<交集>

软件架构文档

版本 <2.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <03/04/2023> | <1.0> | <软件架构设计> | <张奕涵，杜心敏，全雨乐，胡彤，杨菡雪> |
| <17/06/2023> | <2.0> | <修改逻辑视图> | <张奕涵，杜心敏，全雨乐，胡彤，杨菡雪> |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 5

2. 用例视图 5

3. 逻辑视图 8

3.1 MVC架构图 8

3.2 逻辑视图中的重要包 9

4. 进程视图 10

4.1 进程视图 10

4.2 时序图 11

4.3 实施过程 12

5. 部署视图 13

6. 实现视图 14

7. 技术视图 15

8. 数据视图 17

9. 核心算法设计 18

10. 质量属性的设计 18

软件架构文档

1. **简介**

1.1 **目的**

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

**本文档包含以下几个部分：**

* 用例视图
* 逻辑视图
* 进程视图
* 部署视图（物理架构视图）
* 实现视图
* 技术视图
* 数据视图
* 核心算法设计
* 质量属性的设计

**本文档读者包括：**

* 本系统的开发人员（把握用户需求、为编码提供框架）
* 客户（检查是否满足需求）
* 课程老师和助教（检查是否符合课程要求）

1.2 **参考资料**

无

2. **用例视图**

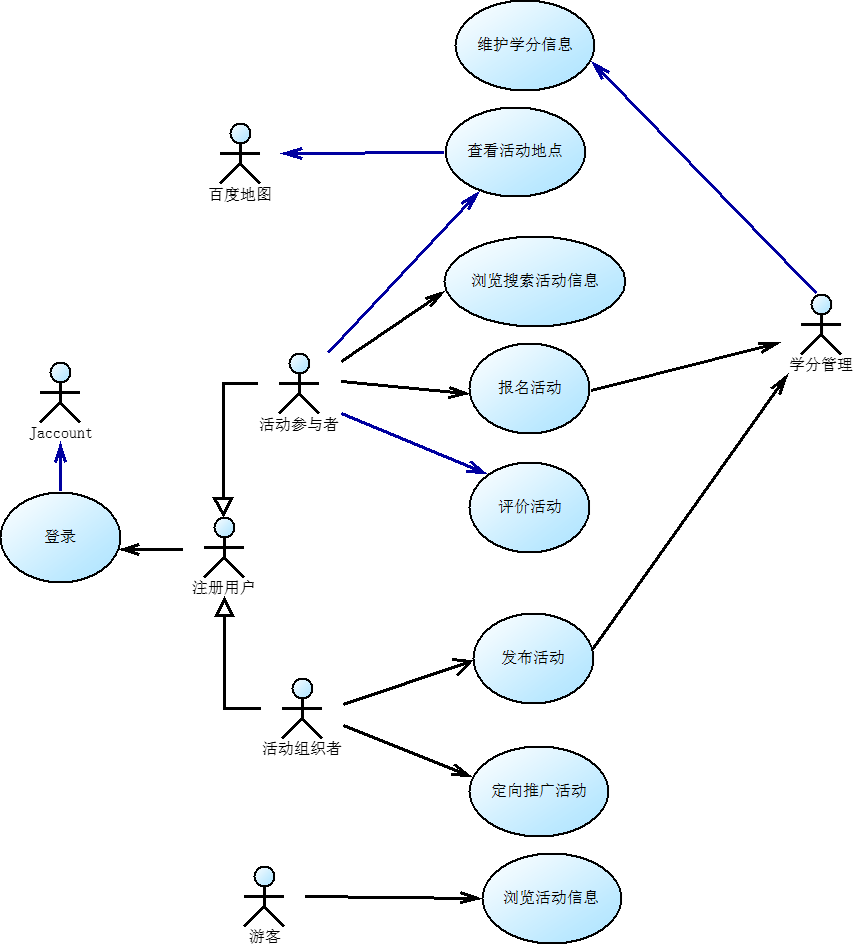
软件体系结构用例视图的描述。用例视图是选择场景集或迭代的焦点用例的重要输入，它描述了一组表示一些重要的中心功能的用例。

