# 基于PPT以及课堂自测的SA复习笔记

#### Software Architecture = Components + Connectors + Constrains（组件+连接件+约束）

软件体系结构不存在统一定义。

## A software architecture style(软件体系结构风格)

* describes a class of architectures(描述一类体系结构)
* independent on the problems(独立于实际问题，强调了软件系统中通用的组织结构)
* found repeatedly in practice(在实践中被多次应用)
* a package of design decisions(是若干设计思想的综合)
* has known properties that permit reuse(具有已经被熟知的特性，并且可以复用)

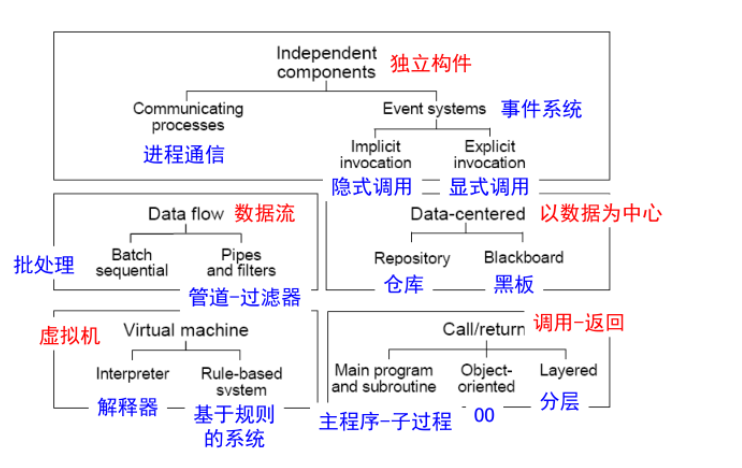
## 描述特定领域中软件系统家族的组织方式的惯用模式(idiomatic paradigm)

## 反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

* Architecture style ={Component/Connector vocabulary,

Topology,Semantic Constraints

}



## Data Flow Software Architecture Style（数据流）

### A data flow system is one

in which

* the availability of data controls the computation (数据的可用性决定着处理<计算单元>是否执行)（数据流体系风格特点：无数据不工作）
* the structure of the design is decided by orderly motion of data from process to process(系统结构由数据在各处理之间的有序移动决定)
* in a pure data flow system,there is no other interaction between processes(在纯数据流系统中，处理之间除了数据交换没有任何其他的交互)
* Components：data processing components
* Interface：input ports+output ports
* Computational model：read data from input ports +compute +write data output
* Connectors: data flow(stream) (unidirectional,usually asynchronous,buffered单向，通常异步，有缓冲）

### 典型数据流风格

* Process Control(过程控制)
* Pipe and Filter(管道-过滤器)
* Batch Sequential(批处理)

### Batch Sequential(批处理)

* Components(processing steps)： independent programs(基本构件：独立的应用程序)
* Connectors ： some type of media -traditionally tape(连接件：某种类型的媒质)
* Topology: Connectors define data flow graph(连接件定义了相应的数据流图，表达拓扑结构)
* Processing steps are independent programs(每个处理步骤是一个独立的程序)
* Each step runs to completion before next step starts(每一步必须在前一步结束后才能开始)
* Data transmitted as a whole between steps(数据必须是完整的，以整体的方式传递)（批处理体系结构风格里，数据传输方式为）

### Pipe and Filter(管道-过滤器)

Filters are independent entities.（请简要说惻管愷-过滤恷体系结构风格惲，过滤恷恿互独立懨恾义。）

* no context in processing streams(无上下文信息)
* no state preservation between instantiations(不保留状态)
* no knowledge of upstream/downstream filters(对其他过滤器无任何了解)

（管道-过滤器：0.概念：构件：过滤器；连接件：管道；每个过滤器都有一组输入和输出，过滤器从管道中读入数据流，对输入流进行内部处理，然后产生输出数据流并写入管道中；1. 优点：良好的隐蔽性，高内聚、低耦合；便于设计者理解；支持功能模块的重用；系统易于维护和扩展；支持某些特定属性的分析；支持并发执行；

1. 缺点：不适合于交互性很强的应用；数据传输无通用标准，系统性能不高；需要处理同步问题。
2. 应用实例：Unix管道，图像处理;）

## CALL/RETURN Software Architecture Style

### 主程序/子程序

* Problem: This pattern issuitable for applications in which the computationcan appropriately

be defined via a hierarchy of procedure definitions.

