



分类号_____

密级_____

UDC _____

学号_____

太原理工大学

毕业设计（论文）

基于 SSM 框架的在线音乐系统的设计

论文题目 与实现

Thesis Topic Design and Implementation of Online Music System Based
on SSM Framework

学 生 姓 名	武天娇
学号	2013006228
所在院系	软件工程学院
专业班级	软件 1329
导师姓名职称	王爱莲讲师
完 成 日 期	2017 年 5 月 10 日

2017 年 5 月 30 日

太 原 理 工 大 学

毕 业 设 计（论 文）任 务 书

第 1 页

毕业设计（论文）题目：

基于 SSM 框架的在线音乐系统的设计与实现

毕业设计（论文）要求及原始数据（资料）：

1. 综述在线音乐系统的现状；
2. 实现基于 SSM 框架的在线音乐系统管理所需的功能；
3. 按照典型软件工程的流程、规范和方法，完成音乐管理模块、登录注册模块、歌曲评论模块的分析、设计、实现、测试和部署；
4. 针对特定的技术或算法，给出较为深入的研究分析结果；
5. 训练检索文献资料和利用文献资料的能力；
6. 训练撰写技术文档与学位论文的能力。

毕业设计（论文）主要内容：

1. 研究和分析在线音乐系统管理模块的功能和非功能需求并形成需求说明书。

在线音乐播放系统：控制音乐播放；收藏到本地音乐；个人中心等模块。

2. 设计系统的原型并形成原型设计文档；
3. 设计系统的体系架构、数据库、开发框架、关键算法等并形成系统设计文档。
4. 模块代码开发及单元测试并形成最终系统实现；
5. 系统集成、功能测试及形成测试报告；
6. 编写系统安装使用文档；
7. 在该系统基础上，加入一定的技术研究型内容，并最终形成毕业论文。

学生应交出的设计文件（论文）：

1. 内容完整、层次清晰、叙述流畅、排版规范的毕业设计论文；
2. 包括毕业设计论文、源程序等内容在内的毕业设计电子文档及其它相关材料。

主要参考文献（资料）：

[1]王艳, 清陈红, 基于 SSM 框架的智能 web 系统研发设计[J]. 计算机工程与设计, 2013(12):4751

[2]陈夫真, 基于 SSM 的某高校教室管理信息系统的设计与实现[J]. 苏州大学 2012(05):83

[3]张宇, 王映辉, 张翔南, 基于 Spring 的 MVC 框架设计与实现[J]. 计算机工程. 2010(04)

[4]杨群, 基于 SSM 的高校排课系统的研究与应用[J]. 苏州大学, 2013(05): 102

[5]陈君, 黄朝兵, 在线音乐网站的设计与开发[J]. 现代计算机(专业版), 2012(05):68

[6]孟伟, 在线音乐市场的现状分析[J]. 企业经济. 2005(08):115

[7]姜静, 孙立权, 在线音乐网站系统的设计与实现, 周口师范学院 计算机科学与技术学院, 2013(08)

[8]Clement Poncelet, Iorent, Jacquemard, Model-based testing for building reliable realtime interactive music systems, Science of Computer Programming, 2016

[9]Anonymous, Live Music Guide Launches Website Providing Album and Concert Reviews and Commentaries, Wireless News, 2010

[10]Kumar U A, Deepashree S R, Personal music systems and hearing, The Journal of laryngology and otology, 2016

专业班级 _____ 软件 1329 班 _____ 学生 _____ 武天娇

要求设计（论文）工作起止日期 _____ 2017 年 3 月 13 日~2017 年 6 月 25 日

指导教师签字 _____ 古雪 _____ 日期 _____ 2017 年 6 月 25 日

教研室主任审查签字 _____ 日期 _____

系主任批准签字 _____ 日期 _____

目录

1 绪论	9
1.1 开发背景和意义	9
1.2 开发环境	11
1.2.1 开发语言	11
1.2.2 开发工具	12
1.2.3 开发方式	12
1.3 可行性分析	17
1.3.1 技术可行性分析	17
1.3.2 操作可行性分析	17
1.3.3 经济可行性分析	18
2 需求分析	19
2.1 用例分析	19
2.2 功能需求分析	20
2.2.1 术语	20
2.2.2 目标与范围	21
2.3 非功能需求分析	21
3 总体设计	22
3.1 系统框架设计	22
3.2 数据库设计	23
3.3 时序图设计	25
4 详细设计	28
4.1 系统流程图	28
4.2 项目结构设计	29
4.2.1 后台结构设计	29
4.2.2 资源文件设计	30
4.2.3 前台 JSP 页面设计	30
4.3 系统主要功能及代码实现	30
4.3.1 用户模块	30
4.3.2 管理员模块	37
4.4 网页交互程序	39

5 系统测试.....	40
5.1 软件测试概述.....	40
5.2 测试用例.....	41
5.2.1 用户模块.....	41
5.2.2 管理员模块.....	42
结论.....	43
参考文献.....	44
致谢.....	45
外文原文.....	46

基于 SSM 框架的在线音乐系统的设计与实现

摘要

随着计算机网络在现实生活中的慢慢深入和渗透,在网络上出现越来越多的娱乐方式,而音乐占据了很大一部分。音乐在生活中是必不可少的,可以放松身心,可以释放情感,给人们的日常生活带来了极大的乐趣,让人们在繁忙疲惫的工作之余可以进行休闲享受。

通过对在线音乐网站的调查与分析,从现实生活中的用户角度理解需求,进一步明确了在线音乐系统所要实现的功能,从而运用 Spring+SpringMVC+MyBatis 框架(基于 Java 语言),MySQL 数据库和 JQuery 实现了一个简单的音乐系统的设计与应用。游客用户只能浏览和试听音乐,可以通过登录注册,创建自己的歌单,收藏喜欢的歌曲,对每一首歌曲都可以发表自己的评论,下载喜欢的歌曲等;后台管理中,管理员通过登录系统,可对曲库、歌手、评论、歌单进行管理和维护,比如向曲库中增加最近的流行歌曲,可以删除不当的评论等操作。

关键词: SSM 框架; 音乐系统; 后台管理; MySQL 数据库

Design and Implementation of Online Music System

Based on SSM Framework

Abstract

As the computer network penetrate slowly in real life, more and more entertainments appear on the Internet, and music occupies a large part. Music is essential in life, you can relax, you can release the emotions, bring great fun to people's daily life, so that people can enjoy leisure in the busy tired of worktime.

Through the investigation and analysis of online music website, from the user point of view in the real life, the functions of the online music system are further determined using the Spring + SpringMVC + MyBatis framework based on the Java language, MySQL database and JQuery language to achieve a simple music system's design and realization. Visitors can only browse and listen to music, they can create their songlists, collect their favorite songs through logging and registering, publish their own comments on each song and download their favorite songs and so on. In the background management, the administrator login the system, he can manage and maintain music library, singer, comment, song list, such as adding the most recent pop songs to the music library and he can delete the inappropriate comments and other operations.

Key words: Spring+SpringMVC+Mybatisframework; music system; background management; MySQL database

1 绪论

1.1 开发背景和意义

不知不觉中，网络早已深入人心。我们身边的同学、同事、父母、朋友，似乎一个个都无法摆脱互联网的使用和影响。信息化的时代带给了我们很多的便利。渐渐地，音乐网站出现了，人们通过音乐网站可以及时地听到流行的歌曲，放松身心，陶冶情操，释放压力。

正因为宽带的推广，带动了音乐网站产业的蓬勃发展，音乐变得无处不在。近年来，全球数字娱乐产业持续发展，音乐产业形势一片大好。音乐网站改变了以往传统音乐产业的产业结构，用户可以在网站上轻轻一点，就可以听到不同国度、不同风格的音乐，简便的操作，流畅的音质让用户享受更完美的体验。

网络时代的我们，非常善于通过 Internet 搜索和寻找自己想要的资源，例如音乐、影视、图书、文档等。这是一个可以让我们“不出家门便可知天下事”的信息数字时代。而本系统为《音乐网站建设》，我们的主要目标人群是喜爱音乐的用户。研究表明，普通用户手动搜索想要的资源次数不能超过 3 次，否则会让人产生一种无所适从的感觉，在降低对资源渴求的同时也会减少对网络资源的信任。就目前而言，专注做音乐的公司市面上有很多家，总体说来可以大概有如下几家公司的发展前景是相对看好的：虾米音乐、QQ 音乐、网易云音乐、酷狗音乐、Apple Music 等。（音乐软件公司目前调研结果仅在中国区有效）

但这些音乐软件或多或少都有属于自己不痛不痒但是却如鲠在喉的不适应。如 QQ 音乐的界面交互非常反人类，对想要音乐社交、查找搜索音乐信息的新手并不是十分的友好；虾米音乐（阿里巴巴）虽然曲库资源和界面交互很好，但是其痛点在于界面的字体相较于其他软件并不是过于友好；网易云音乐虽然现在势头猛涨，看似如日中天，但其最大的问题在于曲库收录的歌曲方向把握逊色一些，它的曲库在于日英方面极强，但是华语歌坛，具有正式版权的歌曲太少。从长远角度看并不利于自身的下一步发展。

上述的三种音乐网站各有特色。但是本系统更希望能够写出一款采用网状的音乐网站来实现以下功能：

- 1.实现在线实时试听；
- 2.用户简便的操作就可以收藏下载歌曲；

- 3.采用层次状或网状的架构，支持后台管理系统功能；
- 4.能够记录用户的喜好、收藏和推荐。

随着互联网的通信技术、多媒体技术、云端存储和大数据技术的指数膨胀，音乐网站作为一款面向喜欢音乐的用户在线网络服务，给用户和对音乐有需求的网友提供了极大的便利，在让用户多一份选择的同时，也提高我们自身的知名度。但有一些音乐网站由于各种不同的原因遭到了社会的淘汰，起原因是多个方面造成的影响。因此，为了用户能够更加迅捷的查找想要的资源，更加迅速的找到音乐，且可以播放和收藏，将音乐加入到自己的音乐盒中、点歌并将自己的建议写入到留言板以便网站的完善等，需要建立一个自由、安全的音乐网站。

流行音乐在网络时代传播迅速，大街小巷，一夜成名，从前音乐的市场是靠单一的唱片购买，而如今，互联网新兴起来，加剧了音乐的传播时效和传播范围，正因为有了音乐网站的诞生，音乐的市场被打开了。音乐网站在设计、建站、创办等方面有了前所未有的突破和改变，比如，网络技术变成音乐市场的载体，使得人们接触到最新的音乐；因为有了音乐网站，使得一些音乐人的创作变得更加简便，通过网络上传到音乐网站，大家就可以听到新鲜出炉的音乐；音乐网站的出现改变了以往人们欣赏音乐的方式，以往人们通过演奏会、电视转播、电子数码产品载体等，现在只需轻轻一点，操作简单，可以欣赏到最新的流行音乐。

随着社会发展，各国的交流非常频繁，人们所熟悉的音乐和音乐人不再单单局限于国内，还有很多来自国外，中国的音乐也随着国家间的交流被传播得更远。在我们经常看到的娱乐节目中，经常可以看到外国人演唱中国的歌曲。音乐不仅仅用于陶冶情操和休闲娱乐，他还随着社会的发展，用于文化交流。所以各国的音乐不进行明确的分类和管理是不可行的。所以，就要有一个系统来管理和归纳分类这些多样的歌曲。音乐网站系统就是为了更好的来管理音乐，将音乐分类。方便大家查找音乐，更好的了解不同国家的音乐风格、特色。

在如今高压的工作学习生活中，许多人们通过欣赏音乐来放松、释放自己的心情，听不同类别的音乐，在不同的情境下，听者会有不同的体会，也许会释然开朗，也许会感动落泪，也许会开心大笑，音乐也是治疗现代人心理压力的一种方法，音乐网站的出现适时的填补了这份空白。

本音乐网站基于 java 语言应用 Spring + SpringMVC + MyBatis 框架模式进行设计开发。在经过大量文献资料的分析整理后，抽象出普通用户对于音乐网站的基本要求和基本使用方式，也渐渐明确了项目最终需要的功能和用户交互方式。

1.2 开发环境

1.2.1 开发语言

(1)本系统采用的是 Java 语言编写的。Java 语言与 C 语言最大的区别是面向对象编程与面向过程编程。好比如我们制作一个象棋类的小游戏。如果使用 Java 语言的话，我们首先制定好下棋的规则，棋盘，棋子等由对象构成的类，再去书写主程序；而若使用 C 语言，我们需要按部就班的，按照下棋的模仿思路完成代码。相比较于面向过程，Java 的面向对象更加直观，适合新手学习。

Java 的第二个特点在于其强壮的可移植性。由于 Java 虚拟机 JVM 的存在，使得 Java 无论在任何平台（Windows，Linux，OS X 等）都具有跨平台运行的能力。Java 程序运行不依赖操作系统的底层，而是依赖于 JVM 虚拟机的运行机制。而 JVM 虚拟机在执行 Java 代码时，会将字节码自动编译成为当前操作系统可以识别的机器语言执行。所以 Java 的另一个绰号是“一次编译，到处运行”。

Java 语言比较于 C 语言而言比较简单，具有跨平台、安全性以及面向对象等特点。跨平台指的是 Java 能运行于不同的平台，引进了虚拟机原理，实现了不同平台的接口并且在虚拟机上运行。

(2).Ajax 和 JQuery 技术

Ajax 是基于 XML 的异步 JavaScript，它最终的效果是在向 web 服务器发送请求时能够局部刷新当前页面。我们在日常使用时常常会遇到这样的情况，比如在线生成即使的表单，或者查看音频视频等，这些好多都用上了 Ajax 的技术。而与 Ajax 相互适应的是另外一种传输格式，Json。不同于传统的数组或是 list，Json 返回的内容经过解析后可直接反映在页面上。

Json 与传统的 xml 相比较，由其优势也有不足。优点在于，首先，Json 格式的消息更加便于人的阅读和机器的写入。Json 具有分割数据的字符，而这种字符又恰恰是 JavaScript 的引擎在数据结构底层解析时的表达方式，所以可以提高效率。

综上所述，可以发现 Ajax 应用程序具有如下优势：

- 1.通过异步刷新，减少用户等待时间，很好的提升了用户的体验度；

- 2.Ajax 引擎在客户端运行，替服务器分担了一些压力和负载，在减少服务器的负担同时也提高了访问的速度；

3. Ajax 利用 Json 格式的特点，优化了服务器和浏览器之间的传输时不必要的数据传递，减少了带宽占用。在有限的容量内可以访问到更多的资源。

Jquery 是基于 JavaScript 开发的库，使用起来相较于 JS 更加的简洁和方便，阅读时也更加直观。它具有以下功能：

1.链式操作方式(对发生在同一个 Jquery 对象上的一组动作，可以直接连接写而无需重复获取对象。);

2.许多成熟的插件可供选择;

3.可靠的事件处理机制;

4.实现动画效果更方便，并且为网站提供 Ajax 交互。

Jquery 已经集成了 js、CSS、DOM 和 Ajax 于一体的强大功能，可以利用很少的代码，完成更多复杂的功能。Jquery 作为封装的库，其目的在于简化开发者使用 js。

1.2.2 开发工具

本项目使用的开发工具和软件包括有：eclipse, HBuilder, Adobe Photoshop CC 2015, Navicat Premium, PowerDesigner, MySql 等。

Eclipse: Java 软件开发环境 (IDE)，主程序运行环境;

HBuilder: HTML 开发环境，建构前台页面;

Adobe Photoshop: 页面所需图片修改及加工;

Navicat Premium: 数据库可视化管理软件，主要用于连接数据库;

PowerDesigner: 数据库设计软件，用于构思和生成数据库中的表;

MySql: 存储数据;

1.2.3 开发方式

本系统是基于 Java 语言，使用 Spring + SpringMVC + MyBatis (SSM) 框架搭建开发的。

1.Spring 框架

谈及 Spring 框架，主要有这么几个方面：控制反转 IoC，面向切面编程 AOP，依赖注入 DI。

IoC，控制翻转。之前使用的简单的 JSP 也好，或是 MVC 的分层模式也好。

均是由自己手动配置加载类和子类。不仅繁琐而且极易出错。而用 Spring 将代码中的配置文件转移到了 xml 之中, 并且将生成类和对象的控制方由程序员交给了 Spring 容器去管理, 而现在只需要在 xml 文件中配置好相对应的类即可。

使用 IoC 最大的好处, 则是原先需要修改代码的部分, 现在仅仅需要修改配置文件即可。方便程序员开发维护的同时, 也提高了项目的容错率, 降低风险。使用 IoC 的缺点也是一目了然, 首先是配置类的过程, 相较于原来的直观, 现在变得略微复杂; 再者是用到框架转化后, 会对性能有轻微的损耗。若在开发对性能要求极高的项目是, 建议自己搭建最适合自己的底层框架。

第二个要说明的是 DI, 中文解释是依赖注入。这个概念与之前所用的完全不同, 是基于 IoC 而衍生的全新思路。在项目中经常使用注解 `@Autowired`、`@Inject` 以及 `@Resource` 来完成控制反转的类实现。其中 `@Autowired`、`@Inject` 较为相似, 处理依赖注入的方式均是通过 `AutowiredAnnotationBeanPostProcessor`, 而 `@Resource` 是通过 `CommonAnnotationBeanPostProcessor` 来完成依赖注入。

`@Autowired` 默认按照类型 (type) 装配。当装配对象为控制时, 设置 `@Autowired(required=false)` 即可。

`@Resource` 不同于 `@Autowired`, 是通过 name 属性完成装配。没有指定 name 属性, 且注解写在字段上时, 默认取字段名进行按照名称查找。

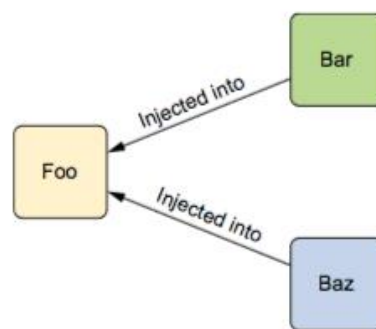


Figure 1.1 Dependency injection involves giving an object its dependencies as opposed to an object having to acquire those dependencies on its own.