活动集成平台的用例有：

* 登录
* 浏览搜索活动信息
* 报名活动
* 评价活动
* 发布活动
* 定向推广活动
* 浏览活动信息
* 维护学分信息
* 查看活动地点

这些用例是由注册用户（活动发布者、活动参与者）、游客、学分管理员、高德地图发起。

2.1 **架构上的重要用例**



2.1.1 **登录**

**简要描述:** 这个用例描述了用户如何登录到活动平台。已注册用户可直接通过账号密码登录，否则进入注册流程或选择其他登陆场景。开始这个用例的角色是注册用户，包括活动参与者和活动发布者。

2.1.2 **浏览搜索活动信息**

**简要描述：** 这个用例允许活动参与者浏览、搜索已经发布的活动信息。这个用例的角色是活动参与者。

2.1.3 **报名活动**

**简要描述：** 这个用例允许活动参与者修改自己的活动列表，包括报名参与活动和取消报名。活动参与者可以从已发布活动列表中选择感兴趣的活动报名参与，提交报名信息；也可以从已报名活动列表取消报名某个活动。这个用例的角色是活动参与者。

2.1.4 **评价活动**

**简要描述：** 这个用例允许活动参与者在参与活动后发布对活动的评价和反馈。这个用例的角色是活动参与者。

2.1.5 **发布活动**

**简要描述：** 这个用例描述了活动发布者如何发布一个活动。这个用例的角色是活动发布者。

2.1.6 **定向推广活动**

**简要描述：** 这个用例允许活动发布者选择可以报名该活动的参与者，不符合条件的活动参与者无法在活动列表浏览到该活动信息。这个用例的角色是活动发布者。

2.1.7 **浏览活动信息**

**简要描述：** 这个用例允许未注册登录用户浏览平台上发布的活动信息，但无法报名活动。这个用例的角色是游客。

2.1.8 **维护学分信息**

**简要描述：** 这个用例描述了用户参与或发布活动之后如何影响学分信息。学分管理员可通过发布者发布的活动信息和报名者参与活动信息维护学生学分，包括赋予活动学分、修改活动参与者学分。这个用例的角色是学分管理员。

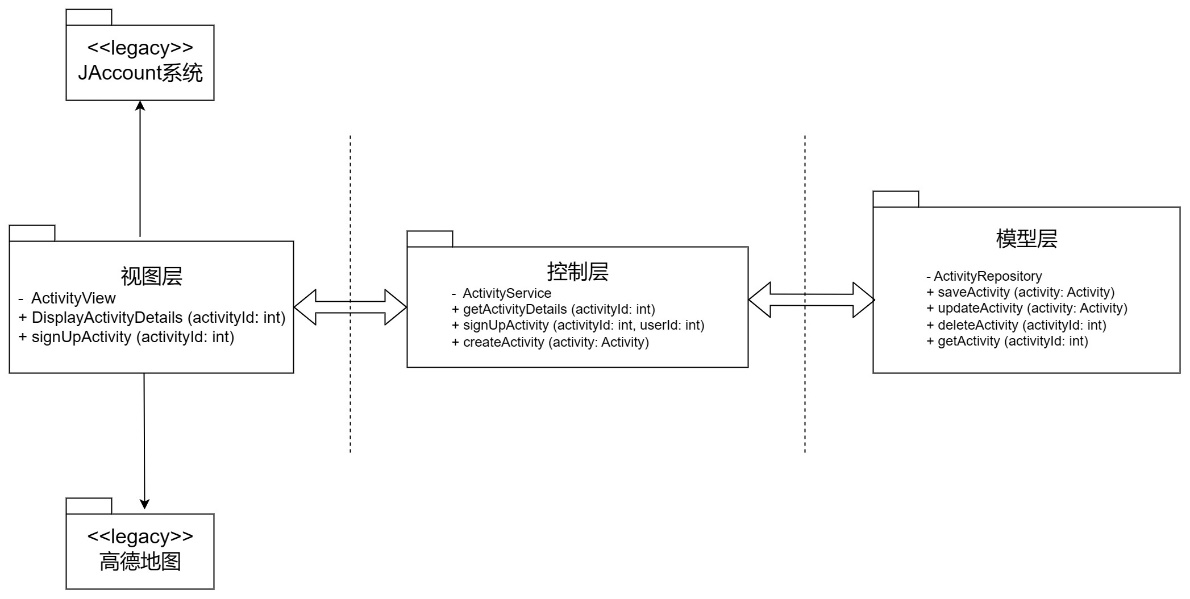
2.1.9 **查看活动地点**

**简要描述：** 这个用例允许活动参与者在成功报名活动后能够通过高德地图服务查看活动地点。这个用例的角色是活动参与者和高德地图。

3. **逻辑视图**

逻辑视图用于描述系统的功能需求，以及描述系统软件功能拆解后的组件关系、组件约束和边界，反映系统整体组成与系统如何构建的过程。包括前端的逻辑架构与JAccount的显式核对以及各种平台内服务，以及控制层下的抓取系统，存储系统，数据库以及互联网是如何影响软件的后端，如何进行关联的。

3.1 **MVC架构图**



逻辑架构:

1. 视图层 (View Layer)

* 交集前端：网站的用户界面，提供给用户直观的操作和浏览功能。
* 平台内服务：为交集前端提供服务支持，如用户登录、活动信息等。

1. 控制层 (Controller Layer)

* 前端逻辑架构：处理交集前端与后端之间的通信，负责将用户请求传递给后端。
* 存储系统：负责管理数据存储，如用户信息、活动数据等。
* 抓取系统：从互联网上获取活动信息，并将其传递给后端相应处理。
* 数据库：存储抓取系统获取的活动信息和用户信息等数据。
* Jaccount接口：用于用户登录验证和权限管理。

1. 模型层 (Model Layer)

* 活动仓库：负责处理活动数据的存储和管理。
* 保存活动：将新的活动信息存入数据库。
* 更新活动：更新已有活动的信息。
* 删除活动：根据活动ID删除对应的活动信息。
* 获取活动：根据活动ID从数据库中获取对应的活动信息。

