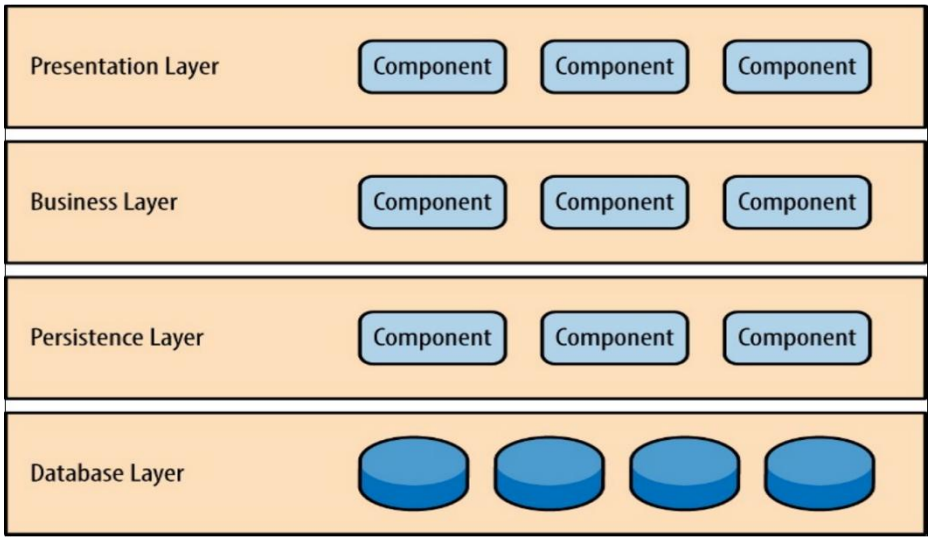
分层架构---四层架构实现



总体使用 SSM 框架进行实现（可以进行注释开发或 xml 配置开发，团队熟悉）

Presentation Layer(表示层)

表示层在前端，页面用 jsp 编写，再用 Tomcat 服务器进行部署

```
<head>
<meta charset="utf-8">
<title>登陆</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="">
<meta name="author" content="">

<!-- Le styles -->
<script type="text/javascript"
    src="${pageContext.request.contextPath}/resources/assets2/js/jquery.min.js"></script>

<!--    <link rel="stylesheet" href="${pageContext.request.contextPath}/resources/assets2/css
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/assets2/css/loader-style.css">
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/assets2/css/bootstrap.css">
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/assets2/css/signin.css">
```

Business Layer(业务逻辑层)

Controller 层对前端页面的数据请求做转发

```
@RestController
public class CommonController{
    @Autowired
    private CommonService commonService;

    private static AipFace client = null;  5 个用法

    @Autowired
    private ConfigService configService;
    获取table表中的column列表(联动接口)
    返回值:
    @RequestMapping("/option/{tableName}/{columnName}")
    @IgnoreAuth
    public R getOption(@PathVariable("tableName") String tableName, @PathVariable("columnNam
        Map<String, Object> params = new HashMap<>();
        params.put("table", tableName);
        params.put("column", columnName);
        if(StringUtils.isNotBlank( cs: level)) {
            params.put("level", level);
        }
```

service 层是主要业务实现，先对业务类定义接口再创建实现

```java
public interface CommonService {  4 个用法  1 个实现
    List<String> getOption(Map<String, Object> params);  1 个用法  1 个实现

    Map<String, Object> getFollowByOption(Map<String, Object> params);  1 个用法  1 个实现

    void sh(Map<String, Object> params);  1 个用法  1 个实现

    int remindCount(Map<String, Object> params);  1 个实现

    Map<String, Object> selectCal(Map<String, Object> params);  1 个用法  1 个实现
```

```java
系统用户
@Service("commonService")
public class CommonServiceImpl implements CommonService {

    @Autowired
    private CommonDao commonDao;

    @Override  1 个用法
    public List<String> getOption(Map<String, Object> params) { return commonDao.getOption(params); }

    @Override  1 个用法
    public Map<String, Object> getFollowByOption(Map<String, Object> params) {
        return commonDao.getFollowByOption(params);
    }

    @Override  1 个用法
    public void sh(Map<String, Object> params) { commonDao.sh(params); }

    @Override
    public int remindCount(Map<String, Object> params) { return commonDao.remindCount(params); }

    @Override  1 个用法
    public Map<String, Object> selectCal(Map<String, Object> params) { return commonDao.selectCal(params); }
```