图 1.1 依赖注入图解

第三个要解释的是 AOP, 直译为面向切面编程。曾经在课堂学过 C 语言的面向过程编程, Java 语言的面向对象编程。而在使用中, 这两种思想无法解决这样的问题, 记录日志信息以及搭建拦截器过滤器等, 此时需要用到面向切面编程的思想 (AOP)。其目的是在编写代码中, 允许分离交叉的方式来增加模块

化。它在不改变现有代码的基础上，为现有代码增添新的功能。这可以将非业务核心的服务（如日志信息等）添加致我们项目的功能之中，来避免代码的杂乱无章。

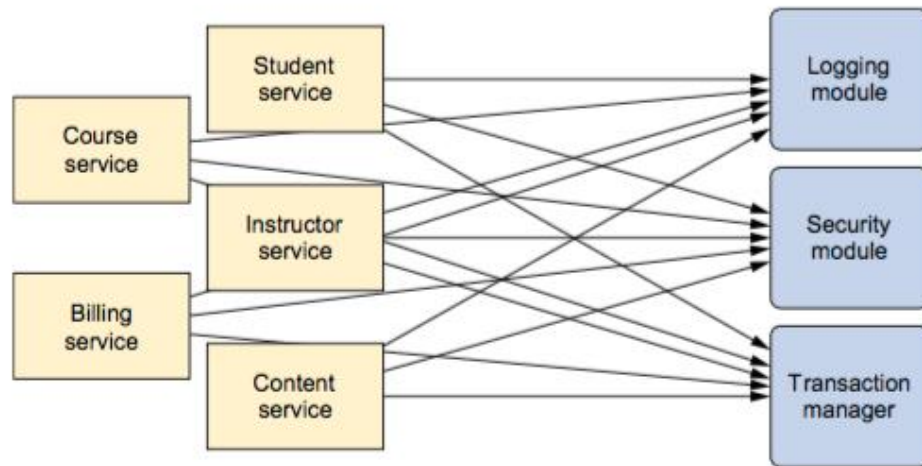


图 1.2 AOP 日志分析图

简而言之，Spring 有以下优点：

- 1.简化 Java 开发过程；
- 2.更少的代码量完成更多的工作量；
- 3.增强内聚，减少耦合；
- 4.配置文件便于维护和更新

2.SpringMVC 框架

Spring MVC 和 SSH 框架中的 Struts2,都属于 MVC 中 View 视觉层的框架。SpringMVC 是 Spring Freamwork 的一部分。

SpringMVC 工作流程是的核心架构运行思路逻辑如下表示：

1、用户在客户端（浏览器）发起请求，请求被送到前端控制器 DispatcherServlet。DispatcherServlet 在收到请求后转发给其他其他处理器。DispatcherServlet 本身只用于调控全局，是的所有的请求入口统一，但不负责各个请求的具体处理；

2、DispatcherServlet 到 HandlerMapping，这个组件会把请求映射为 HandlerExecutionChain 对象（包含一个 Handler 处理器（页面控制器）对象、多个 HandlerInterceptor 拦截器）对象，通过这种策略模式，很容易添加新的映射策略；

3、DispatcherServlet 到 HandlerAdapter，这个组件会把处理的组件封装后，仅提供接口，成为类似适配器的组件。所以 HandlerAdapter 支持多种类型的处理器组件，也就是我们所应用的的适配器设计模式，在支持更多类型的处理组件是更加方便；

4、HandlerAdapte 是处理器功能处理方法的调用。这个组件将会根据适配的结果返回，调用真正的处理器已经封装好的的功能处理方法，然后完成功能处理；最后会返回一个 ModelAndView 对象（包含逻辑视图名、模型数据等）；

5、ModelAndView 的逻辑视图名到 ViewResolver。ViewResolver 将把逻辑视图名称，分析为具体地通过查看过这种策略示意，很容易添加或者更换其他视图技术；

6、View 到渲染视图层，View 会根据传进来的 Model 模型数据进行渲染。此处的 Model 是指我们在 controller 用到的 Model 类，它实际是一个 Map 数据结构，便于其他视图技术的使用和支持；

7、返回控制权给 DispatcherServlet，由 DispatcherServlet 返回响应给用户，到此一个流程结束。

Spring MVC 部分核心组件功能如下：

1、DispatcherServlet：前端控制器。DispatcherServlet 是整个控制流程的中心以及核心，用来专门处理用户发送的请求。用 MVC 模式中的介绍，属于 Control 层。通过 DispatcherServlet 调用其他组件的使用，使得各个组件之间的耦合度降低。

2、HandlerMapping：映射器。HandlerMapping 主要负责通过用户的请求寻找到正确的处理器（及 Handler）。SpringMVC 提供了多种映射方式。如常见的有配置文件方式（..properties 或.xml）、注解（@Resource 或@Autowired）等等。

3、Handler：处理器。程序的两端之后要被处理在自己的请求的 DispatcherServlet 的 DispatcherServlet 的控制处理的控制的前一个步骤。它是与核心有关的服务，通常需要开发者处理程序，根据业务需要特定用户的需求。

4、HandlAdapter：处理器适配器。通过用 HandlerAdapter，可以由处理器的执行应用适配器模式中，运行更多类型的处理器来扩展适配器。

5、ViewResolver：视图解析器。负责处理结果生成 View 视图，ViewResolver 优先逻辑图，说明翻译物理视图名称，即具体网页地址生成 View 视图以及 View

对象，最后的渲染处理结果显示用户页。springmvc 支持多 View 视图类型：如 jstlView, freemarkerView, pdfView 等。

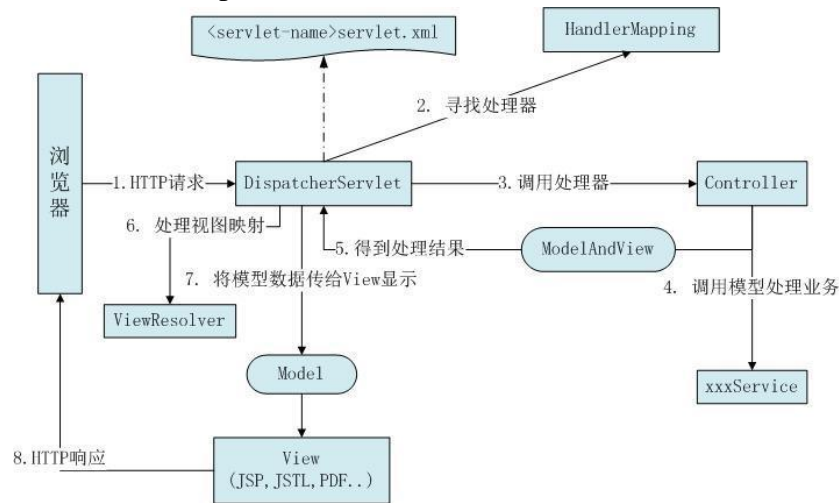


图 1.3 SpringMVC 原理流程图

3.Mybatis 框架

以往的数据库连接方式 JDBC 进行数据查询的时候，往往需要繁琐的 7 个步骤：

1. 加载 JDBC 驱动
2. 建立并获取 JDBC 连接
3. 创建 JDBC statements 对象
4. 设置 SQL 语句的参数
5. 执行 SQL 语句
6. 将查询的结果进行转换和处理，然后返回
7. 关闭 JDBC 连接

这种连接数据库的方式对于项目开发，许多代码是冗余的，于是，mybatis 框架出现了，它对数据库连接的过程进行优化，比如：

1. 使用数据库连接池来减少资源浪费的情况
2. 使用 DataSource 进行解耦
3. SQL 语句统一放在配置文件中，优化代码性能
4. 使用了动态 SQL 语句，可以根据自己的请求去拼凑相应的 SQL 语句
5. 优化了查询结果集
6. 重复代码进行抽离，方便复用。

1.3 可行性分析

可行性分析是为了用最小的代价去确定该系统能否在规定时间内解决。要实现这个目标，就要从 3 种主要的可能性进行分析利弊，来进一步进行判断该系统的需求和规模是否是可实现的，系统的实现是否需要巨大的经济成本和它所带来的经济效益不成正比。可行性分析的任务不是具体研究问题的解决方案，而是研究问题的范围，分析是否有可行的解决方法。

本系统的可行性分析从三方面进行分析，主要包括技术上的可行性，操作上的可行性，经济上的可行性。

1.3.1 技术可行性分析

本系统的技术设计主要分为后台和前台，后台主要基于 Java 语言，运用 Spring 框架、SpringMVC 框架与前台交换数据、Mybatis 框架与数据库进行连接。前台主要基于 HTML5、CSS、JavaScript 语言，运用 Bootstrap 框架。

表 1.1 运用技术分析表

后台技术		前端技术	
整体框架	Spring	基础语言	HTML、CSS、JS
数据库交互	MyBatis	集成框架	Bootsharp
页面交互	SpringMVC		JQuery
数据库	MySql		

就目前所使用到的技术，均是已有成熟的技术之上开发完成的。技术上可以完成预期目标。

1.3.2 操作可行性分析

服务器平台要求：

支持 Intel、AMD 等平台，CPU1.5GHz 以上；

内存 2G 以上；

硬盘 1T 以上（固态硬盘优先）；

数据库：MySQL；

客户端平台要求：

win7 及以上（推荐 win10）或 Liunx 等；

多种浏览器支持；

支持 JavaScript 及 HTML5；

1.3.3 经济可行性分析

本系统经济需求较低，具备成熟的软硬件环境，并且系统不是很复杂，开发周期较短，经济支出有限。通过与传统音乐产业相比，现代音乐网站所需的经济条件较少，需要一个电脑或者一部手机就可以使用，而传统的音乐市场则需要购买实物性的磁带、CD 等，制作成本较高，人员支出较多，所以从长远利益来看，本系统的开发是具有经济可行性的。

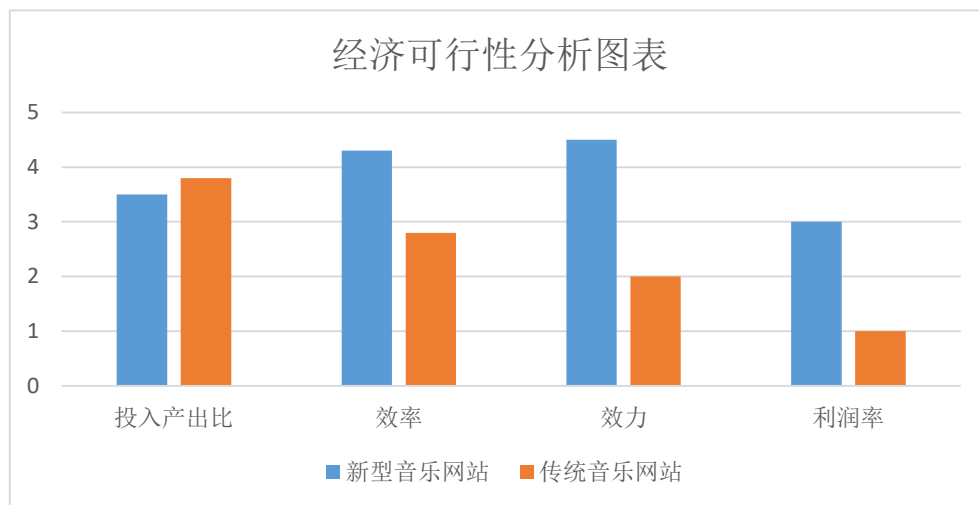


图 1.4 经济可行性分析图

2 需求分析

2.1 用例分析

1. 用户用例

- (1) 用户输入用户名、密码
- (2) 用户收藏音乐到用户歌单
- (3) 用户创建、删除用户歌单
- (4) 用户发表评论
- (5) 用户查看系统歌单内歌曲
- (6) 用户查看歌手的全部歌曲

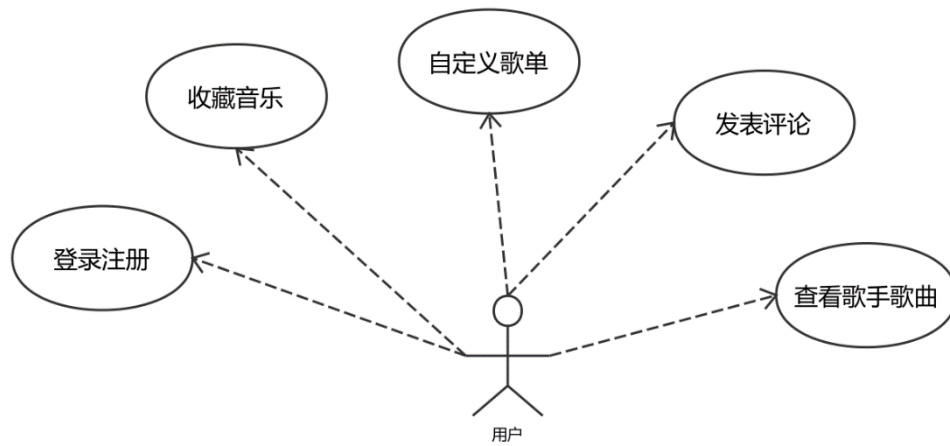


图 2.1 用户用例图

2. 管理员用例

- (1) 管理员登录音乐后台管理系统
- (2) 管理员管理曲库
- (3) 管理员管理歌手
- (4) 管理员管理系统歌单
- (5) 管理员管理歌曲评论

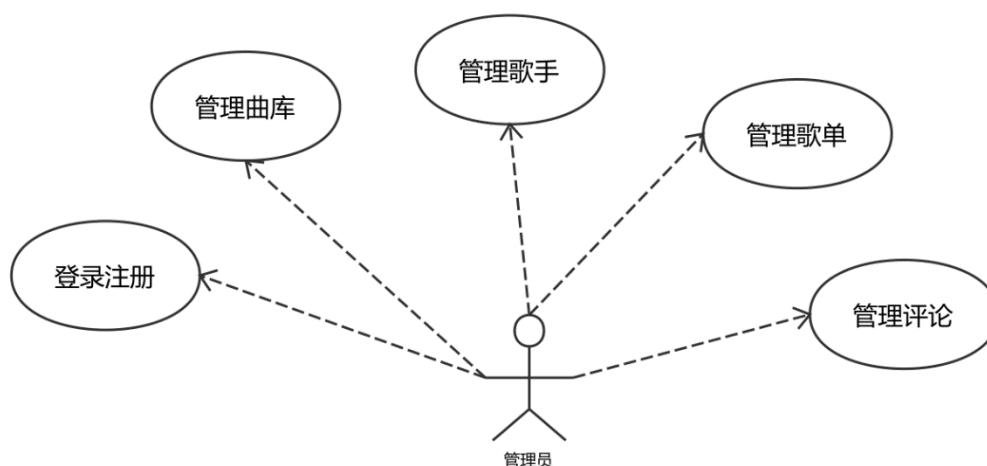


图 2.2 管理员用例图

2.2 功能需求分析

2.2.1 术语

一、用户

用户是音乐系统的直接使用者，通过用户名、密码登录到系统，用户可以在系统中试听、下载音乐、抒发自己对音乐的见解与评论，可以创建、修改、删除自己的歌单，查看所有音乐、收藏音乐到自己的歌单中，查看所有歌手、收藏自己喜欢的歌手，查看所有歌单中的歌曲，同时还有可以修改用户的个人信息。

二、歌手

歌手由管理员进行添加、修改、删除，提供给用户查看、收藏。

三、管理员

管理员是音乐系统后台管理的用户。管理员可以对用户进行删除，可以对音乐进行更新、删除、上传，可以上传、更新、删除歌单，可以对歌手进行更新、删除、可以将不当的评论进行删除，还可以修改自己的个人信息。

四、歌曲

歌曲是用户使用音乐系统的目的。用户可以试听、下载、收藏音乐。

五、歌单

歌单是由后台管理员上传，可更新、删除，提供给用户收藏、下载歌曲。

六、用户自定义歌单

用户不仅可以收藏、试听音乐系统自带的歌单，用户可以自己创建歌单，可以将自己喜欢的不同类型的歌曲收藏到不同的歌单中，同时可以进行更新、删除等操作。

七、评论

用户听过某首歌曲之后，可以对此歌曲抒发自己的见解和评论，使得音乐系统多了互动，多了生机。管理员可对出现不当的评论进行删除。

2.2.2 目标与范围

本系统主要使用者即目标，是爱听音乐的用户，范围之广，不分性别、国籍、年龄，音乐就是交流的语言。

目标人群：普通使用者（用户），系统管理员

发展适配人群：自由歌手、唱片公司、作词作曲家；音乐发烧者、音乐收集者。

2.3 非功能需求分析

一、易用性

该音乐系统只需要用户轻轻一点就可以听到自己喜欢的歌曲，可以下载歌曲到本地，不在线的时候也可以听，不需要网络随时支持，用户收藏、下载等操作都是在简单的步骤下进行的。适合绝大多数人轻松、简单的使用。