通过上述架构，交集网站可以实现用户浏览、查询和参与校园活动的功能。同时，系统内部各个组件之间的分工明确，有助于提高系统的稳定性和可扩展性。

3.2 **逻辑视图中的重要包**

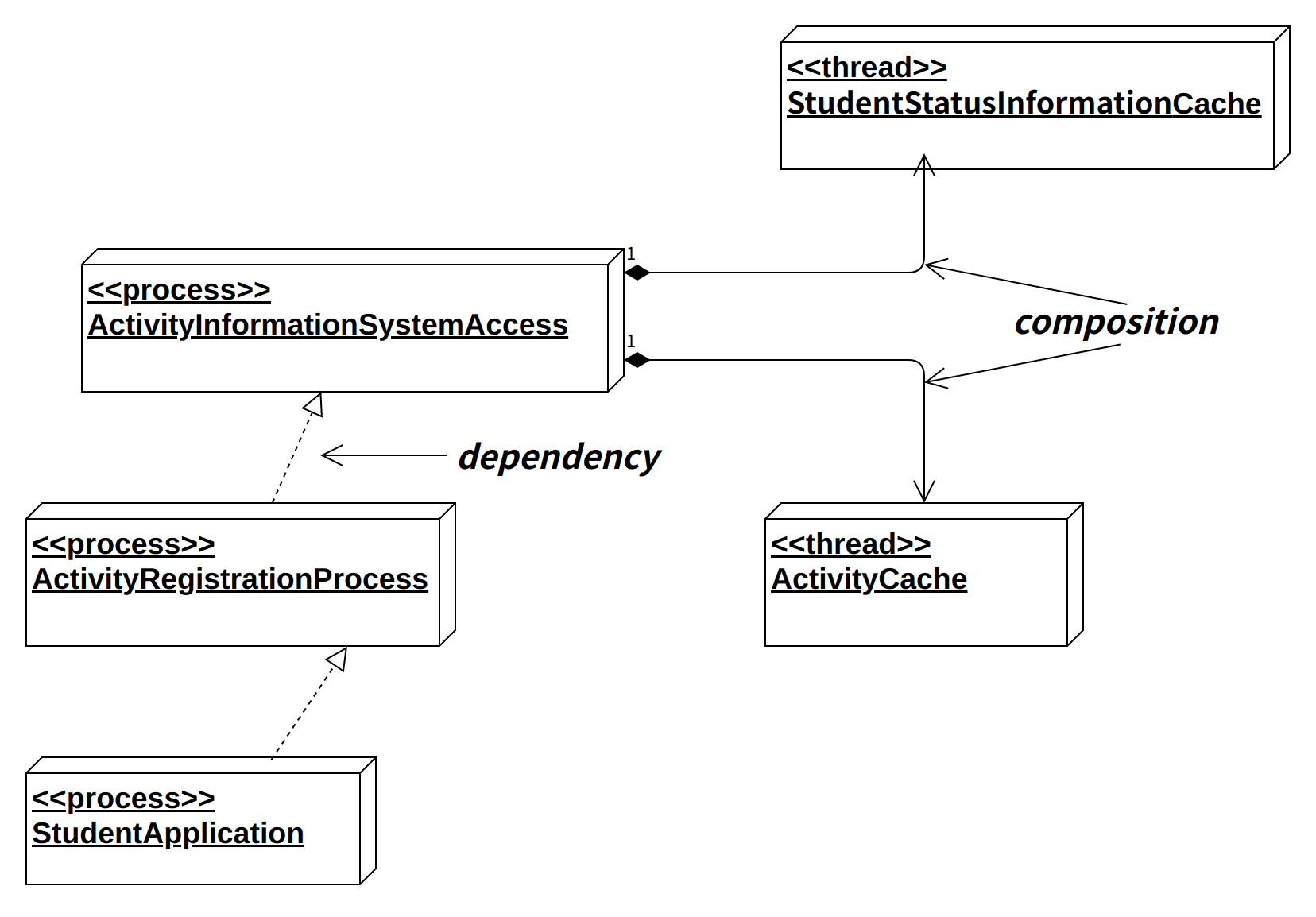
逻辑视图由 3 个主要包组成：用户界面、业务服务和业务对象。

1. 用户界面包：包含参与者用于与系统通信的每个形式的类。边界类的存在是为了支持注册、登录、发布活动、维护活动列表、维护活动信息、选择活动、提交报名申请、维护学生学分等信息、结束活动、查看反馈等功能。
2. 业务服务包：包含用于与JAccount系统接口、控制学生注册和管理学生学分等学籍信息的控制类。
3. 业务对象包：包括活动管理系统的实体类,学生学籍信息相关的实体类和与Jccount系统接口的边界类。

