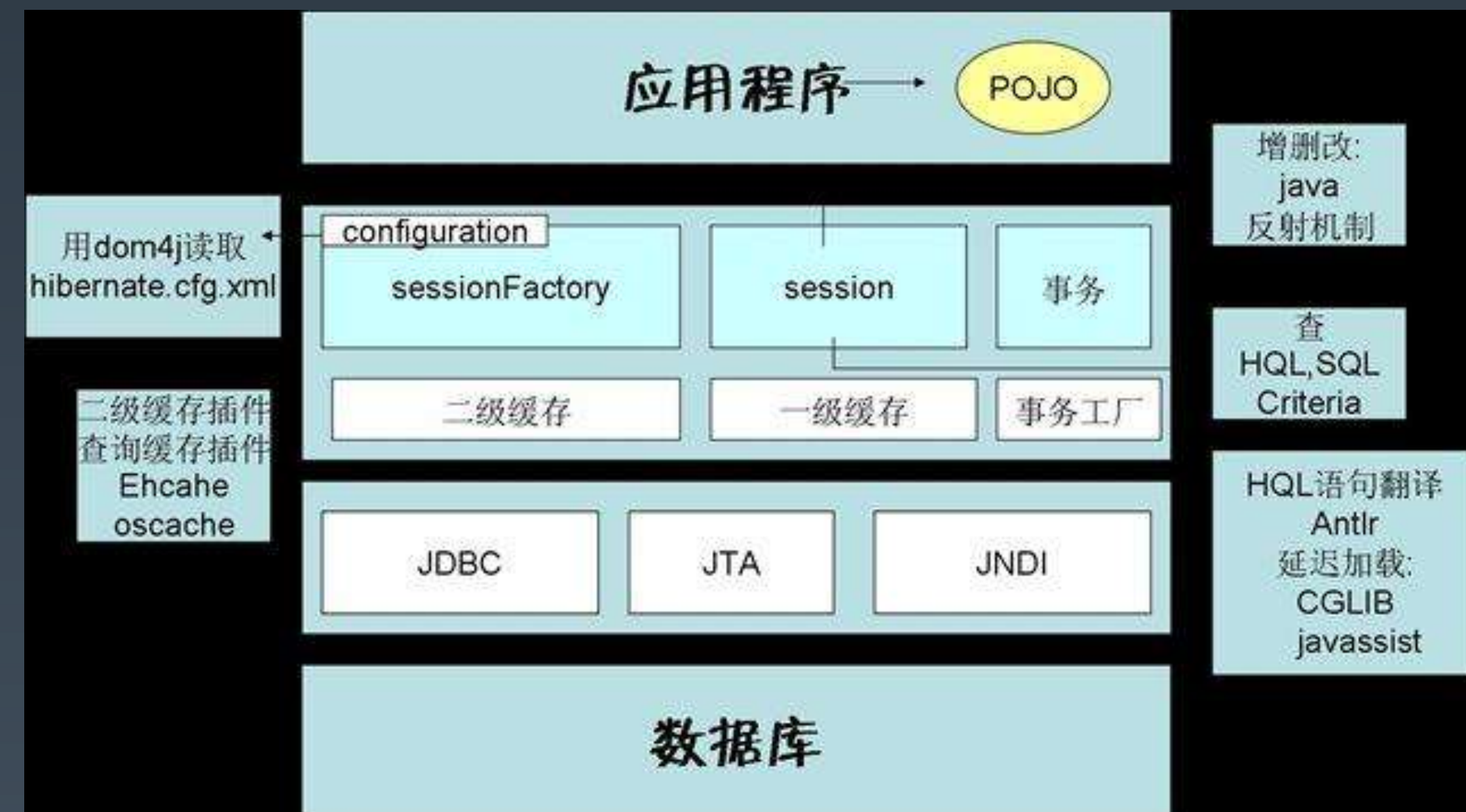
ORM (Object-Relational Mapping) 表示对象关系映射。

Hibernate 是一个开源的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，它将 POJO 与数据库表建立映射关系，是一个全自动的 orm 框架，hibernate 可以自动生成 SQL 语句，自动执行，使得 Java 程序员可以使用面向对象的思维来操纵数据库。