二、兼容性

用户在使用该音乐系统时，使用的浏览器是不确定的，该系统保证了系统对浏览器的兼容性。

三、界面性

该需求主要描述了对产品外观的展望、情绪和风格，该系统设计的界面运用bootstrap 前台框架，设计简单大方，大气简约，给用户提供完美的使用体验。

3 总体设计

3.1 系统框架设计

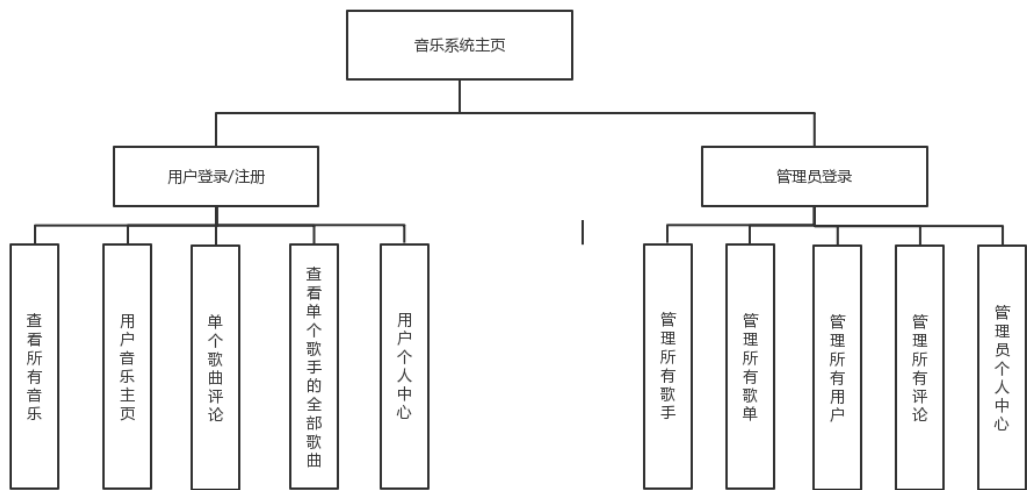


图 3.1 系统整体框架图

本项目分为两个大模块：用户模块和管理员模块。

1. 用户模块：包括用户登录注册模块、用户主页模块、系统主页模块、歌曲评论模块、用户个人中心模块等。
 - (1) 用户登录注册模块：用户只有登录到系统，才能进行其他操作。
 - (2) 用户主页模块：用户可以创建自己的歌单、收藏歌曲到用户歌单内、删除用户歌单内歌曲等操作。
 - (3) 系统主页模块：用户可以查看榜单歌曲、歌手、系统歌单详情。
 - (4) 歌曲评论模块：用户可以对每一首歌发表评论。
 - (5) 用户个人中心模块：用户可以修改自己的个人信息。
2. 管理员模块：包括管理员登录模块、管理曲库模块、管理歌手模块、管理歌单模块、管理评论模块、管理员个人中心模块等。
 - (1) 管理员登录模块：管理员必须经过登录才能进入后台管理系统。
 - (2) 管理曲库模块：管理员可以上传新的歌曲、删除曲库内的歌曲等操作。
 - (3) 管理歌手模块：管理员可以增加歌手信息、删除歌手等操作。
 - (4) 管理歌单模块：管理员可以增加新的系统歌单、删除歌单等操作。
 - (5) 管理评论模块：管理员可以查看全部评论、删除不当的评论。

(6) 管理员个人中心模块：管理员可以修改自己的个人信息。

3.2 数据库设计

音乐网站系统是提供给用户音乐信息，并对音乐信息进行管理的系统，数据库是该系统的核心和基础。并将系统中的信息按照特定的模型组织起来，提供系统可以方便地获取所需信息。同样，数据库设计更是整个系统应用的根基，是软件设计的起点，起着决定性的质变作用。

3.2.1 数据库表设计

根据网站的需求我们创建的数据库主要分为 7 个表：管理员表 admin、评论表 songview、歌曲列表 song、用户表 user、歌手列表 singer、系统歌单表 lists、用户歌单表 userlist。这七个表分别涵盖了整个网站所要用到的数据。下面分别介绍这几个表的结构：

1. 整体结构表

song_list		
id	int(11)	<pk>
songid	int(11)	
listid	int(11)	
listname	varchar(255)	
songname	varchar(255)	
singername	varchar(255)	

user_lists		
id	int(11)	<pk>
userlistid	int(11)	
listname	varchar(255)	
songid	int(11)	
songname	varchar(255)	
singername	varchar(255)	

singer		
id	int(11)	<pk>
singername	varchar(255)	
songname	varchar(255)	
songid	int(11)	
singerid	int(11)	

singersong		
id	int(11)	<pk>
songid	int(11)	
singerid	int(11)	
songname	varchar(255)	
singername	varchar(255)	

song		
id	int(11)	<pk>
songname	varchar(255)	
singername	varchar(255)	
singerid	int(11)	
music	varchar(255)	

song_singer		
id	int(11)	<pk>
songid	int(11)	
singerid	int(11)	
songname	varchar(255)	
singername	varchar(255)	

songview		
id	int(11)	<pk>
content	varchar(255)	
songid	int(11)	
songname	varchar(255)	

userlists		
id	int(11)	<pk>
userlistsname	varchar(255)	

admin		
id	int(11)	<pk>
adminname	varchar(255)	
adminpwd	varchar(255)	

lists		
id	int(11)	<pk>
listname	varchar(255)	
listid	int(11)	

user		
id	int(11)	<pk>
username	varchar(255)	
userpwd	varchar(255)	

图 3.2 数据库整体结构设计表

2.admin 表

admin 表是管理员信息表。主要字段有管理员编号 (id)，管理员名称 (adminname)，管理员登录密码 (adminpwd)。

id	int	11	0	<input checked="" type="checkbox"/>	 1
adminname	varchar	255	0	<input type="checkbox"/>	
adminpwd	varchar	255	0	<input type="checkbox"/>	

图 3.3 admin 表结构

3. songview 表

songview 表是用户评论歌曲信息表。主要字段有评论编号 (id)，评论内容 (content)，评论歌曲编号 (songid)，发表评论用户 (userid)。


id	int	11	0	<input checked="" type="checkbox"/>	 1
content	varchar	255	0	<input type="checkbox"/>	
songid	int	11	0	<input type="checkbox"/>	
userid	int	11	0	<input type="checkbox"/>	

图 3.4 songview 表结构

4. song 表

song 表是全部歌曲表。主要字段有歌曲编号 (id)，歌曲名称 (songname)，歌手编号 (singerid)，歌曲存放地址 (music)


id	int	11	0	<input checked="" type="checkbox"/>	 1
songname	varchar	255	0	<input type="checkbox"/>	
singerid	int	11	0	<input type="checkbox"/>	
music	varchar	255	0	<input type="checkbox"/>	

图 3.5 music 表结构设计

5. user 表

user 表是 user 登录信息表。主要字段有用户编号 (id)，用户名 (username)，用户登录密码 (userpwd)。

id	int	11	0	<input checked="" type="checkbox"/>	 1
username	varchar	255	0	<input type="checkbox"/>	
userpwd	varchar	255	0	<input type="checkbox"/>	

图 3.6 user 表结构设计图

6. singer 表

singer 表是全部歌手表。主要字段有歌手编号 (id)，歌手名称 (singername)。

栏位	索引	外键	触发器	选项	注释	SQL 预览
名					类型	长度 小数点 不是 null
id					int	11 0 <input checked="" type="checkbox"/>  1
singername					varchar	255 0 <input type="checkbox"/>

图 3.7 singer 表结构设计

7. lists 表

lists 是系统歌单信息表。主要字段有系统歌单编号 (id)，系统歌单名称 (listname)。


▶ id	int	11	0	<input checked="" type="checkbox"/>	 1
listname	varchar	255	0	<input type="checkbox"/>	

图 3.8 lists 表结构设计

8. userlist 表

userlist 是用户歌单信息表。主要字段有用户歌单编号 (id)，用户歌单名称 (userlistname)。


▶ id	int	11	0	<input checked="" type="checkbox"/>	 1
userlistsname	varchar	255	0	<input type="checkbox"/>	

图 3.9 userlists 表结构设计

3.3 时序图设计

时序图，是描述一个系统中对象与对象之间有交互行为的时间顺序的动态协作的图。时序图注重对象之间在什么时间顺序的情况下如何进行交互。
根据对系统功能的分析，画出如下时序图。

(1) 用户若有账号，则直接登录，若无账号，则需注册账号进行登录系统。

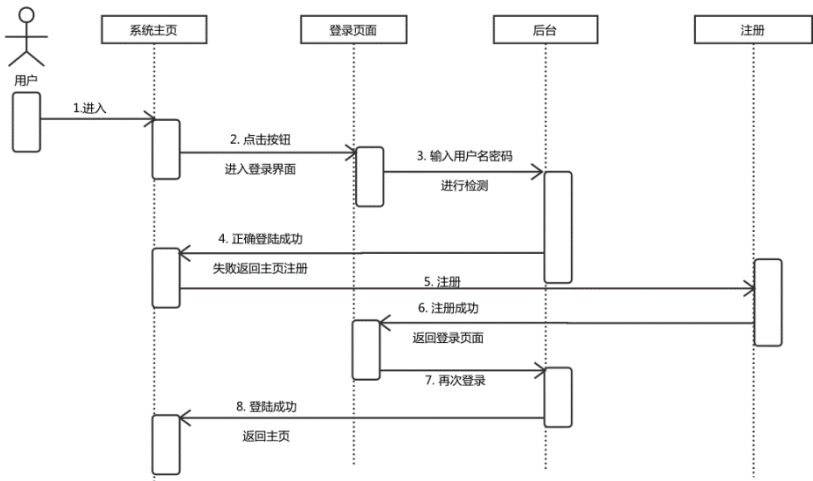


图 3.10 用户登录注册时序图

(2) 管理员可以修改个人的信息。

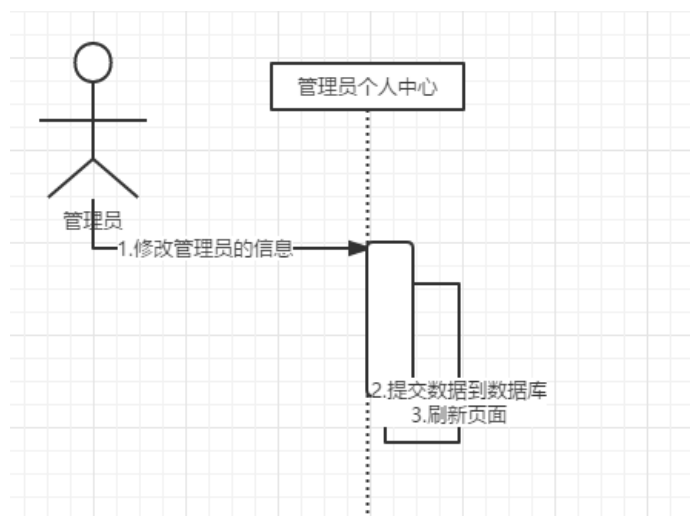


图 3.11 管理员修改个人信息时序图

(3) 用户登录系统进入主页后，可以创建自己的歌单，收藏歌曲到自己的歌单内。

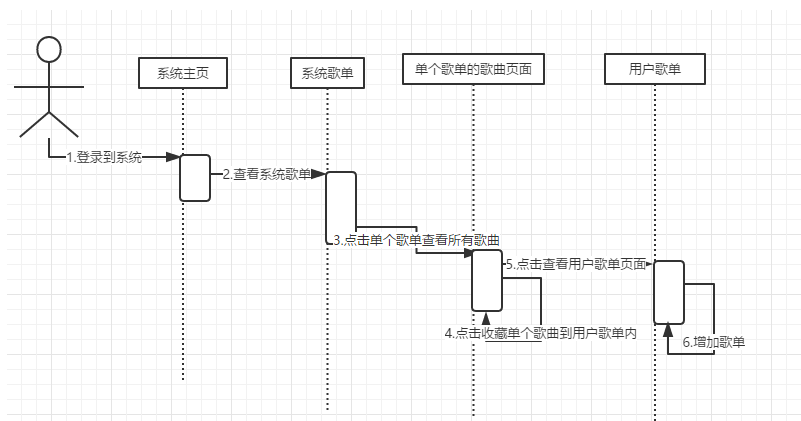


图 3.12 用户收藏歌曲时序图

(4) 用户进入单首歌曲，可以发表自己的评论

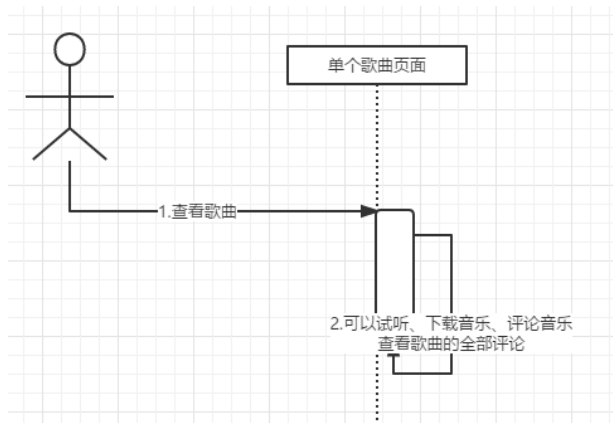


图 3.13 用户评论时序图

(5) 管理员可以增加、删除歌手的信息。

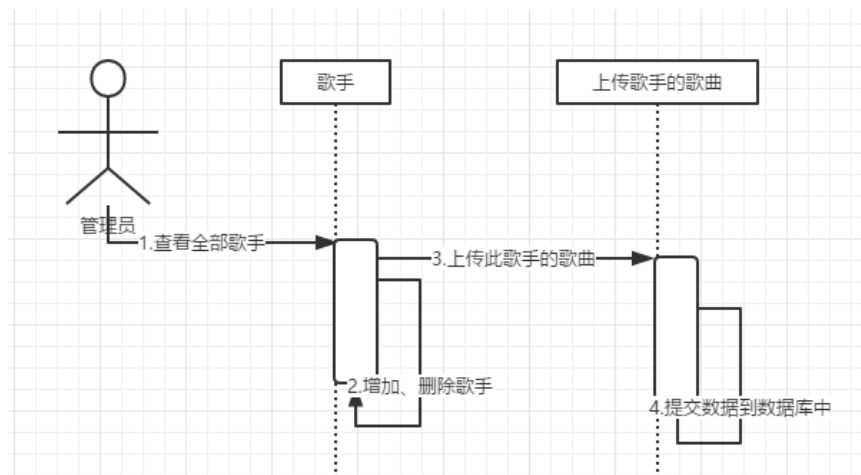


图 3.14 管理员管理歌手时序图

4 详细设计

4.1 系统流程图

音乐网站的流程应顺着系统信息流动的过程逐步地进行，用户采用常用的登陆模式。用户输入用户名和密码，后台会进行验证，正确的话就可以登陆进系统主页面，错误则会提示用户名或者是密码错误。未经注册过的用户要进行注册。用户注册模块包括用户名、密码。用户登录系统后可以创建歌单、试听歌曲、收藏歌曲、评论歌曲、修改个人信息等。

管理员登陆的方式，管理员登陆进去的是后台的界面，可以对用户、歌曲、歌手、评论、歌单的信息进行查看和删除等操作，还可以修改密码。

系统流程图设计：

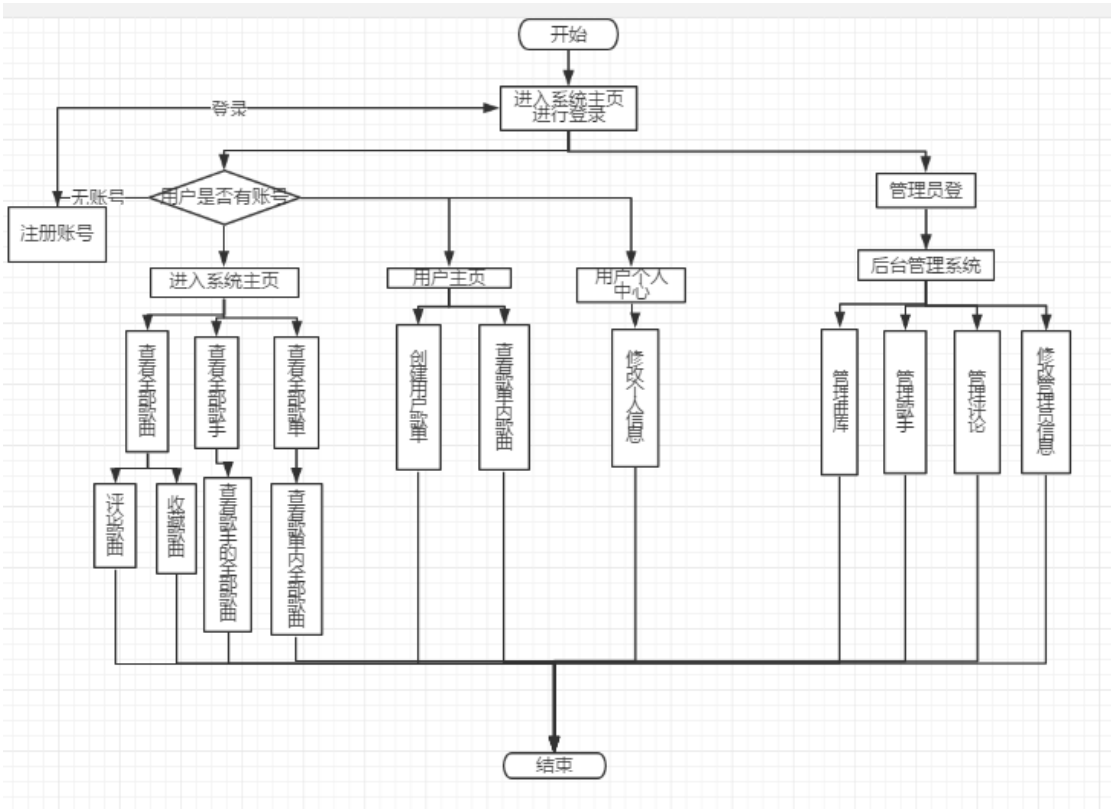


图 4.1 系统流程图设计

4.2 项目结构设计

一个代码整洁、系统架构完整的项目，文件存放和设计是非常重要的。本系统分为处理业务逻辑的后台、框架搭建和整合的资源文件存放、前台 JSP 页面的存放三个模块。

4.2.1 后台结构设计

后台结构重要分为四层：

- (1) entity（实体类）：存放系统中的实体类信息。
- (2) persistence（Mapper-Dao 层）：持久层，封装存放于数据库交互的信息与 SQL 语句。
- (3) service（业务层）：负责处理系统的业务逻辑设计。
- (4) controller（控制层）：调用 service 层及业务接口来控制业务层的流程。