4. **进程视图**

进程模型说明了组织为可执行流程的活动相关操作类。存在支持注册、登陆、发布活动功能以及访问外部JAccount系统和活动列表的流程。

4.1 **进程视图**



1. ActivityInformationSystemAccess

此过程管理对活动信息系统的访问。它可以由多个用户共享。这允许缓存最近检索的校园活动以提高性能。

1. StudentStatusInformationCache

此过程能够实时更新学生用户状态信息，包括与用户相关的活动信息列表，已报名、仅浏览未报名等状态信息和学分等学籍信息。

1. ActivityCache

此进程从旧版活动列表系统中异步检索项目。

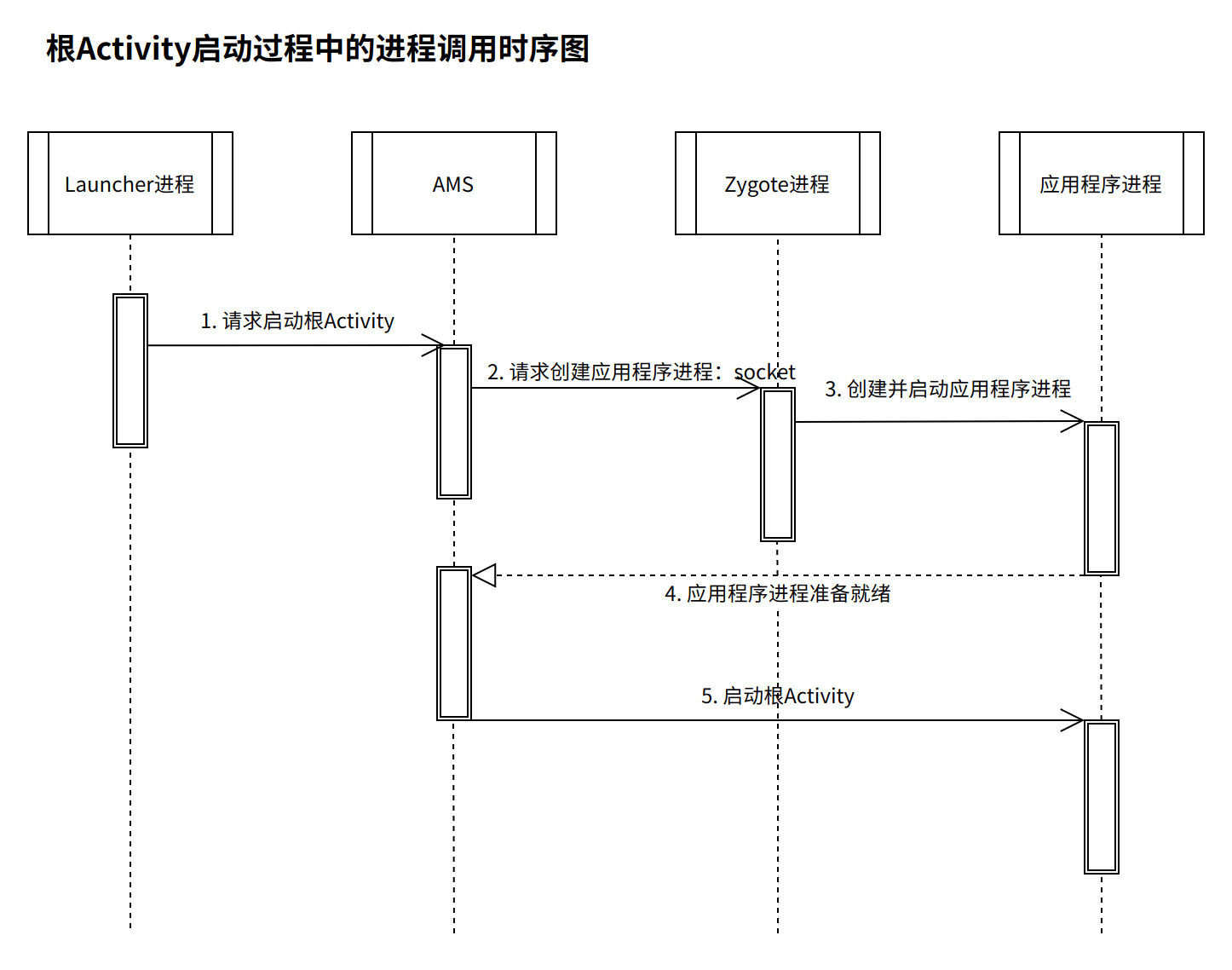
1. ActivityRegistrationProcess

此过程管理学生报名，实时跟踪报名活动的信号。

1. StudentApplication

此过程管理学生功能，包括用户界面处理和与业务流程的协调。对于当前正在报名或浏览活动的每个学生用户，此过程都有一个实例。

4.2 **时序图（Sequence Diagram）**



此时序图描绘了进程交互关系。

4.3 **实施过程**

4.3.1 **远程**

* 远程接口用于识别所有远程对象。作为远程对象的任何对象都必须直接或间接实现此接口。只有远程接口中指定的方法才能远程使用。
* 实现类可以实现任意数量的远程接口，并且可以扩展其他远程实现类。