Hibernate 里需要定义实体类和 hbm 映射关系文件（IDE 一般有工具生成）。

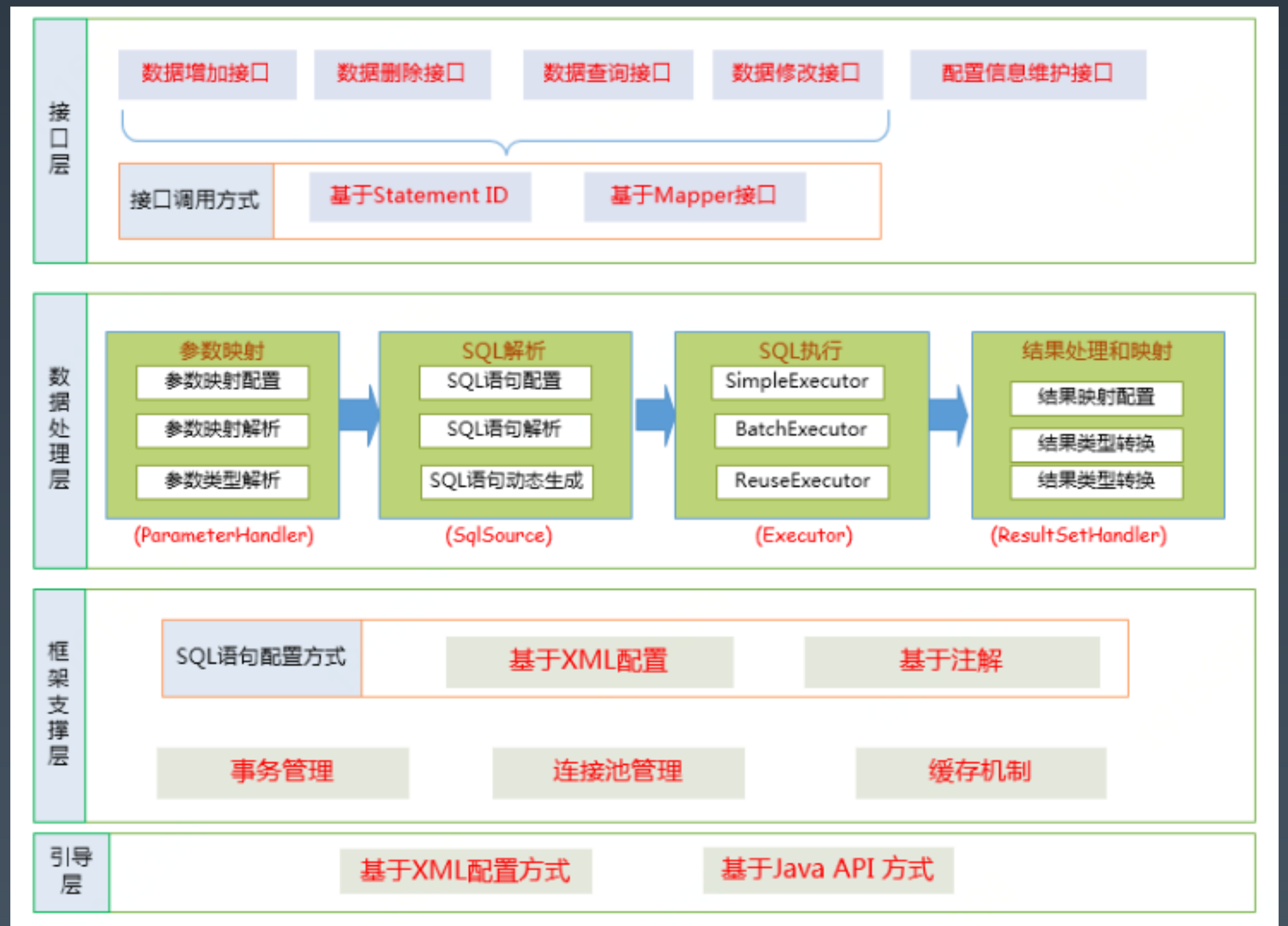
Hibernate 里可以使用 HQL、Criteria、Native SQL 三种方式操作数据库。