* Context: Many programming languages provide natural support for defining nested collections of procedures and for calling them hierarchically.These languages often allow collections of procedures to be grouped into modules,thereby introducing name-space locality.The execution environment usually provides a single thread of control in a single name space.
* Solution:

1. System model:call and definition hierarchy（调用和定义层次结构）,subsystems often defined via modularity（子系统通常通过模块化来定义）
2. Components: procedures and explicitly visible data（过程和显式可见的数据）
3. Connectors: procedure calls and explicit data sharing（过程调用和显式数据共享）
4. Controlstructure: single thread

subsystems often defined via modularity：

模块分解的目的是将系统“分而治之”，以降低问题的复杂性，是软件结构清晰，易阅读、易理解、易于测试和调试，因而也有助于提高软件的可靠性。

##### 总结：

* 主程序/子程序风格

单线程控制，划分为若干处理步骤

功能模块：把步骤集成至模块中

主程序-子过程是从功能的观点设计系统通过逐步分解和细化，形成整个系统的体系结构。

批处理则是从数据流的观点出发，批量处理某个对象。

### 抽象数据类型(ADT)

操作和数据绑定在一起，隐藏实现和其他秘密

### 面向对象

* 方法(动态绑定),多态(子类),重用(继承)
* 对象活动于不同的进程/线程(分布式对象)：CS结构、分层风格

·Encapsulation:Restrict access to certain information

封装：限制对某些信息的访问

·Interaction:Via procedure calls or similar protocol

交互：通过过程调用或类似的协议

·Polymorphism:Choose the method at run-time

多态：在运行时选择具体的操作

·Inheritance:Shared definitions of functionality

继承：对共享的功能保持唯一的接口

·Reuse and maintenance:Exploit encapsulation and locality to increase

productivity

复用和维护：利用封装和聚合提高生产力

（选择面向对象系统的原因：构件：对象；连接件：消息和消息（过程）调用；适用：界面与实现分离的系统

面向对象的优缺点：1.优点：面向对象易维护，易复用；对象反映现实世界，容易分解一个系统；对象对客户实现了隐藏细节，所有可以在不影响其客户的情况下改变对象的实现 ；2.缺点：对象之间的耦合度较紧：一个对象和另一个对象进行交互时，必须知道对象的标识。只要一个对象的标识改变了，必须修改所有显式调用它的其它对象）

### 层次风格

·每层为上一层提供服务，使用下一层的服务，只能见到与自己邻接的层。

适当时候(必不得已的时候),可以允许一定的越层操作。

·大的问题分解为若干个渐进的小问题，逐步解决，隐藏了很多复杂度

·修改一层，最多影响两层，而通常只能影响上层。接口稳固，则谁都不影响

·上层必须知道下层的身份，不能调整层次之间的顺序

·层层相调，影响性能

#### Client-server

-Objects are processes

进程就是对象

-Asymmetric: client knows about servers,but not vice versa

不对称：客户端知道服务器，反之则不然

请简要描述两层，三层c/s与b/s风格区别：

（1）B/S与C/S区别： 1、c/s架构主要应用于局域网内，而b/s架构主要应用于广域网中； 2、c/s架构一般面向相对固定的用户群，对信息安全的控制能力很强，而b/s架构对安全的控制能力相对弱； 3、B/S架构维护升级比较简单，而C/S架构维护升级相对困难。

（2）两层/三层区别：三层结构将界面与业务逻辑分开,而两层结构中界面与业务逻辑是不分的,这样的好处是今后的维护方便,便于功能的扩充. 三层好比是第一层画图，第二层写代码，第三层处理数据。

### 组件

多个接口，二进制兼容，高级中间件

## Virtual Machine Software Architecture Style

* Interpreters(解释器）
* Rule-based systems（规则系统）
* Other（syntactic shells - commmand language processors）

虚拟机体系结构：包括解释器和基于规则的系统；解释器：构件：一个解释器引擎和三个存储区；连接器：对存储区数据的访问和过程调用 1.优点： 可以模拟非本地原生支持的功能；可以在模拟极端条件监测系统；用途广泛 2.缺点：效率低 ；额外软件层的正确性需要验证

## Data-centered Software Architecture Style

* Data-centered Software Architecture involve a shared data source approach to information passing.

### Repository

>A repository is a central place where data is stored and maintained.