Persistence Layer（持久层）

持久层的 entiey 层创建对象实体

```java
@TableName("dailianshangcheng")  50 个用法  1 个继承者
public class DailianshangchengEntity<T> implements Serializable {
    private static final long serialVersionUID = 1L;  0 个用法


    @Contract(pure = true)
    public DailianshangchengEntity() {  2 个用法

    }

    public DailianshangchengEntity(T t) {  0 个用法
        try {
            BeanUtils.copyProperties( dest: this, orig: t);
        } catch (IllegalAccessException | InvocationTargetException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    主键id
    @TableId  2 个用法
    private Long id;
```

持久层的 mapper 层使用 MybatisPlus 做 sql 语句的映射，实现对数据库的查找

MybaitPlus（对 mapper 文件进行管理，避免了 jdbc 硬编码问题）

```xml
<mapper namespace="com.dao.CommonDao">
    <select id="getOption" resultType="String" >
        SELECT distinct ${column} FROM ${table}
        where ${column} is not null and ${column} !=''
        <if test = "level != null">
        and level=#{level}
        </if>
        <if test = "parent != null">
        and parent=#{parent}
        </if>
    </select>

    <select id="getFollowByOption" resultType="map" >
        SELECT * FROM ${table} where ${column}=#{columnValue}
    </select>

    <update id="sh">
        UPDATE ${table} set sfsh=#{sfsh} where id=#{id}
    </update>

    <select id="remindCount" resultType="int" >
        SELECT count(1) FROM ${table}
        where 1=1
        <if test = "type == 1 ">
            <if test = " remindstart != null ">
```

dao 层每一个接口继承对应的 mapper 映射

```
 email
 date  2022-05-04 18:26:05
public interface ErcitousuDao extends BaseMapper<ErcitousuEntity> {  3 个用法

    List<ErcitousuVO> selectListVO(@Param("ew") Wrapper<ErcitousuEntity> wrapper);

    ErcitousuVO selectVO(@Param("ew") Wrapper<ErcitousuEntity> wrapper);

    List<ErcitousuView> selectListView(@Param("ew") Wrapper<ErcitousuEntity> wrapper);

    List<ErcitousuView> selectListView(Pagination page,@Param("ew") Wrapper<ErcitousuEntity> wrapp

    ErcitousuView selectView(@Param("ew") Wrapper<ErcitousuEntity> wrapper);


}
```

druid 连接池管理数据库连接（druid 使用简单，仅需配置对应文件就可运行）

```
<!-- 配置数据源 -->
<bean name="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"
    <property name="url" value="${jdbc_url}"/>
    <property name="username" value="${jdbc_username}"/>
    <property name="password" value="${jdbc_password}"/>

    <!-- 初始化连接大小 -->
    <property name="initialSize" value="0"/>
    <!-- 连接池最大使用连接数量 -->
    <property name="maxActive" value="20"/>
    <!-- 连接池最大空闲 -->
    <property name="maxIdle" value="20"/>
    <!-- 连接池最小空闲 -->
    <property name="minIdle" value="0"/>
    <!-- 获取连接最大等待时间 -->
    <property name="maxWait" value="60000"/>

    <property name="validationQuery" value="${validationQuery}"/>
    <property name="testOnBorrow" value="false"/>
    <property name="testOnReturn" value="false"/>
    <property name="testWhileIdle" value="true"/>
```

数据库 jdbc 配置文件

```
validationQuery=SELECT 1

jdbc_url=jdbc:mysql://127.0.0.1:3306/jspm31pwr?useUnicode=true&characterEncoding=UTF-8&tinyInt1isBit=false&useSSL=false&s
jdbc_username=root
jdbc_password=chendanaini
```

Database Layer(数据库层)

数据库层使用 Mysql 数据库保存数据，使用 Navicat Premium 进行管理

场景构想---网上游戏代练平台

网上游戏代练平台与传统网页电商平台结构一致，使用分层架构将复杂的系统划分为多个层次，每个层次负责特定的功能和任务，可以使得系统设计更加清晰和简洁。前端表示层展示商品数据，接收交互数据表单，后端 controller 层负责在 view 和 service 层转发请求数据，service 负责对请求进行业务处理，持久层表示数据实体和调用数据库中数据。分层架构可以使得不同的开发人员专注于不同的层次或组件的开发工作，从而实现了任务的划分和协作。