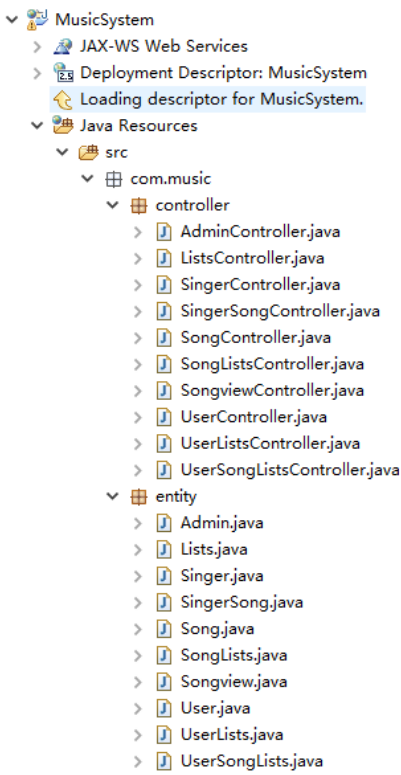


图 4.2 系统后台结构设计图 1

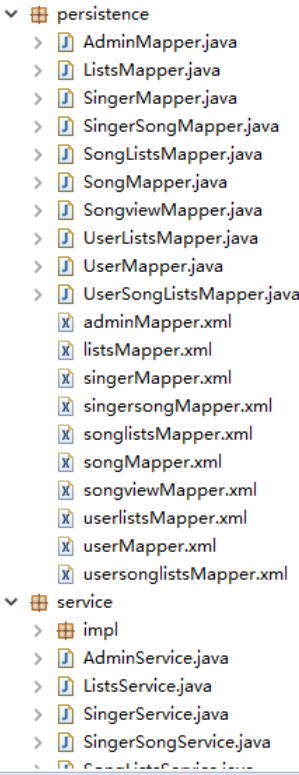


图 4.3 系统后台结构设计图 2

4.2.2 资源文件设计

Spring、SpringMVC、Mybatis 框架的资源文件及它们整合之后的资源文件存放在 src 下的 config 包下。

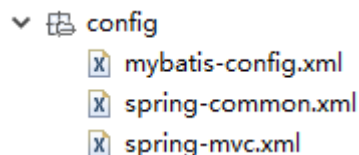


图 4.4 资源文件存放结构图

4.2.3 前台 JSP 页面设计

前台页面设计包括：JSP（view 视图层）页面、前台开发框架的样式文件等。

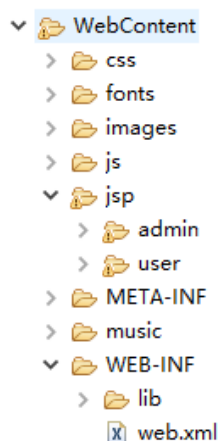


图 4.5 前台页面结构图

4.3 系统主要功能及代码实现

4.3.1 用户模块

(1) 用户登录、注册

用户通过在登录页面输入正确的用户名、密码进行登录。系统后台进行登录验证，将前台输入的用户名、密码与数据库中的数据比较，若相同，登录到主页，进行之后的操作；若不同，则说明该用户没有注册，提示用户返回登录页面进行注册操作，通过在前台输入用户名、密码注册到系统中，然后通过登录，进入系

统主页。

登录截图：

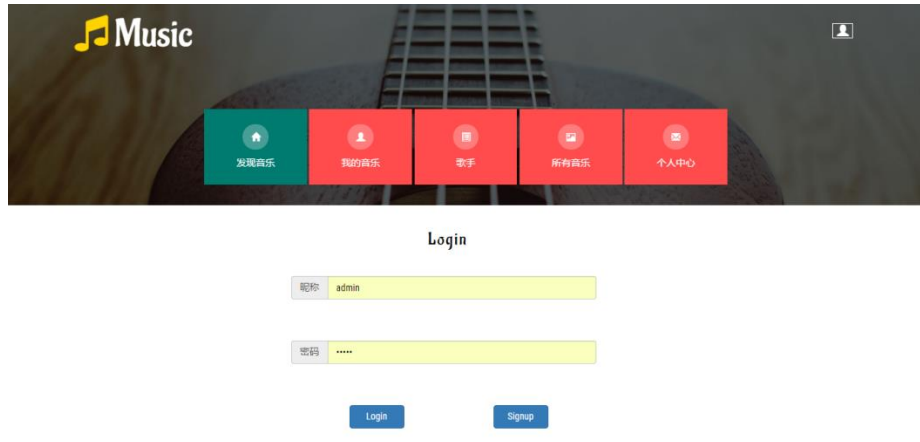


图 4.6 用户登录截图

用户输入表单数据后，后台进行验证：

```
@RequestMapping("/loginUser")
public String loginUser(@RequestParam("username") String username,
                        @RequestParam("userpwd") String userpwd,HttpServletRequest
request) {
    List<User> users = userService.findUserAndPwd(username);
    for(User user : users){
        if(username.equals(user.getUsername())&&
userpwd.equals(user.getUserpwd())){
            //request.setAttribute("userid", user.getId());
            request.getSession().setAttribute("userid", user.getId());
            return "user/index";
        }
    }
    return "user/fail";
}
```

若没此用户名，用户需注册：

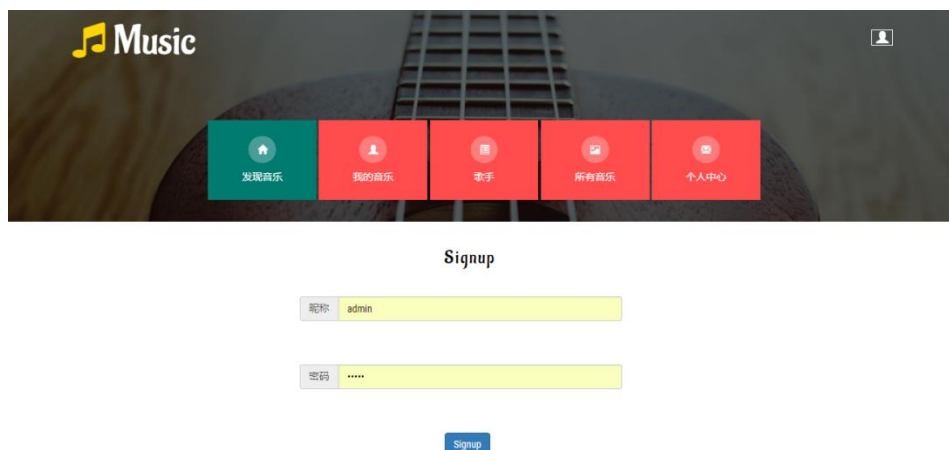


图 4.7 用户注册截图

@RequestMapping("/addUser")

```
public String addUser(@RequestParam("username") String username,
    @RequestParam("userpwd") String userpwd) {
    User user = new User();
    user.setUsername(username);
    user.setUserpwd(userpwd);
    userService.add(user);
    return "user/loginUser";}
```

(2) 用户管理用户歌单模块

用户进行系统主页之后，用户可以查看自己创建的歌单，同时可以创建自己的歌单，收藏歌曲到相应的歌单内。

创建歌单信息截图：

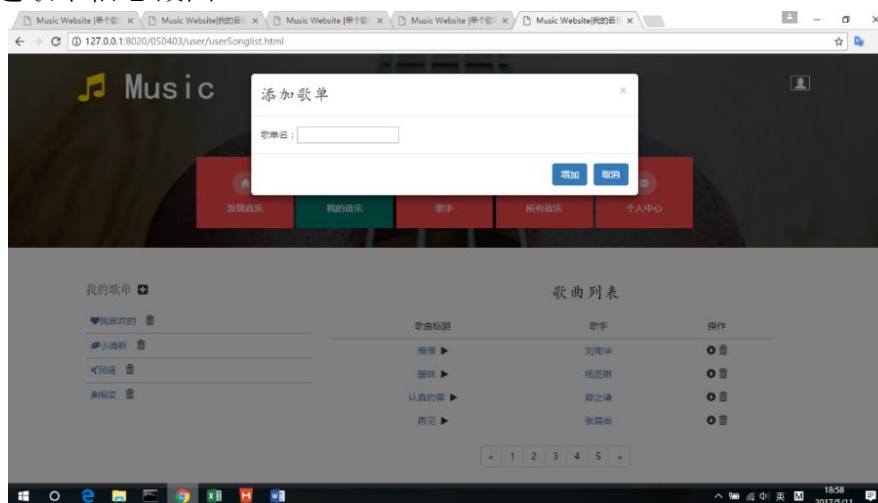


图 4.8 用户创建歌单截图


```
@RequestMapping("/AddUserLists")
```

```
public String AddUserLists(UserLists userlists, HttpServletRequest request)
{
    userlistsService.add(userlists);
    return "redirect:/userlists/getAllUserLists";
}
```

查看用户歌单内的全部歌曲，应用了 ajax 异步请求后台数据

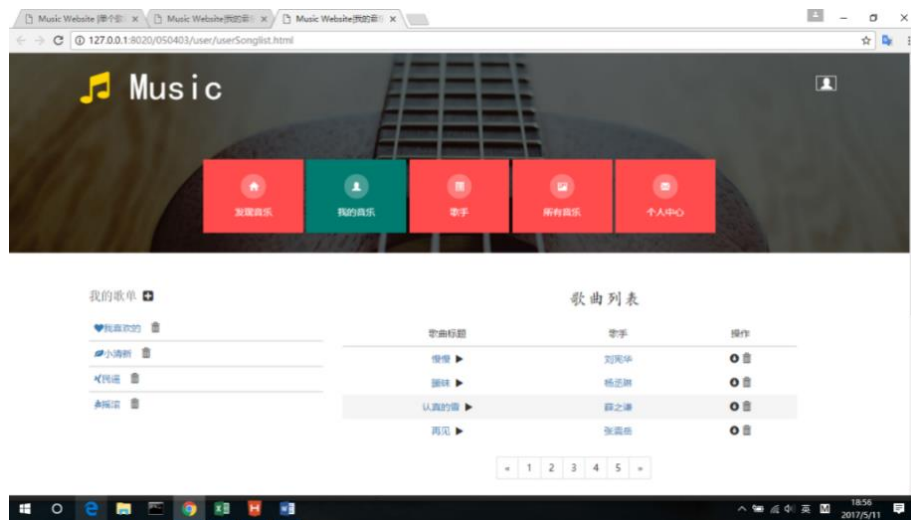


图 4.9 用户查看用户歌单内详情截图

```
$.ajax({  
    type:"POST",  
    url:"<%=basePath%>/userlists/get  
UserLists",  
    data:{  
        "songid":ele,  
        traditional: true,  
        dataType: "json",  
        success:function(data){  
            var temp="";  
            for(var i=0;i<data.length;i++){  
                for(var attr in data[i]){  
                    if(attr === "id"){  
                        temp+="<tr>" +  
                            "<td>" +  
                                "<a  
href='"+''+"<%=basePath%>/song/colle  
ctSong?listid="+data[i][attr]+'"'+">";  
                    }  
                    if(attr === "userlistsname")  
                        temp+=data[i][attr]+"</a>&nbsp; &  
nbsp;</td>" +  
                            "</tr>";
```

(3) 用户查看系统歌单内歌曲模块

用户在系统主页查看到系统歌单，同时点击单个歌单，进入查看相应的歌曲。

```
@RequestMapping("/getSongList")
```

```
public String getSongList(HttpServletRequest request) {  
    List<SongLists> findLists=songListService.findAll();  
    request.setAttribute("findLists", findLists);  
    return "/user/singleLists";  
}
```

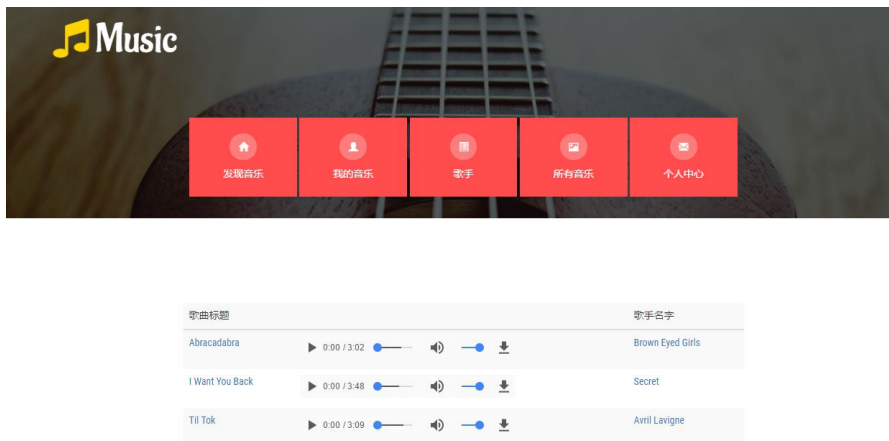


图 4.10 用户查看系统歌单内歌曲详情截图

(4) 用户个人中心模块

用户在个人中心模块进行修改自己昵称、密码。

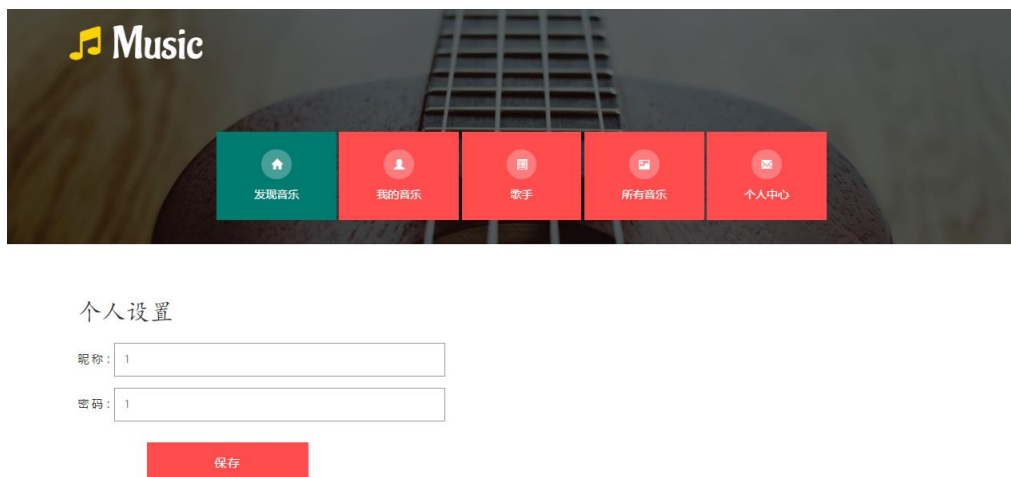


图 4.11 用户个人中心截图

```
@RequestMapping("/updateUser")
```

```
public String updateUser(int id, HttpServletRequest request) {
```

```

User user = userService.findById(id);
user.setUsername(request.getParameter("username"));
user.setUserpwd(request.getParameter("userpwd"));
userService.update(user);
request.setAttribute("user", user);
return "user/index";}

```

(5) 用户收藏歌曲模块

用户在听到喜欢的歌曲可以将这首歌收藏到用户歌单中。

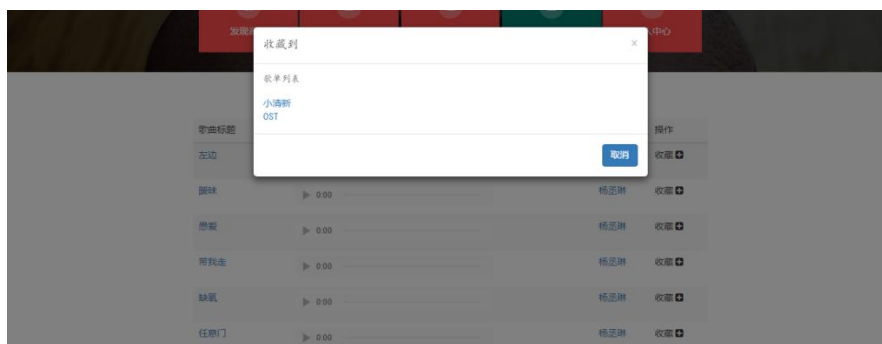


图 4.12 用户收藏歌曲截图

@RequestMapping("/collectSong")

```

public String collectSong(int listid,HttpServletRequest request){
    Integer songid=(Integer) request.getSession().getAttribute("songid");
    Song song=new Song();
    song.setId(songid);
    songService.collectSong(songid, listid);
    return "redirect:/song/getAllSong";
}

```

(6) 用户评论歌曲模块

用户可以评论每一首歌，同时看到这首歌的全部评论。

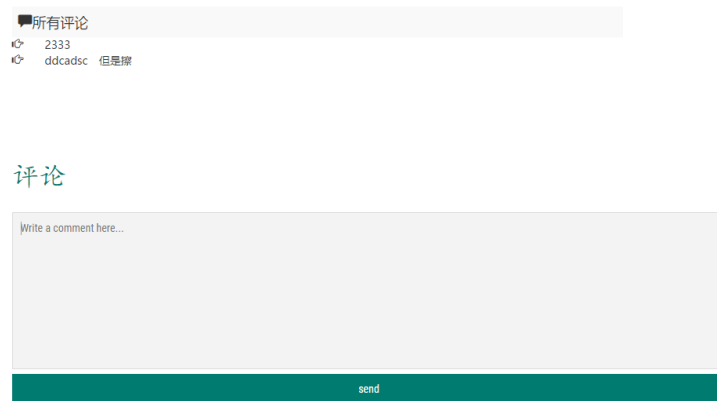


图 4.13 用户评论截图

@RequestMapping("/addsongview")

```
public String addsongview(Songview songview,HttpServletRequest request)
    songviewService.add(songview);
    return "redirect:/song/getAllSong";
}
```

(7) 用户查看某个歌手的全部歌曲模块

用户点击相应的歌手，可以查看到相应歌手的全部歌曲。

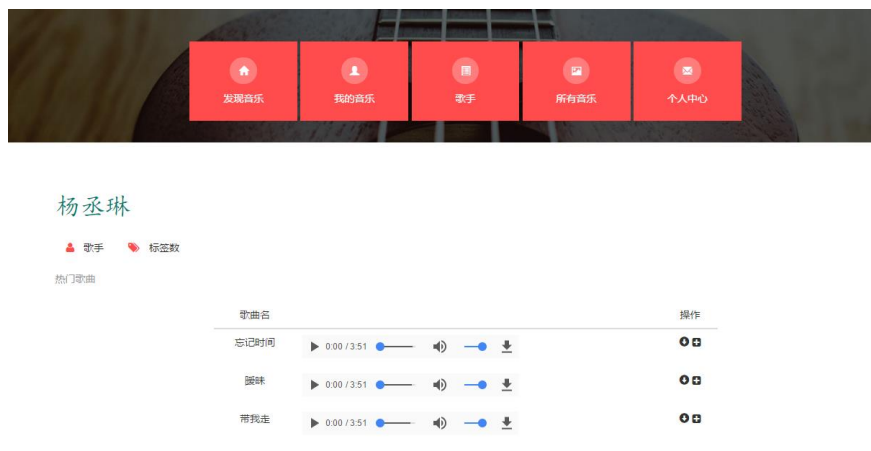


图 4.14 用户查看单个歌手截图

@RequestMapping("/selectOneSinger")

```
public String selectOneSinger(Integer singerid, HttpServletRequest request) {
    List<Singer> selectOne=singerService.selectOne(singerid);
    request.setAttribute("singleSinger", selectOne);
    return "user/singleSinger";}
```