(仓库是存储和维护数据的中心场所)

In a repository style there are two quite distinct kinds of components:

* A central data structure representing the current state;(中心数据结构，表示当前数据的状态)
* A collection of independent components operate on the central data store.(一组对中心数据进行操作的独立构件)

>Connector:Interactions between the repository and its external components.(连接件：仓库与独立构件之间的交互)

>Two major mechanisms:(存在两种交互机制)

1. Database:the types of transactions in an input stream trigger selection of process to execute;(数据库方式：输入流中的事务类型触发需要执行的过程)
2. Blackboard:the current state of the central data structure is the main trigger for selecting processes to execute.(黑板结构：中心数据结构的当前状态触发并选择需要执行的过程)

Problem:This pattern is suitable for applications in which the central issue is establishing,augmenting,and maintaining a complex central body of information.

典型应用场合：

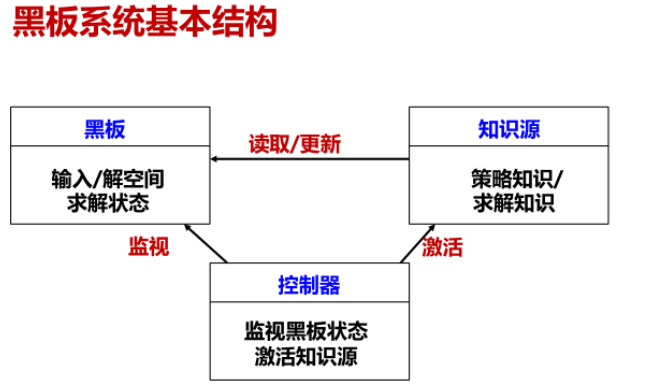
√Data processing(数据处理),driven primarily by the need to build business decision systems from conventional

databases.

√Software development environments(软件开发环境),driven primarily by the need to represent and manipulate programs and designs.

### 黑板体系结构

黑板结构：中心数据结构的当前状态触发并选择需要执行的过程



#### Blackboard data structure黑板数据结构

>Global database containing entire state of problem solution

(全局数据库，用来存储数据、传递信息，包含解域的全部状态)

problem-solving state data,organized into an application-dependent hierarchy.(解决问题过程中的状态数据，以层次形式组织起来)

>knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.(知识源对黑板进行修改，逐渐找到问题的解)

Only means by which knowledge sources interact(各知识源之间的通讯和交互只通过黑板进行)