也可以作为 JPA 适配实现，使用 JPA 接口操作。



# MyBatis

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java 对象)映射成数据库中的记录。



# MyBatis-半自动化 ORM

- 1、需要使用映射文件 mapper.xml 定义 map规则和 SQL
- 2、需要定义 mapper/DAO, 基于 xml 规则, 操作数据库

可以使用工具生成基础的 mapper.xml 和 mapper/DAO

一个经验就是, 继承生成的 mapper, 而不是覆盖掉

也可以直接在 mapper 上用注解方式配置 SQL

# MyBatis 与 Hibernate 比较

MyBatis 与 Hibernate 的区别与联系？

Mybatis 优点：原生 SQL（XML 语法），直观，对 DBA 友好

Hibernate 优点：简单场景不用写 SQL（HQL、Criteria、SQL）

Mybatis 缺点：繁琐，可以用 MyBatis-generator、MyBatis-Plus 之类的插件

Hibernate 缺点：对 DBA 不友好

考虑为什么大公司都用 MyBatis？

## 6.Spring 集成 ORM 与 JPA\*

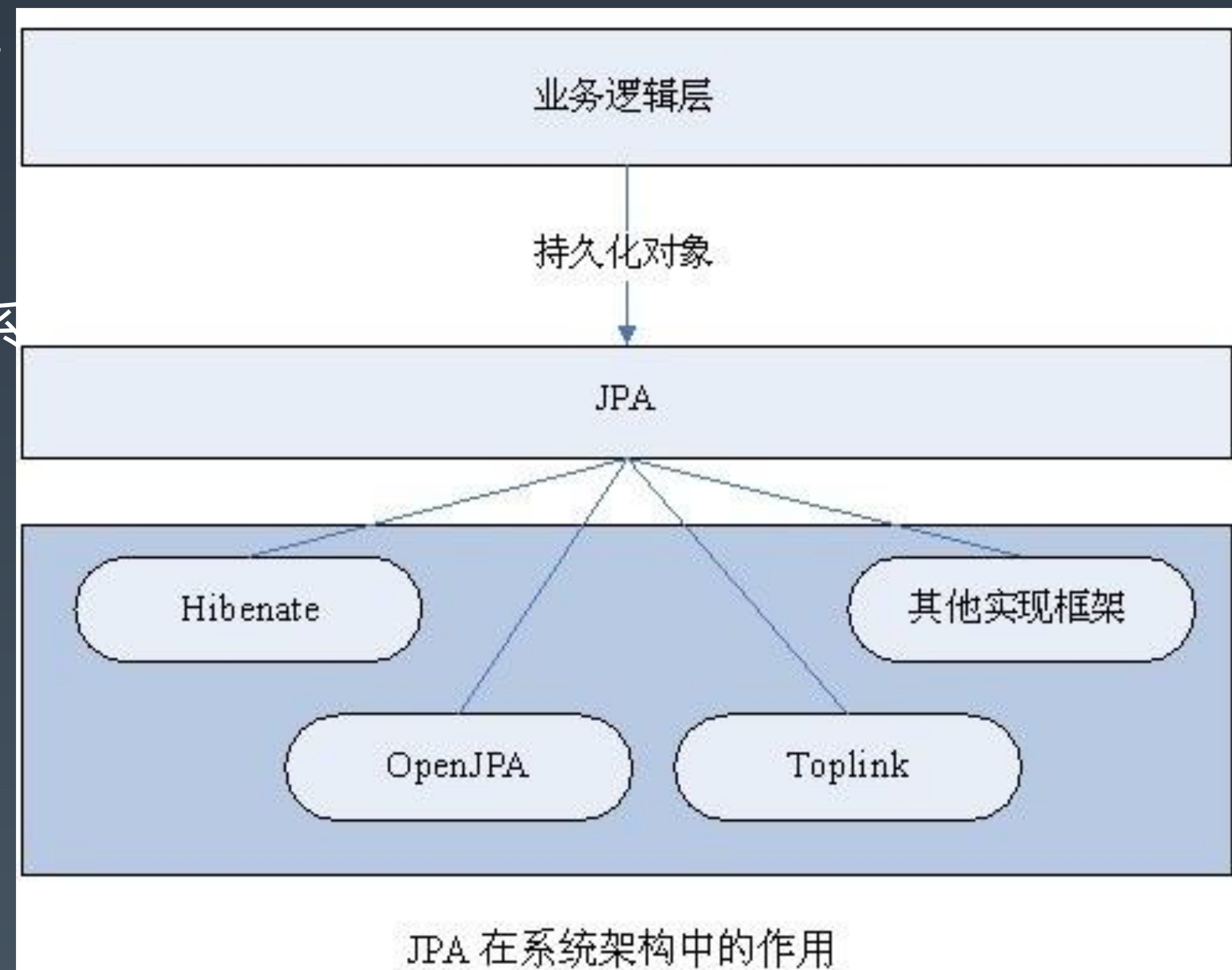


# JPA

JPA 的全称是 Java Persistence API,  
即 Java 持久化 API, 是一套基于 ORM 的规范,  
内部是由一系列的接口和抽象类构成。