4.3.2 管理员模块

(1) 管理员管理曲库模块

管理员查看到曲库的全部歌曲，并且可以上传、删除、修改曲库信息

ajax 异步请求获取到全部歌曲信息

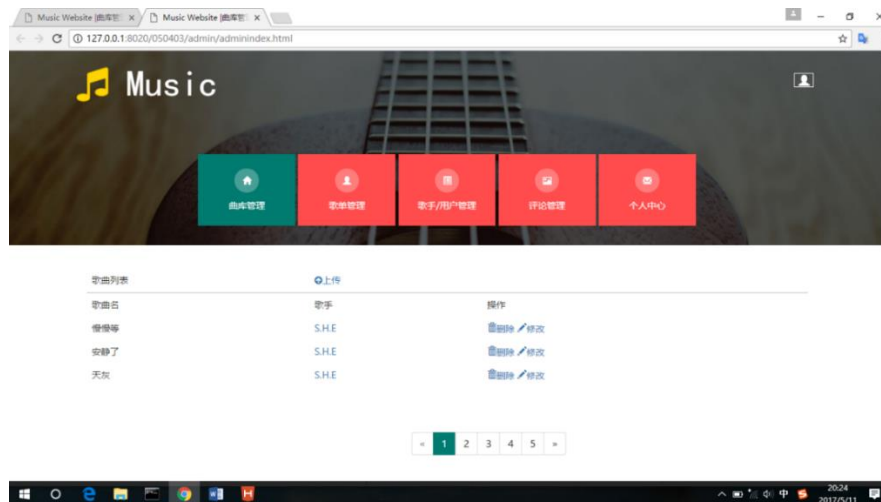


图 4.15 管理员管理曲库截图

增加歌曲信息

@RequestMapping("/addSong")

```
public String addSong(Song song, HttpServletRequest request) {  
    songService.add(song);  
    return "admin/adminindex";  
}
```

(2) 管理员管理歌手模块

管理员可以查看全部歌手信息，并且可以增加、删除歌手信息

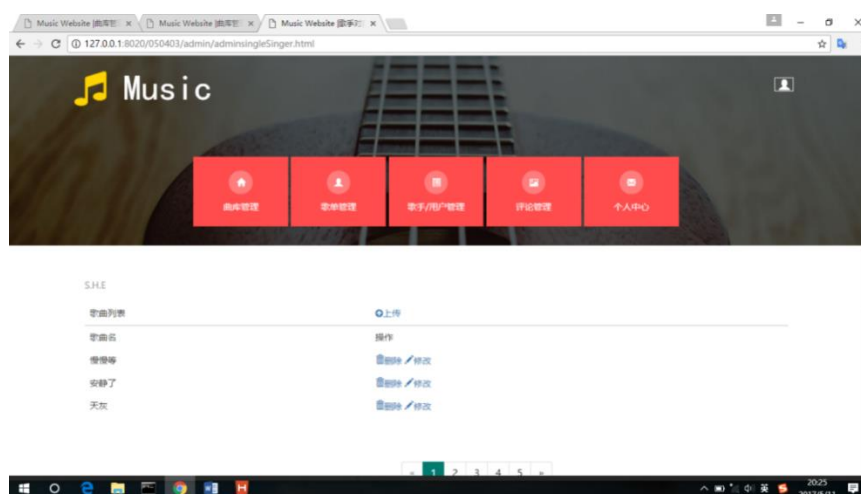


图 4.16 管理员管理歌手截图

```
@RequestMapping("/findSinger")
```

```
public String findSinger(HttpServletRequest request) {
```

```
    List<Singer> findList=singerService.findAll();
```

```
    request.setAttribute("findList", findList);
```

```
    return "/admin/singerUser";}
```

```
@RequestMapping("/OneSinger")
```

```
public String OneSinger(Integer singerid, HttpServletRequest request) {
```

```
    List<Singer> selectOne=singerService.selectOne(singerid);
```

```
    request.setAttribute("oneSinger", selectOne);}
```

(3) 管理员管理评论模块

管理员可以查看全部评论，并且可以对一些不当的评论进行删除。

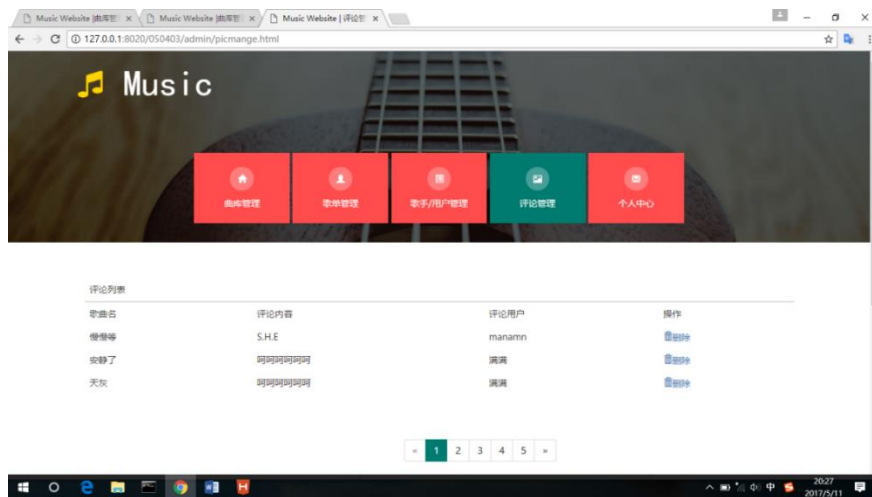


图 4.17 管理员删除评论截图

(4) 管理员个人中心模块

管理员进行个人中心页面，可以修改个人的用户名、密码信息。

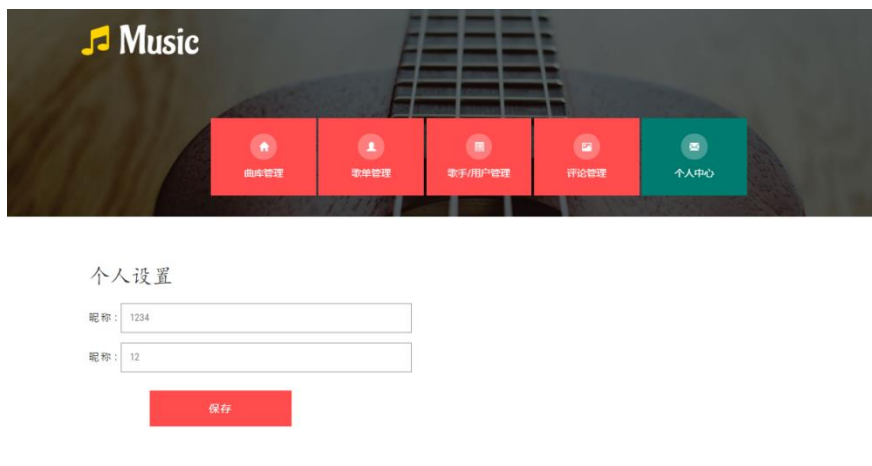


图 4.18 管理员个人中心截图

4.4 网页交互程序

Ajax 异步请求后台数据显示到前台页面输出，实现了页面的局部更新，而不是刷新整个页面，节约了用户等待时间，加强了用户体验。

[illegible]

5 系统测试

5.1 软件测试概述

在大三时学过软件测试，但总是知易行难。任何项目的每一个阶段都会有 bug 的产生，而我们进行测试是为了发现排除 bug，提高系统的健壮性以及可以知兴替。一个好的测试，是它在运行的过程中，遇到的从未出现过或是出现频率较低的 bug，这样可以用最少的人力和最短的时间，发现各种潜在的错误和缺陷。在一个项目的不同生存周期，我们排除问题的办法是不同的，解决问题所付出的代价也是不同的。在需求阶段的问题最容易解决，而在项目编写或是测试维护时才发现的重大问题，往往需要推翻原来的代码，对项目的方法进行重写。如若在交付或运行时出现的 bug，不仅仅会付出大量的人力物力资源，还会降低客户对我们的依赖与信任。

除了在项目的不同阶段测试，在每次测试时的测试方向也有可能不尽相同，如软件测试、硬件测试、网络测试、服务器测试等等。我们根据不同的需求来调整我们的测试方向。通过多方面多角度的测试，来保证系统的质量和可靠性。

我们现在进行的项目测试，是软件系统开发过程中的系统分析与系统设计和实施的最后复查。

软件测试就是利用测试工具按照测试方案和流程对产品进行功能和性能测试，甚至根据需要编写不同的测试工具，设计和维护测试系统，对测试方案可能出现的问题进行分析和评估。执行测试用例后，需要跟踪故障，以确保开发的产品适合需求。

软件测试的几大原则：

1. 软件测试的思想应该贯穿整个项目开发的过程，而并非仅仅在项目的收尾阶段进行测试。软件测试不是一个独立的模块，而是依赖于我们的需求和代码。软件开发的每个环节都可能出现意想不到的 bug。而这些 bug 在项目中造成的影响是具有不可控性和不确定性的。而错误在早期被解决，不仅仅可以减少人力物力资源的消耗，也可以使得开发项目时的思路更加清晰明朗，效率更高。所以要坚持软件开发各阶段的技术评审，把错误克服在早期，从而减少成本，提高软件质量。

2. 对测试用例要有正确的态度。第一，测试用例应当由测试输入数据和预期输出结果这两部分组成；第二，在设计测试用例时，不仅要考虑合理的输入条件，更要注意

不合理的输入条件。因为软件投入实际运行中，往往不遵守正常的使用方法，却进行了一些甚至大量的意外输入导致软件一时半时不能做出适当的反应，就很容易产生一系列的问题，轻则输出错误的结果，重则瘫痪失效！因此常用一些不合理的输入条件来发现更多的鲜为人知的软件缺陷。

3. 测试应当按照严格的测试计划。以避免发生疏漏或者重复无效的工作。

4. 将每一次的测试结果以及反馈报告妥善保存。项目在完善，修改和维护的阶段可能会随时用到。

我们遵循以上的原则，来完成自己的遇到的软件测试问题，消耗最少的资源解决最大的问题，从而可以保证软件的质量。

5.2 测试用例

5.2.1 用户模块

用户模块测试用例：

表 5.1 用户模块测试用例表

测试编号	测试对象	测试用例 (账号密码)	预期结果	实际结果
1	用户登录	mm/123	登录成功	登录成功
2	用户登录	mm/123456	登录失败	登录失败
3	用户注册	注册信息	登录成功	登录成功
4	主页查看	登陆跳转	跳转成功	跳转成功
5	查看歌曲	页面跳转	跳转成功	跳转成功
6	播放音乐	暧昧.mp3	播放成功	播放成功
7	评论测试	手写评论	评论保存成功	评论保存成功

5.2.2 管理员模块

管理员测试用例：

表 5.2 管理员模块测试用例表

测试 编号	测试对象	测试用例 (账号密码)	预期结果	实际结果
1	管理员登录	admin/admin	登录成功	登录成功
2	管理员登录	mm/123456	登录失败	登录失败
4	曲库查看	登陆跳转	跳转成功	跳转成功
5	歌手管理	删除某个歌手	删除成功	删除成功
6	歌单管理	增加新的系统歌单	增加成功	增加成功
7	评论管理	删除不当的评论	删除成功	删除成功

结论

通过本次独立开发毕业设计，让我受益匪浅，大学期间学到的理论知识，在本次毕业设计中得到充分地发挥，也认识到自己的不足，了解了需求对于一个系统来说是非常重要的。在开发过程中，我也遇到了很多技术问题，通过不断地查 API 文档、上网查资料解决了，只有多实践，多动手，熟练地掌握多项技能。

从开始确定论文和系统的主题，总体设计需求分析，参照现有的音乐网站，各个功能模块设计，到实现每一个功能，到最后的测试完成，让我自己清楚地认识到自己的编程能力及知识应用能力，这也是对自己大学四年的一次实践与总结，也真实地了解了开发的基本流程和需求对于一个系统来说非常重要的。

在这次毕业设计中，学到了很多新技术，例如，springMVC 框架、bootstrap 前端框架、Ajax、HTML5 等。发现了运用这些新的技术和框架能够使系统的用户体验度更高，比如，简洁大方的 bootstrap 框架中的模态框。同时在大学中学到的知识，得到了应用，比如 spring 框架。

在这次毕业设计中，锻炼了自己，培养自己不放弃、有耐心的性格，遇到困难不怕困难，勇往直前的精神。

网络的发展，技术的更新，同样学习也是不能间断的。

参考文献

主要参考文献（资料）：

- [1]王艳，清陈红，基于 SSM 框架的智能 web 系统研发设计[J]，计算机工程与设计，2013(12):4751
- [2]陈夫真，基于 SSM 的某高校教室管理信息系统的设计与实现[J]，苏州大学 2012(05):83
- [3]张宇，王映辉，张翔南.基于 Spring 的 MVC 框架设计与实现[J]，计算机工程 2010(04)
- [4]杨群，基于 SSM 的高校排课系统的研究与应用[J]，苏州大学，2013(05):102
- [5]陈君，黄朝兵，在线音乐网站的设计与开发[J]，现代计算机(专业版)，2012(05):68
- [6]孟伟，在线音乐市场的现状分析[J]，企业经济，2005(08):115
- [7]在线音乐网站系统的设计与实现，姜静孙立权，周口师范学院计算机科学与技术学院，2013(08)
- [8]Clement Poncelet, Florent, Jacquemard, Model-based testing for building reliable realtime interactive music systems, Science of Computer Programming, 2016
- [9]Anonymous, Live Music Guide Launches Website Providing Album and Concert Reviews and Commentaries, Wireless News ,2010
- [10]Kumar U A, Deepashree S R, Personal music systems and hearing, The Journal of laryngology and otology, 2016

致谢

这次毕业设计的从选题，构思，分析，代码实现中间的每一个步骤，虽是自己独立完成，但是离不开同学和老师们的帮助。现在回首，越发觉得自己在完成毕设项目中获得的收获和体会，让自己对代码和开发的理解更加深入了一个层次。

在完成项目的过程中，最感谢的是学校指导老师王爱莲和中软基地的古雪老师。王爱莲老师编写代码时思路开阔且有条理，帮我缕清了毕设项目的方向和流程。这为我完成毕设夯实了基础，理解了《音乐网站》是怎么样的一种体系，应该如何实现。从任务设计书到毕业论文的检查，总会在有问题和需要修改的地方提出质疑，为我的毕业设计论文提供了许多宝贵的建议。

在完成毕业设计时，当遇到无法解决的棘手问题，身边的朋友们会向我伸出援助之手。他们向我阐述每个方法具体实现的思路和过程，并指导我调整代码，调整项目。并且能够细心的为我指出项目之中的不足之处和可能会出现问题的地方。非常感谢我的同学带给我的帮助！

最后，最感谢的是自己。在完成项目的过程中一度想要放弃，但是最终解决了自己遇到的每一个难关。不管是自己独立思考，请教他人，查阅文献，搜索资料，尽了自己全部的力量，最终完成了毕业设计项目。凝聚自己心血的成果让自己受益良多，也希望这个项目能够让各位老师喜欢！

在最后，希望自己在日后的学习工作之中，能够继续保持像现在一样的积极状态，去迎接自己人生路上的每一道坎坷。也希望自己能够更加更加勤勉诚挚，不忘初心！

谢谢收看到这里的您！

外文原文

Design and Architecture of Distributed Sound Processing and Database Systems for Web-Based Computer Music Applications

Introduction This article discusses the design and architecture of distributed and World Wide Web-based computer music applications. The discussion is based on the experiences of the Studio On-Line project begun in 1995 at IRCAM, Institut de Recherche et Coordination Acoustique Musique, Paris. Guillaume Ballet was the project's leader, and Rolf Wöhrmann was the primary software developer and software architect. The project received special funding by the French ministers of culture and industry, SUN Microsystems, and Oracle. Studio On-Line illuminates some theoretical aspects of modern client/server and Web technology for computer music applications. It should be noted that similar projects have been started elsewhere; for example, by Xavier Serra (Loureiro and Serra 1997) at the Audiovisual Institute of Pompeu Fabra University, Spain, as well as by Tobias Kunze (Taube and Kunze 1997) at the Center for Computer Research in Music and Acoustics of Stanford University, California.