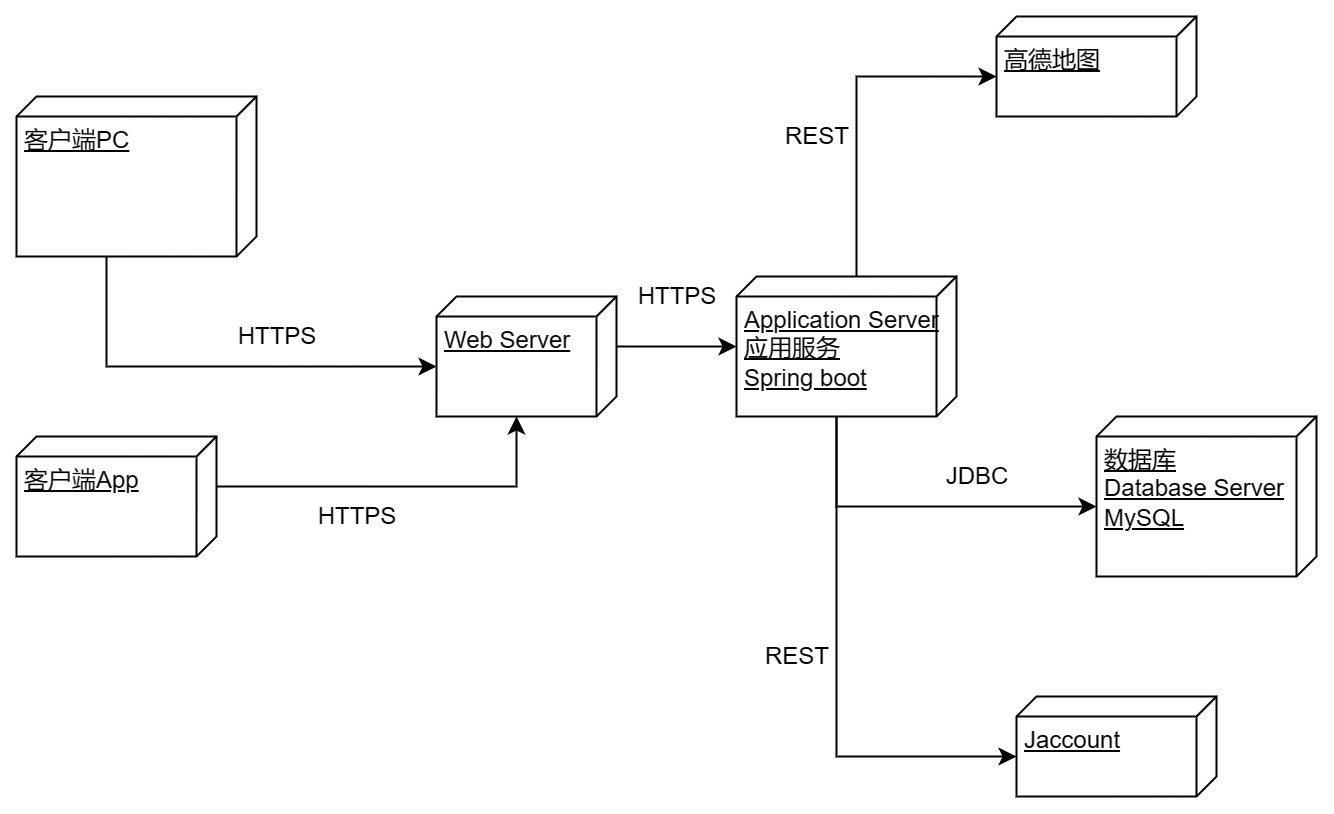
4.3.2 **可运行**

* 可运行接口应由其实例旨在由线程执行的任何类实现。该类必须定义一个名为 run 的无参数方法。
* 此接口旨在为希望在活动时执行代码的对象提供通用协议。例如，Runnable 由类 Thread 实现。
* 处于活动状态仅表示线程已启动且尚未停止。

4.3.3  **线程**

* 线程是程序中的执行线程。Java 虚拟机允许应用程序同时运行多个执行线程。
* 每个线程都有一个优先级。优先级较高的线程优先于优先级较低的线程执行。每个线程也可能不会被标记为守护程序。当在某个线程中运行的代码创建新的 Thread 对象时，新线程的优先级最初设置为与创建线程的优先级相同，并且当且仅当创建线程是守护程序时，该线程才是守护程序线程。