JPA 通过 JDK 5.0 注解描述对象-关系表映射关系,  
并将运行期的实体对象持久化到数据库中。

核心 EntityManager



# Spring JDBC 与 ORM

JDBC

DataSource

Spring JDBC

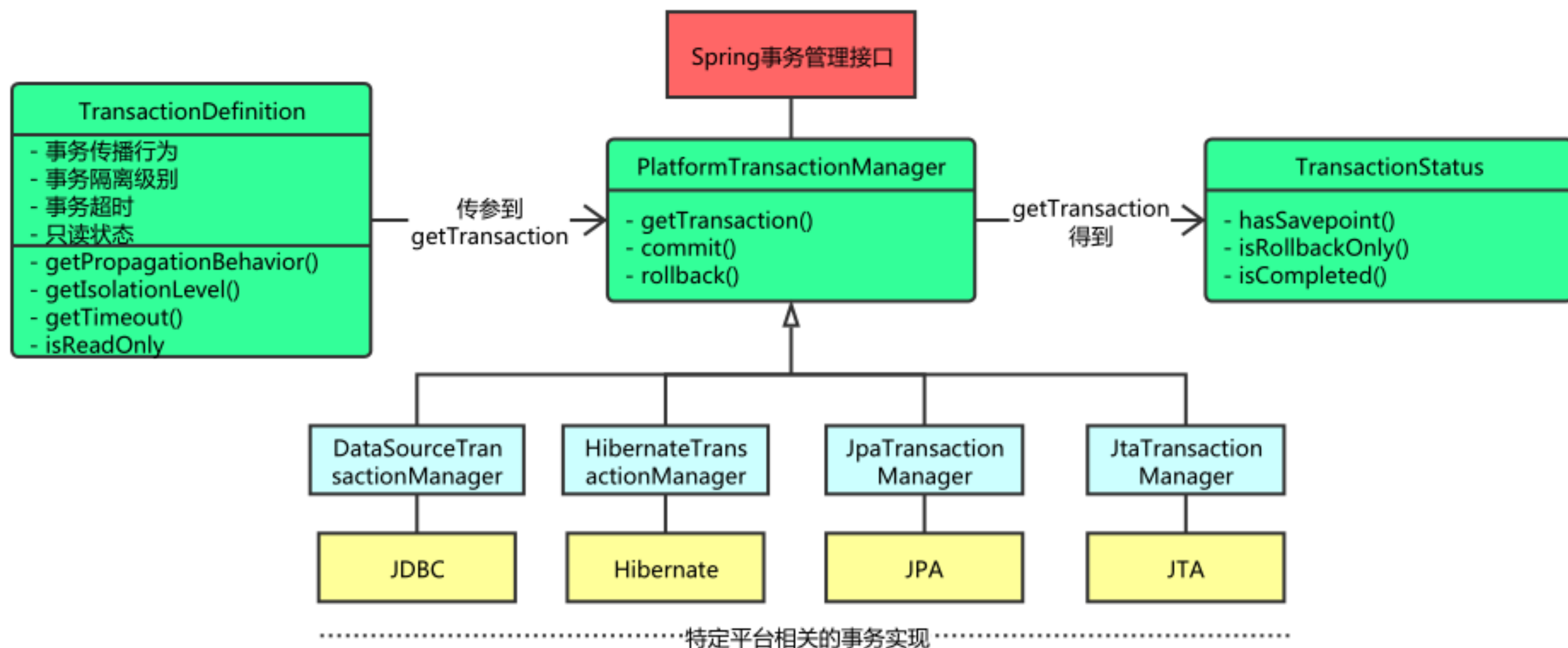
JPA

EntityManager

Spring ORM

Spring Data with NoSQL ?

# Spring 管理事务



JDBC 层，数据库访问层，怎么操作事务？程式化事务管理

Spring 怎么做到无侵入实现事务？声明式事务管理：事务管理器+AOP



# Spring 管理事务

Spring 声明式事务配置参考

事务的传播性：

`@Transactional(propagation=Propagation.REQUIRED)`

事务的隔离级别：

`@Transactional(isolation = Isolation.READ_UNCOMMITTED)`

读取未提交数据(会出现脏读, 不可重复读) 基本不使用

只读：

`@Transactional(readOnly=true)`

该属性用于设置当前事务是否为只读事务，设置为 true 表示只读，false 则表示可读写，默认值为 false。

事务的超时性：

`@Transactional(timeout=30)`

回滚：

指定单一异常类：`@Transactional(rollbackFor=RuntimeException.class)`

指定多个异常类：`@Transactional(rollbackFor={RuntimeException.class, Exception.class})`

# Spring 集成 MyBatis

演示操作 Spring 与 Mybatis :

Mybatis 用法展示

User 表

XML/Mapper

# Spring 集成 Hibernate/JPA

演示操作 Spring 与 Hibernate/JPA :

Hibernate/JPA 用法展示

User 实体类

注解

## 7.Spring Boot 集成 ORM 与 JPA\*

# Spring Boot 集成 JPA/Hibernate

演示操作 Spring Boot 与 MyBatis :