World Wide Web-Based Computer Music Interfaces Over the last few years, the World Wide Web has become a ubiquitous media technology in the world of computer music, as it has in all information-technology-oriented areas. Initially developed as an integrated multimedia documentation system, the Web has now expanded far beyond its original mission. One reason for this dramatic success has been the original concept of Web "hyperlinks." Traditionally, Web pages are written in a language called Hypertext Markup Language (HTML). Apart from the integration of text, images, and sound, HTML also allows the inclusion of embedded Web addresses, or hyperlinks, which point the user's Web browser to other available online HTML documents. Such hyperlinks are presented to the user as highlighted and underlined text; a simple mouse-click on the text presents the user with the linked page. This direct form of interactivity via hyperlinks has proven adequate for providing access to static information. However, as the Web responds to an ever-growing

need for interactivity, the situation has changed dramatically, due in part to the technology of “dynamic HTML.” With dynamic HTML, the computer providing the HTML pages over the Internet (the Web server) generates new HTML pages dynamically, “on the fly.” A simple example would be a Web page providing the current time; in this case, at each user’s request, the Web server would start a program generating appropriate HTML code including the current time. While such interactivity does not require specific input from the user, the next obvious step is to give the user a means for providing input information to programs producing dynamic HTML. The most widespread use of this technique may be found in Web-based “search engines,” wherein a user types a search key into a Web page and then presses a “submit” button. This “submit” button takes the role of a hyperlink, taking the user to the new dynamically generated HTML page containing a list of search results. Media systems based on Web technology such as this can be used for a wide range of musical applications. The reasons for the success of this new technology in the world of computer music may be understood if we examine the structures of conventional computer music software. Historically, computer music programs have generally been controlled by command-line interfaces, whereby the user entered parameters without any kind of graphical user interface (GUI). Although some graphical or real-time extensions have been developed, even today these commandline programs remain in use, as evidenced by the ongoing success of systems such as Csound and Common Music. These systems are typically controlled by specification files and/or command-line options. A similar kind of nongraphical system would include collections of command-line tools that could be tied together via so-called shell programs. Systems such as this are mainly found in the UNIX environment, since command-line shells provide the primary interaction in UNIX. With the advent of GUIs, this text-based paradigm changed completely. Inspired by the Xerox PARC research laboratories, Apple developed a computer controlled solely by a graphical interface, i.e., a computer system without any shell accessible to the user. For the computer music composer, a new era of interaction via mouse and graphic display thus began. This powerful paradigm allowed for the development of extensive graphical programming systems such as Miller Puckette’s Max, initially developed at IRCAM. Modern sample-editing programs based on graphical manipulation of sound data also exemplify this approach. Despite the advantages of the GUI

paradigm, the progression from command-line systems to graphical applications was not without drawbacks. Using a shell as their work environment allows command-line programs to be more open and flexible than purely graphical applications. The interconnection of shell programs is facilitated by UNIX pipes, which provide an easy way to feed the output of one program to the input of another. These interconnections are not easily constructed with GUI-oriented applications. One solution to this GUI drawback was found when applications moved from single-task programs to multipurpose work environments, such as when MIDI sequencers began to provide digital audio functionality. These large-scale work environments were made extensible by providing application programming interfaces (APIs) for the design of plug-ins. These interfaces enable developers to extend an application by dynamically linking objects to the application. A typical use is a plug-in interface, wherein the application can dynamically load modules conforming to the specific API. A prominent example of this is the TDM plug-in API for the Digidesign ProTools hard-disk recording system. IRCAM's GUI-based musical applications such as Diphone (Rodet and Lefevre 1997) also provide this kind of plug-in interface. While plug-ins must be programmed and compiled by specialists, some work environments also give the average user the possibility to create new extensions to a program by providing interfacebuilder functionality. An example would be the mixer maps used in MIDI sequencers such as Cubase. Finally, there are also some multipurpose nongraphical extensions like proprietary scripting languages or macro processors. Many contemporary GUI-oriented applications based on the large-scale work-environment paradigm have grown to complicated systems requiring a high level of hardware performance. Since audiobased musical applications inherently require a large amount of guaranteed CPU performance— even without a graphical interface—a frequent solution has been to split the application into two separate components: an interface part and a calculation part. The latter component often runs on special-purpose hardware, such as IRCAM's Signal Processing Workstation (ISPW), or the Digidesign ProTools' DSP Farm. In contrast, the graphical interface is frequently run on less-expensive standard personal computers, such as an Apple Macintosh. Communication between the two components is made possible by software client/server technology. One can now see that this split of interface from processing engine is similar to the dynamic HTML concept, where the client's browser interface is separated

from the server's processing. The success of this architecture is dependent on two factors: the richness and complexity of the interaction between client interface and processing server, and the speed of the communication between the two components. Compared to client/server systems like IRCAM's Max/FTS, a Web search engine is primitive with respect to user interaction. However, the main drawback of client/server systems like Max/ FTS is that the server's special-purpose hardware is far too expensive for an average musician or composer to afford. Furthermore, such systems require much more specialized maintenance than personal computers. In contrast, the use of a Web search engine has none of these drawbacks from the client's perspective. Since the server is typically provided by an external organization, the client machine can be a relatively simple computer. Here we find one of the key concepts of IRCAM's Studio On-Line project: IRCAM provides a highly specialized and powerful sound-processing and database server that can be used by clients inside or outside IRCAM, and can be easily accessed with a modern Web browser and an Internet connection. It must be pointed out that the rapid increase in computer-hardware performance in recent years has brought another approach: it is now possible to do both jobs—the interface and the audio processing—on the main CPU. Even Miller Puckette has abandoned the client/server paradigm, as exemplified in his early Max/FTS architecture, for a new single-process system called Pd, which incorporates both the interface and the processing engine in the same process. Commercial audio-processing systems like Steinberg's Cubase VST technology have taken a similar path, and Digidesign has released a ProTools PowerMix version that does not require special DSP cards. Nowadays we live with both paradigms without seeing a definitive solution. New special-purpose hardware is still released, like ProTools²⁴ or Yamaha's new DSP PCI card, giving personal computers the power of professional digital mixers. However, both do-it-all-on-the-CPU systems as well as the client/server systems still have to deal with the enormous size and complexity of their applications. Modern software technology tries to cope with these limitations by inventing new paradigms, such as component-based architectures. The component-based concept is to split an application into smaller subsystems called components. Such systems are much more flexible than a plug-in approach, which is still driven by the large application paradigm. Remote Interfaces and Nomadic Computing Graphical user interfaces can be called remote when the processing engine resides on a

different host than the interface. In the following model, a special kind of Web-based remote interface is discussed. These systems represent the most extreme form of remote interfaces, since a user can access the system from any place in the world using Internet technology. This concept is similar to SUN's idea of "nomadic computing," which allows a user to use any computer anywhere in the world, so long as the computer has a network connection to the server and client software such as a modern Web browser. Even the user's data can be stored permanently on the server, and does not necessarily have to reside on his or her local interface computer. In-house server-based computing is a good solution for institutions that cannot provide each member with a powerful personal computer. Access to file, print, and application servers allows people to work on different computers. Furthermore, institutions such as IRCAM can provide a network speed that is sufficient for in-house multimedia applications. Advantages in the areas of system administration and software management are obvious, since the application software need only be updated on the server. Likewise, for clients, the administration can be reduced to maintenance that is specific to browser software. In such systems, the portability of applications becomes less important—especially for in-house developments—since applications only have to run on the server. Finally, while the server has to be powerful and is therefore quite expensive, clients are much more simple and cheaper in cost. Although this concept of remote interfaces is quite fascinating, it has major drawbacks with respect to musical applications. While the network speed is typically sufficient for classical applications that process alphanumeric data, the network is normally too slow for professional-quality multimedia data. Therefore, the use of music- and audio-application servers has been restricted to people with access to high-speed connections.

Active Web Components Although the concept of dynamic HTML is still widely in use, there has been a growing request for more interactive Web applications. One major problem of dynamic HTML is the restriction that a whole new page must be constructed in response to a user's request. Another problem is that while moving from one dynamic HTML page to the next, there is a need to retain persistent data related to a specific user, such as log-in information. Three solutions to this problem have been found. First, the page can store this kind of information in so-called hidden fields, which are part of the dynamic HTML protocol called the common gateway interface (CGI). Second, the browser itself can store persistent

data for the user in a format called “cookies.” Finally, the persistent data may be stored directly in the URL (hypertext resource locator). Nevertheless, in all three of these scenarios, the Web page itself remains static. In order to give a Web page some intelligent and interactive behavior without requiring a round trip to the Web server, scripting languages such as Netscape’s JavaScript have been invented. With these scripts—embedded in the page’s HTML code—simple forms of interactivity can be realized. Nevertheless, this technology is not sufficient for advanced computer music applications. Only with the advent of active components embedded in Web pages can real applications be realized. The most prominent example of such active components are Java applets—small applications that take a certain region of the Web page, and whose code is transferred over the Internet. Since these so-called movable-code systems require important features—such as security protection—which are not provided in traditional languages, a special language called Java was developed by SUN Microsystems. Java programs do not compile to machine code, but rather to bytecode, which is executed on virtual machines in a manner similar to Smalltalk. However, in contrast to Smalltalk, Java virtual machines are either extensions to browsers or stand-alone programs that interpret the Java bytecode and therefore act as virtual CPUs. Hardware Java CPU chips have even been developed, mostly for use in embedded systems. One can imagine many applications for Java applets in the computer music world. In fact, the client interface of IRCAM’s Studio On-Line project consists primarily of a Java applet, with some dynamic HTML additions. Although there are other possible forms of active Web components, they were rejected by the Studio On-Line team for several reasons. First of all, there is the already-mentioned plug-in concept for Web browsers; one could imagine a special Studio On-Line plug-in. There are several disadvantages with plugins, however, foremost of which is the fact that they must be compiled to machine code, and therefore must be ported to various client platforms. Second, while the paradigm of using external helper programs with Web browsers does exist, the communication is too simple and restricted compared to applets or even plug-ins. In fact, in this case it would be far easier to not even use a browser at all, and to have a C++ application that would directly connect to the server. With this idea the security problems are solved, but compatibility and the guiding philosophy of the Studio On-Line project are lost. Thus, the idea for the client interface in Studio

On-Line was to build a single Java applet. It has actually become quite a large applet compared to the normal size of currently known applets. One solution for downsizing the applet would be to split it into smaller components; however, at the time of the initial Studio On-Line planning, the compound-document technology for Java (called JavaBeans) was not mature enough to be useful, and even browsers did not support this kind of technology. Another solution discussed later is the use of servlets, modern forms of dynamic HTML, which are much more intelligent and based on the Java language. (A similar system was made by NeXT called Web Objects, based on their favored language, Objective-C.) In this way, making dynamic HTML instead of an all-Java applet should decrease the size of the client.

Sound Databases There are several challenges for organizing audio data in databases. For one, the amount of data may be enormous. Modern industrial databases like Oracle, Informix, or Sybase do allow the storage of large binary data directly in table spaces; they also include methods for random access to these BLOBs (Binary Large Objects). However, this technology is quite new, and long-term experience with large amounts of this kind of data, especially in magnitudes of hundreds of gigabytes, is lacking. One typical alternative for storing the sound data directly in table spaces is to store it in a file system. In this approach, the database tables only have references to the files containing the respective binary data. This design has some advantages, since the loss of data is here more easily handled than with corrupted table spaces. For traditional alphanumeric data, one argument for database storage is the inherent security compared to file-system storage. In the case of large amounts of data, storage is not handled by normal, typically insecure file systems, but by special secure architectures such as those used in RAID (Redundant Array of Inexpensive Disks) systems. Here the data is stored redundantly, in the sense that error detection and correction codes are included along with the data. Thus the special file system can automatically replace lost data via the result of the computation between codes and data. For the user or application programmer, this process is transparent, since the handling is done automatically either by the host computer or by special dedicated hardware. The RAID file systems are therefore much more proven for storing large amounts of data than databases, although databases maintain the advantage when used in distributed systems. Another difference with respect to alphanumeric databases is the aspect of query, which is much more

complicated in the case of multimedia data. Efficient, professional-quality search algorithms for audio samples are still not available, although much research is being done in this area. (Some new commercial systems are on the market, such as that developed by Muscle Fish for Informix databases.) The central concept for Studio On-Line's sound database was the distinction made between storage of media data in the RAID system and storage of meta-information in the database. Therefore, the tables in our Oracle database only contain the meta-data of the audio samples, such as instrument type, playing technique, pitch, dynamic, etc. The link between the two is done by storing the file paths of the samples in the database tables. Although we were not familiar with SGI's Studio Central system while designing our Studio On-Line system, their system took a similar approach of storing the media data separately. This commercial system, mainly targeted for asset management, combines the two physical-storage approaches into a single notion of a logical data repository, providing an API for accessing the media database as a compact unit from the viewpoint of the client application. Thus SGI's Studio Central system provides the middleware and a set of APIs for realizing shared access to multimedia databases. The Studio On-Line Project at IRCAM The Studio On-Line project at IRCAM is managed by Guillaume Ballet. The main software architecture was done by Rolf Wöhrmann. The development was first done by Rolf Wöhrmann, and has been continued by Peter Hoffmann, Rodolphe Bailly, and Riccardo Borghesi. The recordings have been directed by Joshua Fineberg and Fabien Lévy. On the hardware side, there is a dedicated server computer, a dual-processor SUN Enterprise 3000 UltraSPARC II. The storage of the sound database is handled by a RAID hard-disk array, RSM 2000, with approximately 200 GB of usable disk space. Silicon Graphics O2 and Indy workstations are used for software development, while a PowerPC.

Macintosh, a PC, and an X-terminal are used for testing and administration. The database used for storing the meta-data of the recorded sounds is Oracle 8. The middleware used is a Visigenic Java/C++ CORBA (Common Object Request Broker Architecture) broker. This is an industry-standard distributed object and component system, which is explained later. While the client applet was written completely in Java, the servers have been written in C++ and Java. The system is available on IRCAM's intranet, as well as on the Internet (at <http://sol.ircam.fr>). Apart from providing access to the sound database, Studio On-Line gives

access to some of IRCAM's sound-processing tools. Sample applications of IRCAM's sophisticated phase vocoder SVP are available, as well as a simple interface to the sinusoidal modeling system, Additive. Figure 1 shows an example of a graphical interface associated to a four-band, time-varying, band-pass filter computed with SVP. The recordings for the sound database have been made for classical instruments like flute, clarinet, oboe, bassoon, trumpet, horn, trombone, violin, viola, cello, double bass, etc. In the future, we will include samples of modern playing techniques such as multiphonics, key clicks, etc. The instruments have been recorded with six microphones, typically with a contact microphone on the instrument, one microphone approximately 1 m away from the instrument, one pair of stereo microphones about 2 m apart, and two microphones at the rear of the room. The reason for using so many microphones was to capture the instrument from as many positions and directions as possible, to facilitate future analysis and studies. The recordings for Studio On-Line have been made in IRCAM's Espace de Projection with a Sonic Solutions 24-bit/ 48-kHz hard-disk recording system, on which they have been also edited and stored in Sonic Solutions proprietary 24-bit format on CD-ROMs. From there they have been converted to noncompressed AIFC files, and finally stored on the RAID system. The most important aspect of the sound-database interface is the problem of how the user finds the right sound he or she needs among the thousands of available samples. The main interface Studio On-Line provides is a selection by categories such as instrument, playing technique, note, dynamic, etc. As shown in Figure 2, the user selects these features one at a time, in any order, until he or she restricts the set of fitting samples to a single sound which can be auditioned or marked for further processing or for downloading. By clicking on each category, the user is presented with a list of the available choices, in dependence with the already-set categories. Even the list of available categories is context-dependent, since all descriptors are not applicable to every instrument (for example, the category "string" is not useful for brass instruments). While this kind of keyword searching in the database is quite successful when a user already knows what kinds of sounds are desired, another search interface based on psychoacoustic sound properties has been developed in cooperation with the IRCAM psychoacoustics team. The interface is based on search by similarity, and is shown in Figure 3. In this case, all sounds of the database have been preanalyzed for special psychoacoustic parameters which are stored in a dedicated

database table. Then the parameters of the search key are compared to a specific set of the database sounds, and a ranking of best fit is established by calculating distance vectors. The sound-processing tools provided by Studio On-Line can be run not only with the sounds of the database, but also with sounds the user uploads. On the server, each user has a certain directory for storage of his or her own sounds and for the results of sound transformations. The upload of custom sounds to the server is done with a simple dynamic HTML form, based on extensions to the CGI protocol which allows server uploads. For security reasons, applets do not have direct access to files on the user's local hard disk. A technology called signed applets might solve this, wherein an external organization approves and signs applets with respect to security issues. Studio On-Line's Three-Tier Architecture As illustrated in Figure 4, Studio On-Line's architecture is a typical three-tier system, an architecture becoming more and more favored over the classical two-tier client/server paradigm (Orfali and Harkey 1998). The main concept of three-tier systems is to place another layer between client and server, which has several advantages over the two-tier architecture. First of all, it allows the use of a thin client. Most of the complexity of the client's communication to the server API is now handled in the middle tier, and is therefore hidden to the client. In fact, the middle tier serves as an abstraction of the server's complexity, reduced to just the application domain's needs. Second, the client is now free from low-level and direct communication with the server. This is quite an important security feature, since there is no possible direct access to the server. Third, all of the client's transactions are bundled and monitored by the middle tier, which improves error handling and keeps the systems consistent. In the Studio OnLine system, the middle tier consists of a set of Java and C++ programs which communicate to each other via a CORBA broker. In addition, CORBA is used in the communication of the client with the middle tier. Since this must be done over the Internet, if necessary, routers and firewalls (the CORBA protocol called IIOP, for Internet Inter-ORB Protocol) may be encapsulated in HTTP, the hypertext transfer protocol normally used for Web pages. Also, the connection port to the server is configurable, and may be assigned to port 80, which is normally open on all routers. As already mentioned, the client consists mainly of a single Java applet. The third tier consists of the actual servers, like the Oracle database, the RAID systems, and all of the IRCAM sound-transformation programs, such as SVP or Additive. The communication

between the middle and the third tier is heterogeneous. The main-database communication is handled by the Java Data base Connection (JDBC) system, which has become a standard in Java database computing. As already mentioned, many aspects of Studio On-Line's architecture are similar to SGI's Studio Central system. Studio Central can be configured in a two-tier or three-tier manner. Also, SGI uses the CORBA standard for communication, although theirs is based on IONA's Orbix broker, while Studio On-Line is based on Visigenic's broker. The SGI Studio Central splits the storage of meta-information from the storage of media data. However, Studio Central is more flexible, since it just provides a set of APIs. Thus, for each storage system a driver must be written, while for the client the kind of the storage system is transparent, because it is hidden by the API. Studio Central also provides much more advanced modeling of the media objects residing on the server. In order to support custom extended data types by simply defining them in specification files, SGI has defined a generic specification model. Targeted for all kinds of multimedia data like audio, images, animations, and video, Studio Central provides a professional framework for distributed-media applications.