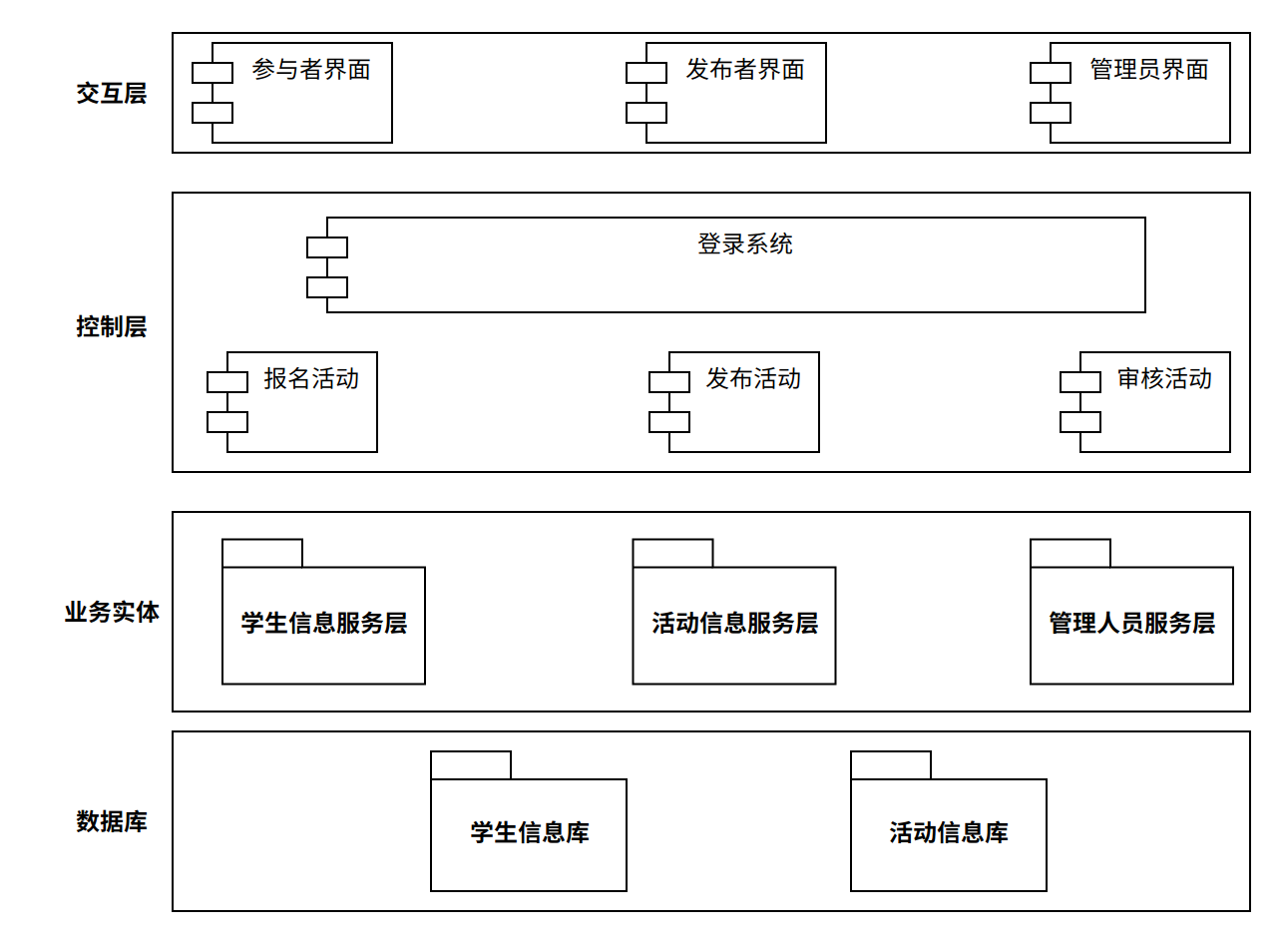
5. **部署视图**



体系结构部署视图的描述了“交集”平台配置的各种物理节点，以及将任务分配到物理节点的过程。以下是对每个物理节点的详细解释：

1. 客户端PC：用户可以通过浏览器访问校园活动信息平台，通过浏览器向Web服务器发出HTTPS请求。
2. 客户端APP：用户可通过在移动设备上的客户端App访问校园活动信息平台，通过移动应用程序向Web服务器发出HTTPS请求。
3. Web服务器层：负责接收和处理客户端的HTTPS请求，并返回响应结果，如Apache或Nginx。
4. 应用服务器层：位于Web服务器与数据库之间，负责处理业务逻辑，包括身份认证、权限控制、数据管理等。应用服务器主要采用Java EE技术栈，例如Spring、Struts、Hibernate等框架。
5. 数据库层：存储和管理系统中的数据，包括用户信息、活动信息等。数据库采用MySQL、SQL Server等关系型数据库。
6. 高德地图：提供地理位置信息和地图服务，用于显示活动地点和提供导航功能。
7. Jaccount：用于验证用户的身份；根据用户账号的权限控制用户访问各种特定功能和资源，如发布活动、学分管理等。