准备环境与配置依赖

Configuration 与配置文件

Pojo、Mapper 与服务类

启动类

访问测试

# Spring Boot 集成 MyBatis

演示操作 Spring Boot 与 MyBatis :

准备环境与配置依赖

Configuration 与配置文件

Pojo、Mapper 与服务类

启动类

访问测试

# Spring/Spring Boot 使用 ORM 的经验

- 1、本地事务（事务的设计与坑）
- 2、多数据源（配置、静态制定、动态切换）
- 3、线程池配置（大小、重连）
- 4、ORM 内的复杂 SQL，级联查询
- 5、ORM 辅助工具和插件

## 8.总结回顾与作业实践



## 第 10 节课总结回顾

Spring Boot

Hibernate

MyBatis

Spring+ORM

## 第10节课作业实践

1. (选做) 总结一下, 单例的各种写法, 比较它们的优劣。
2. (选做) maven/spring 的 profile 机制, 都有什么用法?
3. (必做) 给前面课程提供的 Student/Klass/School 实现自动配置和 Starter。
4. (选做) 总结 Hibernate 与 MyBatis 的各方面异同点。
5. (选做) 学习 MyBatis-generator 的用法和原理, 学会自定义 TypeHandler 处理复杂类型。
6. (必做) 研究一下 JDBC 接口和数据库连接池, 掌握它们的设计和用法:
  - 1) 使用 JDBC 原生接口, 实现数据库的增删改查操作。
  - 2) 使用事务, PreparedStatement 方式, 批处理方式, 改进上述操作。
  - 3) 配置 Hikari 连接池, 改进上述操作。提交代码到 Github。

附加题 (可以后面上完数据库的课再考虑做):

1. (挑战) 基于 AOP 和自定义注解, 实现 @MyCache(60) 对于指定方法返回值缓存60秒。
2. (挑战) 自定义实现一个数据库连接池, 并整合 Hibernate/Mybatis/Spring/SpringBoot。
3. (挑战) 基于 MyBatis 实现一个简单的分库分表+读写分离+分布式 ID 生成方案。

THANKS!