Client/Server Technology and Distributed Computing in Computer Music

We now take a closer look at the technologies used in computer music for client/server communication, and at the reasons they are used for different applications. In fact, many classical applications use custom-made protocols mostly realized via a stream-based technology like UNIX sockets or pipes, which are relatively simple to realize. However, since they are low-level protocols and do not use any middleware technology, all error handling, communication handshakes, etc., must be performed by the application itself. While the design of these kinds of applications are mostly driven by hardware issues—i.e., the need to run the audio engine on special-purpose hardware—modern applications have tended to move back to single-process applications like Miller Puckette's Pd system or Steinberg's Cubase VST. For Web-based computer music applications using Java applets, client/server communication is a necessity, at least while doing audio processing, since the virtual machine in the browser is not fast enough to handle audio (although it might be sufficient for MIDI-related tasks). Therefore, the applet in the Studio On-Line project is only used for the interface to the server. Since the communication over the Internet is much more complicated and unstable than in hardware-motivated client/ server systems, intelligent

communication protocols are required. Furthermore, since the Java language used for applets is object oriented, we required a seamless integration of client/server communication into the object-oriented programming paradigm. In contrast to this, there did already exist a classical client/server technology based on the procedural programming paradigm: SUN's remote procedure calls (RPC), which are used in their network file system (NFS). One of the first commercially available distributed-object systems was Distributed Objects in NeXTSTEP. Initially based on NeXT's operating system and Objective-C, NeXTSTEP's Distributed Objects have now been ported to other UNIX systems. However, owing to its platform restrictions and the history of the NeXT company, their distributed-object system has not become a prominent one in the market, and it is not clear whether or not Rhapsody will change this situation. More general and portable solutions had to be found, and thus the CORBA system has become an industry standard. CORBA, the Common Object Request Broker Architecture, is a language- and operating-system-independent distributed-object system based on a component model (for the complete specification, see the Web at http://www.omg.org/corba/corba_iop.html). In so-called language mappings, the integration for specific programming languages is defined. Currently there are mappings for C++, Smalltalk, Java, and even C. The heart of a CORBA system is the broker, which can be viewed as an object bus (Mowbray and Malveau 1997). Several companies are now providing CORBA brokers for all types of operating systems like UNIX, Windows 95, Windows NT, NeXTSTEP, etc; even public domain brokers are available on the Internet. While CORBA is widely used in industrial applications (Mowbray and Zahavi 1995) and is accepted by many companies, the only competitor is Microsoft's DCOM system. While DCOM was for quite some time restricted to C++, Microsoft has now also included Java in their DCOM system. SUN itself has also developed a distributed-object system for use in their Java language: the Java Remote Messaging Interface (RMI). However, this system is restricted to Java only, and will not be available in other programming languages. Efforts are currently being made to merge Java RMI into the CORBA system by adopting CORBA's IIOP protocol for Java RMI. Distributed-object systems for use in applets must meet certain requirements. First of all, they have to be implemented in Java. Second, they have to be efficient for communicating over the Internet. Third, they have to provide automatic error

handling for problems like connection loss, which happens quite often on the Internet. And last, but not least, the protocol has to be transported over firewalls. Only the CORBA system could meet all of our requirements; thus we based Studio On-Line on CORBA. As a side effect, we gained the flexibility to write our servers in Java as well as C++, since with CORBA's language independence they can communicate easily with each other. For Studio On-Line, the most important requirement was the ability to communicate across firewalls. We realized this by using Visigenic's Gatekeeper program, which acts as a kind of proxy that may, if necessary, encode the CORBA-specific communication protocol IIOP in the standard HTTP protocol used for HTML Web pages. This technology is called tunneling, and makes it possible to work with Studio On-Line anywhere in the world. Some final comments should be made concerning two technologies called middleware and component architectures. The former is simply a new term for all technologies that provide application programmers with higher-level application-oriented APIs to low-level protocols. Middleware creates abstractions from the underlying transport protocol. Prominent examples include various database-access middleware layers such as Microsoft's ODBC or SUN's JDBC. Studio On-Line also uses middleware for accessing the meta-data in our database. Studio On-Line is mainly Javabased; thus we use the JDBC system, since it is database independent in contrast to the proprietary APIs provided in Oracle. The second technology mentioned is the component-architecture paradigm. After the shift in application development to object-oriented architectures, it became apparent that while solving many problems of traditional procedural programming, we nevertheless acquired new kinds of problems. Most of these are due to the enormous complexity of large object-oriented systems. One reason for this complexity is the amount of dependencies between object interfaces, which can become difficult to manage. A solution was to group object classes into functional subsystems called components. These component-based architectures eventually led to the idea of compound documents, i.e., document containers that can contain any kind of application components tied together within a generic framework. Although Apple's proposal of such a structure called OpenDoc died relatively quickly due to its enormous complexity, Microsoft based its ActiveX system on their component-architecture model. Even SUN devised a component- and compound-document architecture for their Java language called JavaBeans. Problems,

Solutions, and Future Work of Studio On-Line The biggest technological challenge of the Studio On-Line project has already been mentioned: the network speed available to most people is still too slow for audio applications. While this can be solved in consumer audio with intelligent audio compression, this cannot be a solution for the professional audio quality demanded by computer music. However, the network-speed bottleneck may change in the future as the demand for fast networks is eventually answered by industry. Another technological problem is the growing incompatibility of browsers, as the recent discussions between Netscape and Microsoft demonstrate. Also, since there is heavy commercial pressure to release ever faster and newer versions of their browsers, it is not surprising that the new technology offered commercially is sometimes not finished, is often unstable, and is still changing in design. Ultimately, Studio On-Line's decision to use an applet for the client user interface may cause problems for users not trained in this kind of technology. Most users understand simple troubleshooting methods, such as restarting their computer when the system crashes; however, they are not accustomed to the special types of problems that can occur while using applets. A possible solution for the client problem is the idea of client scalability. We need simpler ways to access the basic services of Studio On-Line without using high-end applet technology. In fact, we need to provide some parts of Studio On-Line in pure dynamic HTML to which users are accustomed and for which there are no browser-compatibility issues. The already-mentioned servlet approach could realize this. In contrast to CGI-based dynamic HTML, which starts a small program for each user's request, servlets are permanently running Web-server extensions written in Java. For each user's request, only a certain method in the servlet object is called. Since servlets are persistent servers, the user's problem of storing persistent data from page to page is solved quite easily. Servlets are programmed in Java, which makes it easy to reuse our Java and C++ servers by means of CORBA. Also, a direct database connection via JDBC is possible with servlets. Therefore it is possible to provide both kinds of client interfaces—applets and dynamic HTML—while using the same server applications. In the future, Studio On-Line must be integrated with more of IRCAM's sound-processing applications. The challenge here is to simplify the complex GUIs in order to implement them efficiently via applets. Envisaged are applications that provide non-real-time access to IRCAM's 3-D audio system, Spatialisateur

(Jot and Warusfel 1995), running on FTS but providing a custom user interface. Also, the sinusoidal-modeling tool HMM (Depalle, Garcia, and Rodet 1993), based on hidden Markov models, might be integrated into the system. Adaptation of the existing architecture is also envisaged for educational applications in the framework of experiments on new domestic services.

Acknowledgments Many thanks to Joseph Butch Rovin for his corrections to the original text.

References Depalle, P., G. Garcia, and X. Rodet. 1993 (Minneapolis, Minnesota). "Tracking of Partial for Additive Sound Synthesis Using Hidden Markov Models." Proceedings of the 1993 International Conference on Acoustics, Speech and Signal Processing. Los Alamitos, California: IEEE. Jot, J.-M., and O. Warusfel. 1995. "A Real-Time Spatial Sound Processor for Music and Virtual Reality Applications." Proceedings of the 1995 International Computer Music Conference. San Francisco: International Computer Music Association. Loureiro, R., and X. Serra. 1997. "A Web Interface for a Sound Database and Processing System." Proceedings of the 1997 International Computer Music Conference. San Francisco: International Computer Music Association, pp. 360–363. Mowbray, T. J., and R. C. Malveau. 1997. CORBA Design Patterns. New York: Wiley. Mowbray, T. J., and R. Zahavi. 1995. The Essential CORBA: Systems Integration Using Distributed Objects. New York: Wiley. Orfali, R., and D. Harkey. 1998. Client/Server Programming with Java and CORBA. New York: Wiley. Rodet, X., and A. Lefevre. 1997. "The Diphone Program: New Features, New Synthesis Methods, and Experience of Musical Use." Proceedings of the 1997 International Computer Music Conference. San Francisco: International Computer Music Association. Taube, H., and T. Kunze. 1997. "An HTTP Interface to Common Music." Proceedings of the 1997 International Computer Music Conference. San Francisco: International Computer Music Association, pp. 204–207

基于 Web 的计算机音乐应用的分布式声音处理和数据库系统的设计与架构

介绍

本文讨论了设计和架构的分布式和万维网计算机音乐应用。讨论是基于的关于 Studio On-Line 项目的经验。1995 年开始于 IRCAM, Institut de Recherche- et 协调声学音乐, 巴黎纪尧姆芭蕾舞团是项目的领导者, 也是罗尔夫 Wörmann 是主要的软件开发人员和软件架构师。该项目获得特别奖法国文化部长的资助行业, SUN 微系统和 Oracle。工作室在线阐释了一些理论方面现代客户端/服务器和 Web 技术的计算机音乐应用。应该指出的是类似的项目已经在其他地方开始了例如, Xavier Serra (Loureiro 和 Serra, 1997) 在庞贝法布拉大学视听学院, 西班牙, 以及 Tobias Kunze (Taube 和 Kunze 1997) 在计算机研究中心在斯坦福大学音乐与声学学院, 加州。

万维网计算机音乐接口在过去的几年里, 万维网已经有了成为无处不在的媒体技术计算机音乐的世界, 因为它在所有的信息面向技术的领域。最初开发作为一个综合多媒体文件系统, 网络现在已经远远超过了它原来的使命。这个戏剧性成功的一个原因一直是 Web 的原始概念“超级链接”。传统上, 网页被写入在一种称为超文本标记语言的语言中 (HTML)。除了整合文字, 图像, 和声音, HTML 也允许包含的嵌入式 Web 地址或超链接, 其中将用户的 Web 浏览器指向其他可用在线 HTML 文档。提出了这样的超链接以突出显示和下划线显示给用户文本; 一个简单的鼠标点击文字就可以了用户与链接的页面。

这种通过超链接的直接交互形式已被证明足以提供静态访问信息。但是, 随着 Web 响应情况的日益增长的需求部分原因在于技术发生了巨大变化的“动态 HTML”HTML, 提供 HTML 页面的电脑通过互联网 (Web 服务器) 生成新的 HTML 页面动态, “飞行”。一个简单的示例将是提供当前的网页时间; 在这种情况下, 在每个用户的请求下, Web 服务器将启动一个生成适当的程序包含当前时间的 HTML 代码。虽然这种交互性不需要具体的从用户输入, 下一个明显的一步是以向用户提供用于提供输入信息的手段到产生动态 HTML 的程序。这种技术的最广泛使用可能是在基于 Web 的“搜索引擎”中发现, 其中 a 用户将搜索键键入网页, 然后按“提交”按钮。这个“提交”按钮承担超链接的角色, 将用户带到新的动态生成的 HTML 页面包含搜索结果列表。基于 Web 技术的媒体系统这可以用于广泛的音乐应用。

这个新成功的原因电脑音乐世界的技术可能如果我们检查传统的结构，应该理解电脑音乐软件。历史上，电脑音乐节目有一般由命令行界面控制，用户输入的参数没有任何类型的图形用户界面（GUI）。虽然有些图形或实时扩展已经开发，甚至今天这些命令行方案仍然在使用，正如系统的持续成功，如 Csound 和普通音乐这些系统通常被控制通过规范文件和/或命令行选项。一种类似的非地理系统将包括命令行工具的集合可以通过所谓的 shell 程序绑定在一起。这些系统主要见于 UNIX 环境，因为命令行 shell 在 UNIX 中提供主要交互。随着 GUI 的出现，这种基于文本的范式彻底改变灵感来自施乐苹果开发了 PARC 研究实验室计算机仅由图形界面控制，即没有任何外壳可访问的计算机系统给用户对于电脑音乐作曲家，通过鼠标和互动的新时代图形显示就开始了。这个强大的典范允许发展广泛图形编程系统如 MillerPuckette 的 Max，最初在 IRCAM 开发。

基于图形的现代样本编辑程序声音数据的操纵也体现了这一点方法尽管 GUI 模式的优点，从命令行系统到图形化应用程序并没有缺点。使用一个 shell 作为其工作环境允许命令行方案更加开放灵活比纯图形应用程序。互连的 shell 程序由 UNIX 来实现管道，提供了一种简单的方式来输送输出的一个程序输入另一个。这些互连不容易构建面向 GUI 的应用程序。发现了这种 GUI 缺点的一个解决方案当应用程序从单任务程序移动时多用途工作环境，如当 MIDI 音序器开始提供数字时音频功能。这些大型工作环境通过提供应用程序可扩展编程接口（API）插件设计。这些接口使开发人员能够使用动态扩展应用程序将对象链接到应用程序。典型的用途是插件接口，其中应用程序可以动态地执行加载模块符合具体要求 API。TDM 的一个突出的例子就是 TDM Digidesign ProTools 硬盘的插件 API 录音系统 IRCAM 的基于 GUI 的音乐应用如 Diphone（Rodet 和 Lefevre）1997）也提供了这种插件接口。插件必须编程和编译由专家，一些工作环境也给普通用户创造新的可能性通过提供 interfacebuilder 来扩展程序功能。一个例子就是 MIDI 音序器中使用的混音器地图，如 Cubase。最后还有一些多用途非专有扩展名，如专有脚本语言或宏处理器。

许多当代 GUI 面向应用程序基于大规模的工作环境范式已经发展到复杂的系统需求高水平的硬件性能。自从音频基础音乐应用本身就需要一个大量保证 CPU 性能 - 即使没有图形界面 - 一个频繁的解决方案已将应用程序分解为两个单独的组件：接口部分和计算部分。后一种组件经常运行专用硬件，如 IRCAM 的信号处理工作站（ISPW）或 Digidesign ProTools 的 DSP 农场。相比之下，图形界面经常以较便宜的标准运行个人

电脑，如苹果 Macintosh。两个组件之间的通信是通过软件客户端/服务器技术实现。现在可以看到这个接口的拆分处理引擎与动态 HTML 类似概念，客户端的浏览器界面与服务器的处理分离。成功的这种架构取决于两个因素：丰富多彩的复杂性客户端界面与处理之间的交互服务器，以及通信速度两个组件之间。与客户端/服务器系统相比 IRCAM 的 Max / FTS 是一款网页搜索引擎关于用户交互。但是，那客户端/服务器系统的主要缺点如 Max / FTS 是服务器的专用硬件对一般音乐家来说太贵了，作曲家负担得起。此外，这种系统需要更专业的维护比个人电脑相反，使用 Web 搜索引擎没有这些缺点客户的观点。因为服务器是典型的由外部组织提供，客户端机器可以是比较简单的电脑。在这里，我们找到了 IRCAM 的关键概念之一工作室在线项目：IRCAM 提供高度的专业强大的声音处理和数据库服务器可以由客户端使用或在 IRCAM 之外，可以轻松访问现代 Web 浏览器和 Internet 连接。必须指出的是，快速增长电脑硬件性能近年来带来了另一种方法：现在有可能做这两个工作 - 界面和音频处理主 CPU。甚至米勒 派克特也是放弃了客户端/服务器范例，例如在他早期的 Max / FTS 架构中，为一个新的单过程系统称为 Pd，其中包含界面和处理引擎在同一个过程中。商业音频处理系统如 Steinberg 的 Cubase VST 技术 Digidesign 已经采取了类似的道路 ProTools PowerMix 版本没有需要特殊的 DSP 卡。现在我们生活在两个范式之中看到一个明确的解决方案。新专用硬件仍然被释放，像 ProTools | 24 或雅马哈新的 DSP PCI 卡，给个人电脑专业数字混音器的力量。但是，这两个都是一样的 CPU 系统以及客户端/服务器系统仍然需要处理他们的庞大规模和复杂性应用程序。现代软件技术试图通过发明新的应对这些限制范例，如基于组件的架构。基于组件的概念是拆分应用到称为组件的较小子系统中。这样的系统比 a 更灵活插件方法，这仍然是由大的驱动应用范例。远程接口和游牧计算图形用户界面可以称为远程处理引擎驻留在不同的主机上比界面。在下面的模型中，一个特殊的讨论了一种基于 Web 的远程接口。

这些系统代表了最极端的形式远程接口，因为用户可以访问系统从世界上任何使用互联网的地方技术。这个概念类似于 SUN 的想法“游牧计算”，允许用户使用任何一台电脑在世界任何地方，只要是计算机具有到服务器的网络连接和客户端软件，如现代 Web 浏览器。即使用户的数据也可以永久存储服务器，并不一定必须驻留在他或她的本地界面计算机上。内部基于服务器的计算是一个很好的解决方案对于不能提供每个的机构会员与强大的个人电脑。访问允许文件，打印和应用程序服务器人们在不同的电脑上工作此外，IRCAM 等机构可以提供网络速度足以满足内部多媒体的需求应用程序。领

域的优势系统管理和软件管理是显而易见的，因为应用软件需要只能在服务器上更新。同样，对于客户来说，管理可以减少到维护这是特定于浏览器软件。在这样的系统，应用程序的可移植性成为自从以来，不太重要 - 尤其是内部发展应用程序只需要运行服务器。最后，服务器必须是强大的因此相当昂贵，客户很多更简单和更便宜的成本。虽然这个远程接口的概念是相当迷人，它有很大的缺点到音乐应用。而网络速度通常足以用于古典应用那个进程的字母数字数据，网络通常对于专业质量的多媒体来说太慢了数据。因此，使用音乐和音频应用程序服务器已被限制人们可以访问高速连接。

活动 Web 组件虽然动态 HTML 的概念仍然存在广泛使用，越来越多的要求更互动的 Web 应用程序。一个专业动态 HTML 的问题是这样的限制必须构建一个全新的页面根据用户的要求另一个问题是从一个动态 HTML 页面移动到下一个，需要保留与 a 有关的持久性数据特定用户，如登录信息。三个解决方案已经发现这个问题。首先，页面可以存储这种信息在所谓的隐藏的领域，这是动态的一部分 HTML 协议称为通用网关接口（CGI）。

二，浏览器本身可以存储持久性用户以一种格式调用数据“cookies”。最后，持久性数据可能是直接存储在 URL（超文本资源定位符）中。然而，在所有这三种情况下，网页本身保持静态。为了给一个网页一些聪明和交互式行为，无需往返到 Web 服务器，脚本语言如 Netscape 的 JavaScript 已经被发明了。与这些脚本嵌入在页面的 HTML 中可以实现代码简单的交互形式。不过，这项技术还不够先进的电脑音乐应用。只有与网络中嵌入的活动组件的出现页面可以实现真正的应用程序。这个活跃的最突出的例子组件是 Java 小程序 - 小应用程序它占据了网页的某个区域其代码通过互联网传输。以来这些所谓的可移动代码系统需要重要功能 - 如安全防护 - 这不是传统的语言，a 被称为 Java 的特殊语言由 SUN 开发微系统。