#### Control控制器

>时刻监视黑板状态变化 对黑板上信息的当前状态进行判断和评价

>当黑板的状态满足了知识源的执行条件时，该知识源被控制器触发并进行计算，然后将结果更新到黑板上，这种更新又导致其他知识源参与计算并更新黑板，直到找到问题解为止



## Event System Software Architecture Style

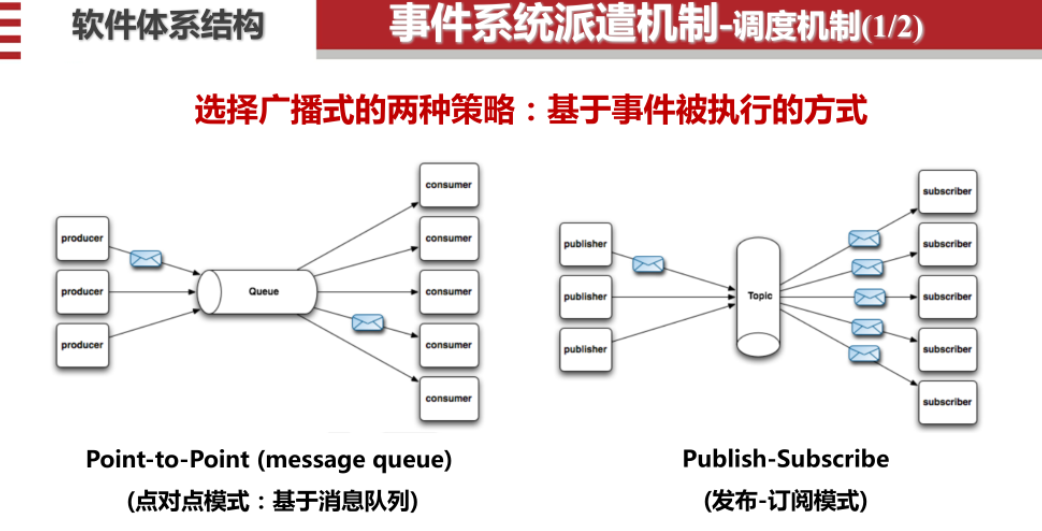
* 隐式调用

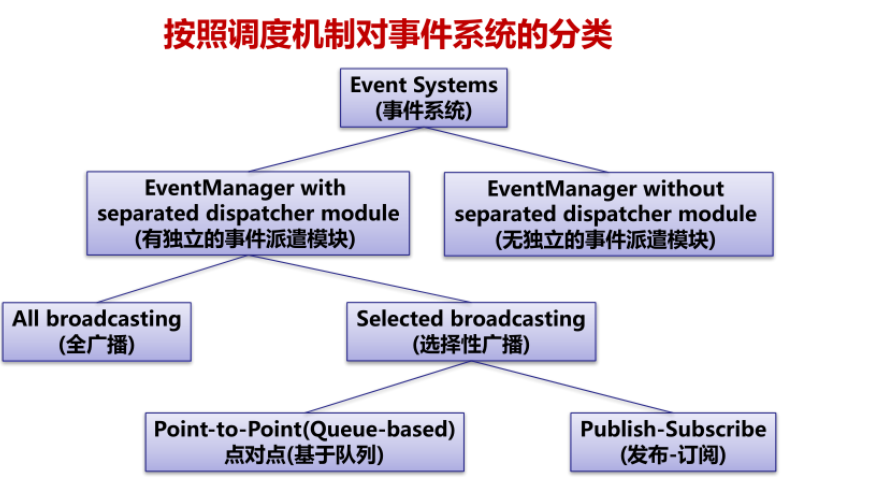
>主要特点

√事件的触发者并不知道哪些构件会被这些事件影响，相互保持独立。

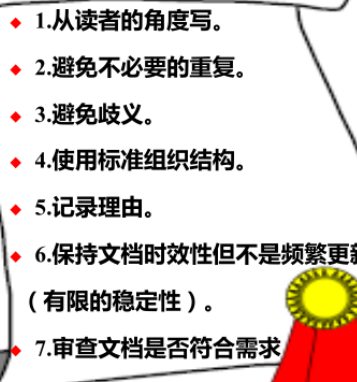
√不能假定构件的处理顺序，甚至不知道哪些过程会被调用。

√各个构件之间彼此之间无连接关系，各自独立存在，通过对事件的发布和注册实现关联。





## 文档部分（应该会考一个题目）



## 4+1UML

* 硬件设备不属于软件体系结构的影响因素
* 采用“4+1”视图模型来描述软件系统的体系结构。在该模型中，最终用户侧重于逻辑视图；系统工程师侧重于部署视图
* 用例图定义了系统的功能需求，它是从系统的外部看系统功能，并不描述系统内部对功能的具体体现
* 物理视图（Physical view ）物理视图通常也叫做部署视图
* 开发试图和逻辑视图之间可能存在一定的映射关系
* 开发视图是逻辑视图的实现

· 逻辑视图：对于面向对象设计来讲，就是对象模型。

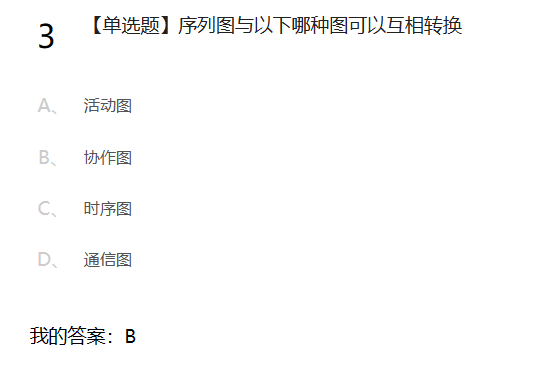
· 运行视图：涉及并发和同步等方面。

· 物理视图：展现软件到硬件的映射，以及软件的分布式部署。

· 开发视图：描述软件在开发环境中的静态组织方式。

· +1 场景视图：是其它几个视图的补充，用于通过use case将其它几个视图串联起来。

[4+1视图解读 - 简书 (jianshu.com)](https://www.jianshu.com/p/55778ec87ed8)