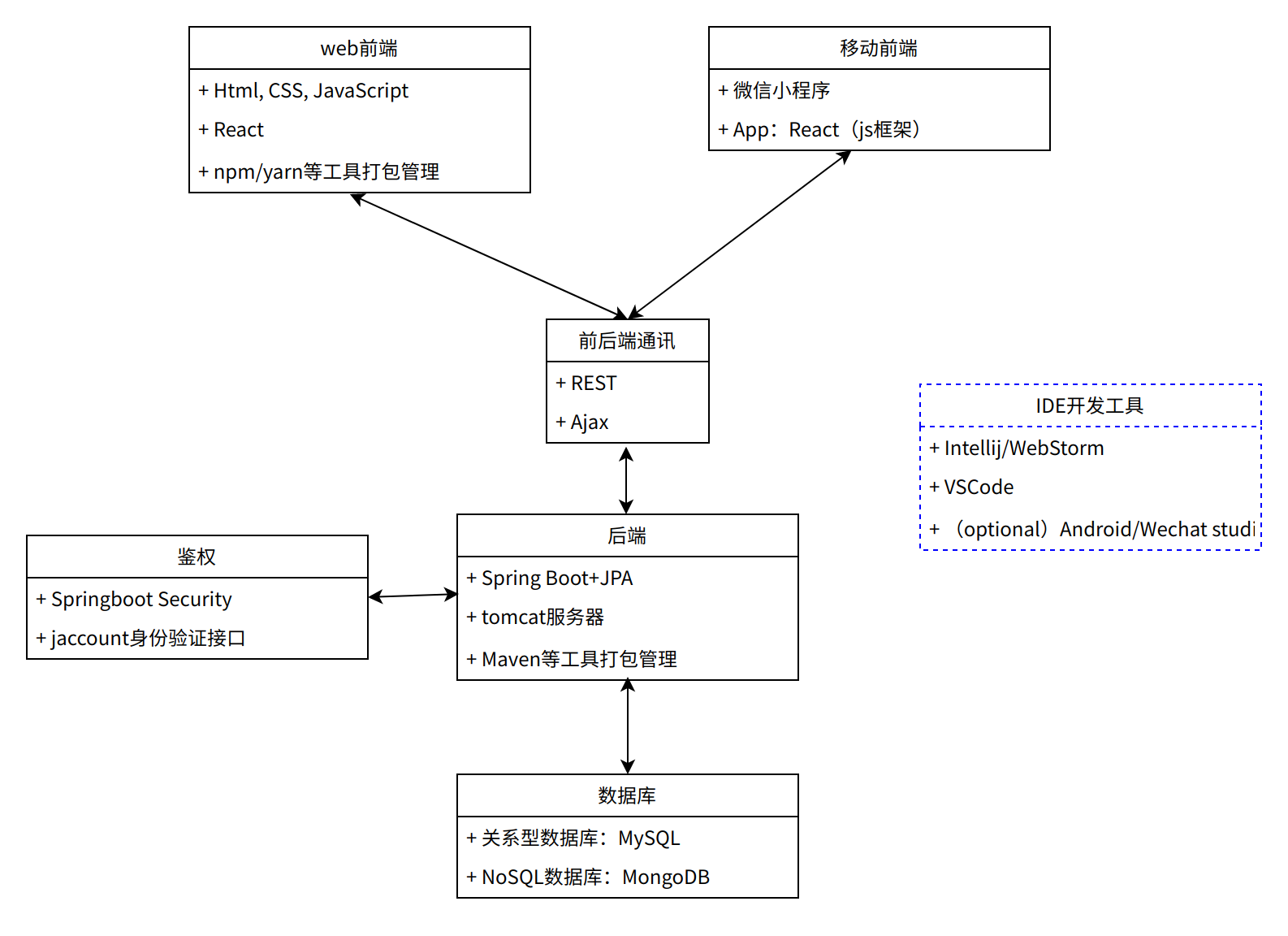
6. **实现视图**



开发视图采用分层样式。划分为交互层（UI）、控制层、业务实体、数据库。每一层划分了相对独立的模块。某个子系统只能依赖于同一层或下层的子系统，以尽量减少模块之间依赖关系。

7. **技术视图**

[技术栈的选型，包括编程语言、开发工具、框架、数据库、中间件等。]