Java 程序不编译到机器代码，而是字节码，这是执行的在虚拟机上以类似的方式短暂聊天。但是，与 Smalltalk，Java 相反虚拟机是浏览器的扩展或解释 Java 的独立程序字节码，因此充当虚拟 CPU。硬件 Java CPU 芯片甚至已经开发出来，主要用于嵌入式系统中。可以想象许多 Java 应用程序苹果在计算机音乐界。其实，IRCAM 的 Studio On-Line 的客户端界面项目主要由 Java applet 组成一些动态的 HTML 添加。虽然那里是活动 Web 组件的其他可能形式，他们被在线工作室拒绝团队有几个原因首先是有的已经提到的 Web 浏览器的插件概念;可以想象一个特别的在线工作室插入。插件有几个缺点，然而，其中最重要的是这样的事实它们必须被编译成机器代码，和因此必须移植到

各种客户端平台。

使用外部的范式帮助程序与 Web 浏览器确实存在，沟通过于简单和限制与 applet 甚至插件相比。其实，在这种情况下，甚至不要使用它会更容易浏览器，并拥有一个 C++ 应用程序将直接连接到服务器。有了这个想法安全问题得到解决，但兼容性和工作室的指导理念在线项目丢失。因此，Studio 中客户端界面的想法在线是构建单个 Java 小程序。实际上成为相当大的 applet 相比当前已知的小程序的正常大小。一个解决方案小程序的缩小将是分裂变成较小的部件；然而，在的时候最初的 Studio On-Line 规划，复合文件 Java 技术（称为 JavaBeans）不够成熟，不够有用，甚至浏览器都不支持这种技术。稍后讨论的另一个解决方案是使用 servlet，现代形式的动态 HTML，这是更加智能和基于的 Java 语言。（类似的系统是由 NeXT 称为 Web 对象，基于他们的青睐语言，Objective-C。）以这种方式，使动态 HTML 而不是全 Java 小程序应该减小客户端的大小。声音数据库组织音频有几个挑战数据库中的数据。

一，数据量可能是巨大的现代工业数据库如 Oracle, Informix 或 Sybase 确实允许存储大型二进制数据直接在表空间中；他们也包括随机访问这些 BLOB 的方法（二进制大对象）。但是，这项技术是相当新，长期的经验与大这种数据的数量，特别是在数量上数百 GB，缺乏。一个典型的用于直接存储声音数据的替代方法表空间将其存储在文件系统中。在这种方法中，数据库表只有引用包含相应二进制数据的文件。这个设计有一些优点，因为数据丢失在这里更容易处理而不是损坏桌子空间对于传统的字母数字数据，一个数据库存储的参数是固有的安全性与文件系统存储相比。在大量数据的情况下，存储是不由正常的，通常不安全的文件系统处理，而是通过特殊的安全架构，如那些用于 RAID（冗余的廉价阵列）磁盘）系统。这里数据被冗余地存储，在这个意义上，错误检测和校正码与数据一起包括在内。因此专用文件系统可以自动替换通过计算结果丢失数据代码和数据之间。对于用户或应用程序程序员，这个过程是透明的，因为处理由... 自动完成主机或专用硬件。因此，RAID 文件系统更多被证明用于存储大量数据而不是数据库，虽然数据库保持优势用于分布式系统。另一个区别相对于字母数字数据库是查询方面，要复杂得多在多媒体数据的情况下。高效，专业的品质音频样本的搜索算法仍然没有，尽管很多研究在这方面做了。（一些新的商业广告系统在市场上，如开发的通过 Muscle Fish for Informix 数据库。）Studio On-Line 的声音的中心概念数据库是存储之间的区别的媒体数据在 RAID 系统和存储中数据库中的元信息。因此，我们的 Oracle 数据库中的表仅包含音乐样本的元数据，如乐器类型，播放技巧，音调，动态等两者之

间的链接是通过存储文件完成的数据库表中样本的路径。虽然我们不熟悉 SGI 的 Studio 中央系统，同时设计我们的在线工作室系统，他们的系统采取了类似的方法分别存储媒体数据。这个商业广告系统主要针对资产管理，将两种物理存储方法结合使用提供逻辑数据存储库的单一概念用于作为访问媒体数据库的 API 从客户端应用的角度来看紧凑型单元。因此，SGI 的 Studio Central 系统提供中间件和一组实现的 API 共享访问多媒体数据库。IRCAM 工作室在线项目 IRCAM 的 Studio On-Line 项目管理由纪尧姆芭蕾舞团主要的软件架构由 Rolf Wöhrmann 完成。发展是由罗尔夫·沃尔曼 (Rolf Wöhrmann) 首先完成的由 Rodolphe 的 Peter Hoffmann 继续说道 Bailly 和 Riccardo Borghesi。录音有由 Joshua Fineberg 和 Fabien Lévy 执导。在硬件方面，有一个专用的服务器计算机，双处理器 SUN Enterprise 3000 UltraSPARC II。存储声音数据库由 RAID 硬盘阵列 RSM 2000 处理，具有大约 200 GB 的可用磁盘空间。Silicon Graphics O2 和 Indy 工作站是用于软件开发，而 PowerPC 使用 Macintosh，PC 和 X 终端测试和管理。用于存储元数据的数据库记录的声音是 Oracle 8。中间件使用的是 Visigenic Java / C++ CORBA (Common 对象请求代理架构) 代理。这个是行业标准的分布式对象和组件系统，稍后解释。当时客户端小程序完全用 Java 编写，服务器已经用 C++ 和 Java 编写的系统也可以在 IRCAM 的内部网上使用如互联网 (<http://sol.ircam.fr>)。另外从提供访问声音数据库，Studio 在线可以访问一些 IRCAM 声音处理工具。样品申请 IRCAM 的复杂相位声码器 SVP 是可用，以及与正弦曲线的简单接口建模系统，添加剂。图 1 显示与 a 相关联的图形界面的示例计算的四频带，时变，带通滤波器与 SVP。声音数据库的录音已经为古典乐器，如长笛，单簧管，双簧管，巴松，喇叭，喇叭，长号，小提琴，中提琴，大提琴，低音提琴等。未来，我们会包括现代演奏技巧的样本如多声道，按键等。仪器已经录制了六个麦克风，通常使用仪器上的接触式麦克风，一个麦克风约 1 米远从仪器，一对立体声麦克风大约 2 米，和两个麦克风房间的后方。使用这么多的原因麦克风是用来捕获仪器的尽可能多的职位和方向，方便未来分析研究。录音对于 Studio On-Line 已经在 IRCAM 中进行了 Espace de Projection 与声波解决方案 24 位/48 kHz 的硬盘录像系统，就可以了也被编辑并存储在 Sonic Solutions 中专有的 24 位格式的 CD-ROM。从那里它们已被转换为非压缩 AIFC 文件，最后存储在 RAID 系统上。声音数据库的最重要的方面界面是用户发现问题他或她在数千人中所需的正确声音的可用样品。主界面 Studio On-Line 提供的是按类别选择如乐器，演奏技巧，笔记，动态等。如图 2 所示，用户选择这些功能一次一个，按任何顺序，直到他或她将一套拟合样本限制在一个单个声音可以被试镜或

标记用于进一步处理或下载。点击在每个类别上，用户被呈现依赖于可用选择的列表已经设定的类别。甚至可用的列表类别是上下文相关的，因为所有描述符不适用于每个仪器（例如，类别“string”对于没有用铜管乐器）。而这种关键字在数据库中搜索当用户已经相当成功知道什么样的声音是需要的，另一个基于心理声学的搜索界面物业已经合作开发 IRCAM 心理声学队。界面就是基于相似度搜索在这种情况下，数据库的所有声音都有为特殊心理声学参数进行了预分析它们存储在专用数据库中表。那么搜索键的参数是相比一组具体的数据库声音，并通过计算建立最佳匹配的排名距离向量 Studio 提供的声音处理工具在线可以运行不仅与声音数据库，还有用户上传的声音。在服务器上，每个用户都有一个目录存储他或她自己的声音和声音转换的结果。上传的定制声音到服务器是用一个简单的动态 HTML 表单，基于扩展到允许服务器上传的 CGI 协议。为了安全起见，小程序没有直接访问权限到用户本地硬盘上的文件。一种技术所谓的签名小程序可能会解决这个问题，其中外部组织批准和在安全问题上签署小应用程序。工作室在线的三层架构如图 4 所示，Studio On-Line 的架构是一个典型的三层体系，一个架构越来越受到青睐经典的双层客户端/服务器范例（Orfali 和 Harkey 1998）。三层的主要概念系统是在客户端之间放置另一层和服务器，它有几个优点两层架构。首先，它允许使用的瘦客户端。大部分的复杂性现在客户端与服务器 API 的通信在中层处理，因此隐藏给客户事实上，中层是一个抽象服务器的复杂性降低到了只是应用程序域的需求。

二，客户现在没有低级和直接的沟通与服务器。这是非常重要的安全功能，因为没有可能直接访问服务器。第三，所有客户的交易被捆绑和监控中间层，改进了错误处理保持系统一致。在 Studio OnLine 中系统，中间层由一组组成与通信的 Java 和 C++ 程序彼此通过 CORBA 代理。另外，在通信中使用 CORBA 的客户端与中间层。既然这样必须通过互联网完成路由器，必要的话和防火墙（CORBA 协议称为 IIOP，用于 Internet Inter-ORB 协议）可以被封装在 HTTP 中，超文本传输协议正常用于网页。另外，连接端口到服务器是可配置的，可以分配到 80 端口，这是通常打开的路由器如前所述，客户端包含主要是单个 Java 小程序。第三层组成的实际服务器，如 Oracle 数据库，RAID 系统和所有 IRCAM 声音转换程序，如 SVP 或添加剂。中间的沟通第三层是异构的。主数据库通信由 Java 数据处理基础连接（JDBC）系统，已经成为 Java 数据库计算中的一个标准。如前所述，Studio 的许多方面在线的架构类似于 SGI 的 Studio 中央制度。Studio Central 可以配置以两层或三层的方式。另外，SGI 使用 CORBA 标准的通信虽然他们的基础是 IONA 的 Orbix 经纪人，而 Studio 在线是基于 Visigenic 的经纪人 SGI Studio

Central 分割元信息的存储从媒体数据的存储。然而，Studio Central 更灵活，因为它提供一套 API。因此，对于每个存储系统 a 司机必须写，而为客户存储系统的种类是透明的，因为它被 API 隐藏。Studio Central 也提供更先进的媒体建模驻留在服务器上的对象。为了支持通过简单定义定制扩展数据类型他们在规范文件中，SGI 已经定义了一个通用的规格型号。针对各种各样的多媒体数据如音频，图像，动画，和视频，Studio Central 提供专业分布式媒体应用程序框架。客户端/服务器技术和分布式计算机音乐计算我们现在仔细看看这些技术用于计算机音乐用于客户端/服务器通信，并以其用于不同的原因为由应用程序。其实很多古典应用程序使用大多数实现的定制协议通过基于流的技术，如 UNIX 插座或管道，这是相对简单的实现。但是，由于它们是低级协议并且不要使用任何中间件技术，所有的错误处理，沟通握手等，必须由应用程序本身执行。而这些应用的设计是主要由硬件问题驱动，即需要在专用硬件上运行音频引擎应用程序倾向于移动回到像 Miller 这样的单进程应用程序 Puckette 的 Pd 系统或 Steinberg 的 Cubase VST。对于基于 Web 的计算机音乐应用程序使用 Java 小程序，客户端/服务器通信是必需的，至少在进行音频处理时，因为浏览器中的虚拟机不是足够快以处理音频（尽管可能是足以用于 MIDI 相关任务）。因此，Studio On-Line 项目中的 applet 仅被使用用于与服务器的接口。自从沟通以来在互联网上复杂得多而不是硬件驱动的客户端/服务器系统，智能通信协议是必要的。此外，由于 Java 语言用于 applet 是面向对象的，我们需要客户端/服务器的无缝集成沟通成为面向对象编程范例。与此相反，已经有了存在经典的客户端/服务器技术基于程序规划范式：SUN 的远程过程调用（RPC）用于其网络文件系统（NFS）。第一个商业可用的分布式对象之一系统是分布式对象下一步。最初是基于 NeXT 的运作系统和 Objective-C，NeXTSTEP 的分布式对象现在已被移植到其他 UNIX 系统。但是由于其平台限制和 NeXT 公司的历史，他们的分布式对象系统还没有成为突出的一个在市场上，还不清楚是否或者 Rhapsody 不会改变这种情况。更多必须找到一般和便携式解决方案因此 CORBA 系统已经成为一个行业标准。

CORBA，通用对象请求经纪人架构，是与语言和操作系统无关的分布式对象系统基于组件模型（完整的）规范，在所谓的语言映射中，特定编程的集成语言被定义。目前有映射对于 C++，Smalltalk，Java，甚至 C 一个 CORBA 系统是经纪人，可以是被视为客车（Mowbray 和 Malveau1997）。几家公司正在提供用于所有类型操作系统的 CORBA 代理像 UNIX，Windows 95，Windows NT，NeXTSTEP 等;甚至是公共领域的经纪人在互联网上可用。而 CORBA 是广泛的用于工业应用(Mowbray 和 Zahavi 1995)，

被许多公司接受，唯一的竞争对手是微软的 DCOM 系统。虽然 DCOM 有相当一段时间的限制到 C ++，微软现在还包括 Java 他们的 DCOM 系统。

SUN 本身也发展了一个用于他们的分布式对象系统 Java 语言：Java 远程消息接口 (RMI)。但是，这个系统是有限的仅 Java，并且不会在其他编程中可用语言。目前正在努力将 Java RMI 合并到 CORBA 系统中通过采用 CORBA 的 Java RMI IIOP 协议。用于小程序的分布式对象系统必须满足一定的要求。首先，他们必须在 Java 中实现。第二，他们有效地通过互联网进行通信。第三，他们必须提供自动错误处理出现连接丢失等问题经常在互联网上。最后，但不是至少，协议必须被转运防火墙。只有 CORBA 系统可以满足所有的要求；因此我们基于 Studio On-Line 在 CORBA 上作为副作用，我们获得了灵活性自己写的我们的服务器在 Java 和 C ++CORBA 的语言独立性可以互相沟通。对于工作室在线，最重要的要求是跨防火墙进行通信的能力。我们意识到这通过使用 Visigenic 的守门员程序，作为一种代理人，如有必要，编码 CORBA 特定的通信协议 IIOP 在使用的标准 HTTP 协议中用于 HTML 网页。这种技术被称为隧道，并可以与 Studio 工作在线在世界任何地方。应该提出一些最后的意见两种技术称为中间件和组件架构前者只是一个新的适用于所有提供应用的技术程序员具有更高级别的应用导向 API 到低级协议。中间件创建从底层运输抽象协议。突出的例子包括各种数据库访问中间件层等 Microsoft 的 ODBC 或 SUN 的 JDBC。在线工作室还使用中间件访问元数据在我们的数据库。工作室在线主要是 Javabased；因此我们使用 JDBC 系统，因为它是数据库独立于专有权 Oracle 提供的 API。提到的第二个技术是组件架构范例。申请转移后发展到面向对象架构，很明显，虽然解决了传统程序设计的许多问题，然而，我们收购了新的问题。其中大部分是由于巨大的大型面向对象系统的复杂性。

一这种复杂性的原因是依赖的数量在对象接口之间，可以成为难以管理一个解决方案是组合对象类被称为功能子系统组件。这些基于组件的架构最终导致了复合文件的想法，即可以包含的文件容器任何类型的应用程序组件绑在一起在通用框架内。虽然苹果的这样一个名为 OpenDoc 的结构 的提案死了 相对较快，由于其巨大的复杂性，Microsoft 以其组件架构为基础的 ActiveX 系统模型。甚至 SUN 设计了一个组件和复合文档架构他们的 Java 语言叫做 JavaBeans。问题，解决方案和未来工作的工作室在线工作室最大的技术挑战在线项目已经被提到：大多数人都可以使用网络速度音频应用程序缓慢虽然这可以解决消费者音频与智能音频压缩，这不能是专业的解决方案电脑要求的音质音乐。但是，网络速度瓶颈未来可能会因为快速的需求而改变网络最终由业界回答。另一个技

术问题越来越不兼容的浏览器，作为最近的讨论 Netscape 与微软展示。另外，由于商业压力很大释放他们的更快更新的版本浏览器，新技术并不奇怪商业上有时候没有完成，经常是不稳定的，而且还在变化设计。最终，Studio On-Line 的决定使用 applet 作为客户端用户界面可能对于没有接受过这种培训的用户造成问题技术。大多数用户了解简单的疑难方法，如重启电脑当系统崩溃时；但是，他们是不习惯特殊类型的问题这可能会在使用 applet 时发生。客户端问题的一个可能的解决方案是客户端可扩展性的想法。我们需要更简单的方法访问 Studio On-Line 的基本服务使用高端小程序技术。其实我们需要提供 Studio 在线的一些部分用户习惯的纯动态 HTML 并且没有浏览器兼容性问题已经提到的 servlet 方法可以实现这一点。

与 CGI 相比动态 HTML，启动一个小程序对于每个用户的请求，servlet 都是永久运行 Web-Server 扩展在 Java 中。对于每个用户的请求，只有一个调用 servlet 对象中的方法。以来 servlet 是持久服务器，用户的问题从页到页存储持久数据很容易解决。Servlet 被编程 Java，这使得我们很容易重用我们的 Java 和 C++ 服务器通过 CORBA。另外，一个直接的数据库可以通过 JDBC 进行连接小服务因此，可以提供两者各种客户端界面 - 小程序和动态 HTML - 同时使用相同的服务器应用程序。未来，Studio On-Line 必须集成更多的 IRCAM 的声音处理应用程序。这里的挑战是简化复杂 GUI，以便通过它们有效地实现它们小程序 Envisaged 是提供的应用程序非实时访问 IRCAM 的 3-D 音频系统，Spatialisateur (Jot and Warusfel 1995)，运行 FTS，但提供了一个自定义的用户界面。另外，正弦建模工具 HMM (Depalle, Garcia, 和 Rodet 1993)，基于隐马尔科夫模型，可能被集成到系统中。适应现有的建筑也被设想用于教育实验框架中的应用对新的国内服务。

致谢

非常感谢 Joseph Butch Rovin 的修改到原文。