## 质量属性



* 是非功能性需求
* 质量属性场景的6个组成部分

刺激源(source):谁造成的刺激

刺激(stimulus):一个影响系统的情况

制品(artifact):系统被影响的部分

环境(environment):刺激发生时系统所处的状态

响应(response):刺激所产生的结果

响应衡量指标(response measure):如何评估响应

### 六个质量属性以及对应的提升策略

* 可用性：异常；心跳；主动冗余；被动~；回滚；Ping/echo；进程监控；投票；服务下线；事务；
* 可修改性：模块高内聚，低耦合；模块通用；使用中介；隐藏信息；维持语义一致；多态；发布订阅模式；限制可能的选择；
* 性能：限制待处理事件队列长度；限制执行时间；减少要处理的数据总量；提高计算效率； 先来先服务；固定优先级调度；动态优先级；控制采样频率；管理事件率；
* 安全性：维护数据完整和机密；验证用户；限制访问；用户授权；减少暴露；
* 可测试性：记录/回放；内部监控；接口和实现分离；
* 易用性：用户界面和系统其它部分隔离开；支持撤销；系统给用户反馈；系统猜测用户要完成的任务；系统给用户提供一致的体验



安全性案例：所谓SQL注入，就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令，比如先前的很多影视网站泄露VIP会员密码大多就是通过WEB表单递交查询字符暴出的，这类表单特别容易受到SQL注入式攻击．当应用程序使用输入内容来构造动态sql语句以访问 数据库 时，会发生sql注入攻击。 如果代码使用存储过程，而这些存储过程作为包含未筛选的用户输入的字符串来传递，也会发生sql注入。 黑客通过SQL注入攻击可以拿到网站 数据库 的访问权限，

## Software Architecture Evaluation

四个软件评估概念

风险点：是一个潜在的有问题的架构决策；failure

非风险点：是安全的良好架构决策分析，可以提高质量帮助实现目标的决策；e.g.假定消息的到达速率是每秒一次，一次处理的时间小于30ms。如果对一个更高优先级的处理的响应时间要求是1秒钟，此系统可行；

敏感点：是一个或多个组件的属性，对于实现特定的质量属性响应至关重要；e.g. 虚拟专用网络的保密级别可能对加密encryption的位数 bits很敏感；封装encapsulated维护maintain量大modify（小变化产生大影响）

权衡点：是影响多个属性的属性，是多个属性的敏感点；e.g.改变加密级别可能会对安全性security和性能performance产生重大影响

ATAM（Architecture Tradeoff Analysis Method）是一种软件体系结构评估方法，它主要关注于分析软件体系结构的优点和缺点，以便在设计和开发过程中做出明智的决策。以下是ATAM软件体系结构评估的流程、涉及人员及关键技术的简要叙述：