* Web前端

1. Html, CSS, JavaScript
2. React（js框架）
3. 使用npm/yarn等工具打包管理

* 服务器后端

1. Spring Boot+JPA实现
2. Java语言
3. tomcat服务器
4. Maven等工具打包管理

* 前后端通讯

1. REST （Restful web services）、Ajax

* 数据库

1. 关系型数据库：MySQL/Mariadb
2. NoSQL数据库：MongoDB

* 鉴权

1. Springboot Security
2. jaccount身份验证

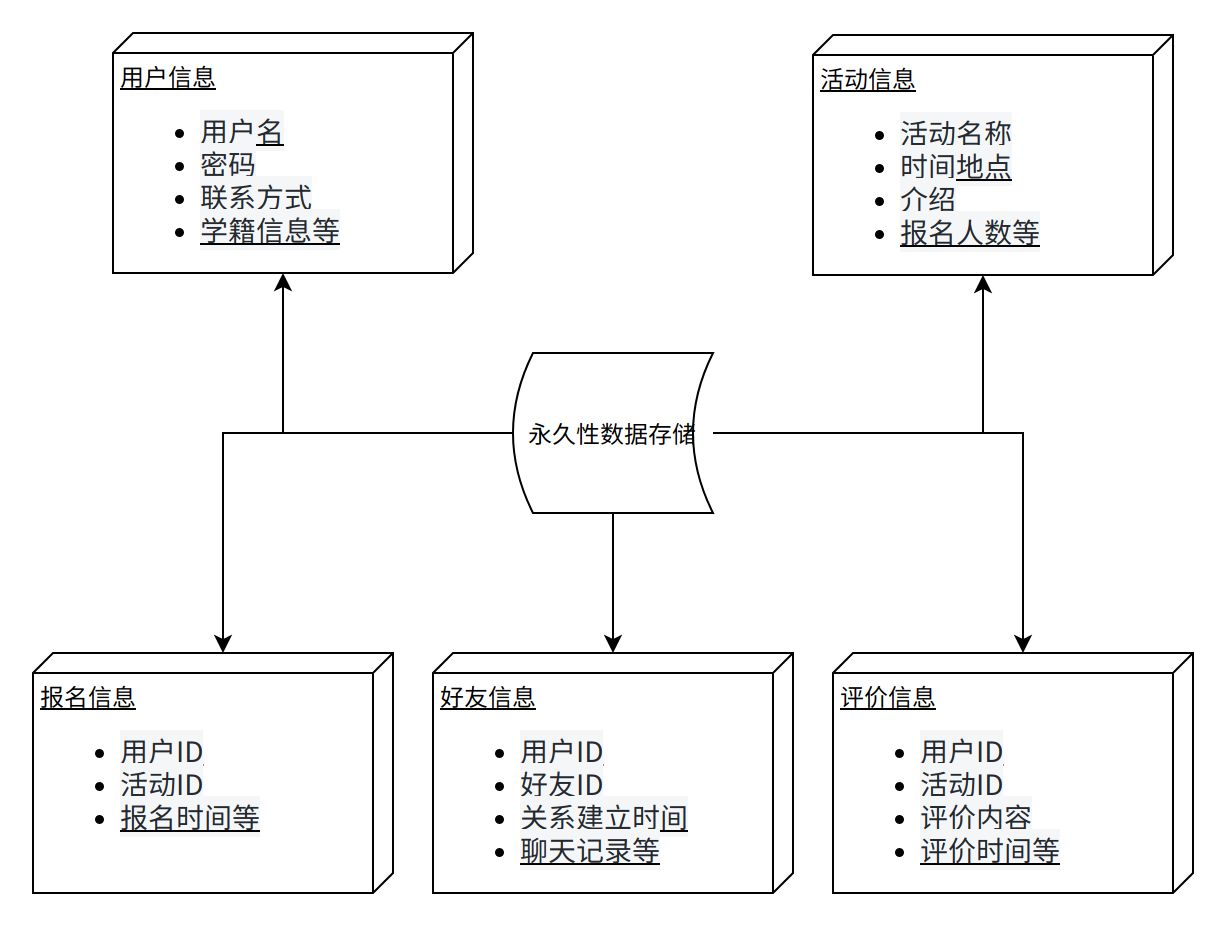
* IDE开发工具

1. IntelliJ / WebStorm
2. Visual Studio Code
3. Android Studio（optional）
4. WeChat Studio（optional）

8. **数据视图**

[从永久性数据存储方面来对系统进行说明。如果几乎或根本没有永久性数据，或者设计模型与数据模型之间的转换并不重要，那么本节就为可选。]

[如果需要，可增加其他视图。]



1. 用户信息：包括用户名、密码、邮箱、个人信息等，这些信息需要被永久性存储以便用户下次登录时恢复信息。
2. 活动信息：包括活动名称、时间、地点、介绍、报名人数等，这些信息需要被永久性存储以便用户可以查看历史活动信息，并且发布者可以更新活动信息。
3. 报名信息：包括用户ID、活动ID、报名时间等，这些信息需要被永久性存储以便可以统计活动的报名人数，同时也可以让用户查看历史报名信息。
4. 评价信息：包括用户ID、活动ID、评价内容、评价时间等，这些信息需要被永久性存储以便可以记录历史评价信息，并且可以给活动评分和排序。
5. 好友关系：包括用户ID、好友ID、关系建立时间等，这些信息需要被永久性存储以便用户可以查看历史好友关系，并且在系统内部发送聊天信息时可以找到对应的好友。

9. **核心算法设计**

推荐算法设计:

1. 基于用户兴趣爱好的推荐：我们可以使用协同过滤算法，通过分析用户历史行为和偏好，以及活动的属性特征，建立用户和活动的关联矩阵，然后通过计算相似度来推荐相似类型的活动。
2. 热门活动推荐：我们可以使用基于流行度的推荐算法，通过统计每个活动的浏览量、收藏量、参加量等数据，然后按照一定的规则进行加权排序，最后推荐排名前几的活动。

10. **质量属性的设计**

1. 性能方面：我们采用分布式架构，将系统拆分成多个子系统。同时，我们会针对网站特性进行优化，以提高各自的性能。缓存技术也会继续应用，以加快响应速度。
2. 可扩展性方面：我们将使用分层、分块架构，保证网站版本的可扩展性。后期可根据需求，独立部署和扩展相关服务。
3. 可靠性方面：我们对前后端进行分离，采用主从复制和数据备份技术，保证数据的一致性和可靠性。
4. 易用性方面：我们将根据网站的特点，进行相应的界面设计和交互设计，以提供良好的用户体验。新用户登录时会配置教程。
5. 可移植性方面：采用可跨平台的开发框架，确保网站可以在不同的浏览器和操作系统上运行。