评估流程：

1. 评估准备：确定评估的目标和范围，组建评估团队，并准备相关的文档和资料。

2. 描述ATAM方法：评估小组负责人向参与评估的风险承担者介绍ATAM评估方法，解释评估的原则、方案和目标。

3. 描述商业动机：项目经理或系统客户从业务角度介绍系统的概况，包括业务环境、背景和业务约束等。

4. 描述体系结构：首席架构师或相关人员介绍项目的体系结构，包括设计理念和实现方式。

5. 确定体系结构方法：评估小组确定采用的体系结构方法，为后续分析提供依据。

6. 生成质量属性效用树：评估团队根据体系结构方法和商业动机，生成质量属性效用树，用于分析体系结构在质量属性方面的表现。

7. 分析体系结构方法：评估团队对体系结构进行深入分析，识别潜在的问题和风险，并提出改进措施。

8. 讨论和分级场景：评估团队与风险承担者一起讨论并分级重要的系统使用场景，以评估体系结构在不同场景下的表现。

9. 描述评估结果：评估团队总结评估结果，并向相关人员报告评估发现和建议。

涉及人员：

1. 评估小组负责人：负责整个评估过程的组织和协调。

2. 项目经理或系统客户：提供商业动机和业务背景信息。

3. 首席架构师或相关人员：介绍项目的体系结构。

4. 评估团队成员：包括开发人员、测试人员、维护人员等，他们参与体系结构的分析和讨论。

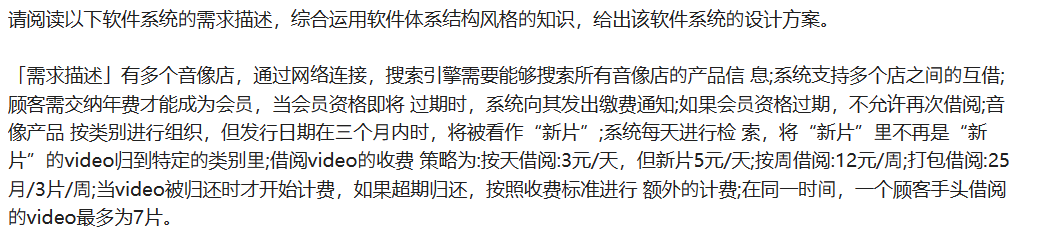
关键技术：

1. 质量属性效用树：这是一种用于分析体系结构在质量属性方面表现的工具，它可以帮助评估团队识别关键的质量属性并评估体系结构在这些属性上的表现。

2. 场景评估法：ATAM属于场景评估法的一种，它关注于挑选出重要的系统使用场景，并根据不同场景中各架构的表现进行评估。这种方法可以确保评估结果更加贴近实际使用情况。

综上所述，ATAM软件体系结构评估流程涉及多个阶段和人员，通过采用关键技术和方法，可以全面评估软件体系结构的优点和缺点，为设计和开发过程提供有价值的指导。

## Applications



基于所给的需求描述，我们可以采用以下软件体系结构风格来设计这个音像店网络系统：

\*\*软件体系结构风格\*\*：

1. \*\*分层体系结构\*\*：系统可以划分为表示层、业务逻辑层和数据访问层。表示层负责与用户交互，业务逻辑层处理核心逻辑，数据访问层则负责数据存取。

2. \*\*客户端-服务器架构\*\*：多个音像店作为客户端，通过网络连接到中央服务器。服务器负责集中处理搜索、借阅、归还、费用计算等逻辑。

3. \*\*事件驱动架构\*\*：对于会员资格过期通知、新片归类等事件，系统可以采用事件驱动的方式进行处理。

\*\*设计方案\*\*：

\*\*1. 数据模型设计\*\*：

- 会员表：包含会员ID、姓名、联系方式、年费缴纳状态、年费缴纳日期等信息。

- 产品表：包含产品ID、产品名称、发行日期、类别、价格等信息。

- 借阅记录表：包含借阅ID、会员ID、产品ID、借阅日期、应还日期、实际归还日期、借阅类型（按天/彝/打包）、费用等信息。

\*\*2. 系统功能模块\*\*：

- \*\*会员管理模块\*\*：负责会员的注册、续费、资格审核和过期通知。

- \*\*产品管理模块\*\*：管理音像产品的信息，包括添加、修改、删除、归类等。特别地，需实现自动检测新片的功能，并将不再属于新片范围的video归到对应类别。

- \*\*搜索模块\*\*：通过网络连接，实现跨店搜索功能，检索所有音像店的产品信息。

- \*\*借阅管理模块\*\*：处理顾客的借阅请求，检查会员资格、借阅数量限制，并计算费用。

- \*\*归还管理模块\*\*：处理video的归还操作，更新借阅记录，计算费用，处理超期归还的额外费用。

\*\*3. 系统交互流程\*\*：

- 当用户进行搜索时，通过搜索引擎接口将搜索请求发送到服务器，服务器处理请求后返回搜索结果。

- 当会员进行借阅时，系统验证会员资格和借阅数量限制，然后创建借阅记录，并计算费用。

- 每天系统定时运行任务，检测新片并将过期的“新片”重新归类。

- 当会员资格即将过期时，系统通过邮件或短信通知会员进行续费。

\*\*4. 技术选型\*\*：

- 使用Web技术（如HTML/CSS/JavaScript）构建前端界面。

- 后端可以使用Java、Python等语言进行开发，配合数据库技术（如MySQL、MongoDB等）存储数据。

- 网络通信可以采用HTTP协议或RESTful API进行客户端和服务器之间的通信。

- 对于事件驱动的部分，可以使用消息队列（如RabbitMQ、Kafka）进行异步处理。

\*\*5. 安全性考虑\*\*：

- 保证数据传输的安全性，采用HTTPS协议进行通信。

- 对用户输入进行验证和过滤，防止SQL注入等安全漏洞。

- 对敏感数据（如会员信息、支付信息等）进行加密存储。

\*\*6. 性能优化\*\*：

- 对于搜索功能，可以使用全文搜索引擎（如Elasticsearch）提高搜索效率。

- 对于大量数据的处理，可以使用缓存技术（如Redis）减轻数据库压力。

- 对于定时任务，可以使用任务调度框架（如Quartz）进行管理和优化。

通过上述设计，我们可以构建一个功能完善、性能良好且安全的音像店网络系统，满足给定的需求描述。

​