《多媒体技术》实验报告

|  |  |
| --- | --- |
| 成员1： | 2220213301 邹康 软件1班 70% |
| 成员2： | 2220214175 姚熹文 软件1班 30% |

1. 实验选题

大作业选题1：媒体文件的压缩（图像的压缩）  
大作业选题2：多媒体系统的制作（图片压缩程序的Web网站）

基于图片压缩算法理解实现压缩网站多媒体系统

1. 实验要求

* 设计一个应用系统，实现对媒体文件的压缩
* 可以选择无损压缩或有损压缩，给出压缩比结果。
* 除了可以实现和测试教材上的压缩算法外（2种以上），必须体现对于该类压缩算法的自我设计和创新，或者学习和实现一种教材之外的压缩算法。
* 可以以网站等形式集成；
* 主题明确、积极向上

1. 实验内容
2. **平台选择**

**开发工具**

* 前端：vscode
* 后端：JetBrains WebStorm

**前端**

* Vue
* Vue-CLI 脚手架
* vuex 状态管理
* Axios 请求库
* ant Design vue组件库
* 前端工程化：ESLint + Prettier + JavaScript

**后端**

* Java Spring Boot 开发框架
* junit 测试工具

原因：

Java提供丰富的图像处理库和强大的数据处理能力，能更好的处理和实现算法。

Spring Boot简化了后端服务的开发和部署。

Vue.js灵活且易于与Java后端集成，适合构建动态交互界面，相对与直接原生h5c3和js组合，使用组件化模块工程化，更好展现多媒体系统。

1. **处理技术介绍**（如果是算法设计，需要给出算法的基本思想介绍；如果是媒体处理，需要给出处理的基本思路）
2. 前端Web界面设计：通过Vuex-Cli脚手架初始化项目，使用Vuex来存储全局数据，使用layout布局来设计界面，采用header，content，footer模块。  
    在header模块设计头部导航栏（包含icon，各个功能模块）作为二级路由连接入口；  
    在content模块进行压缩程序展示设计进行实现，里面包括文件上传模块，压缩设置模块，算法动态展示模块，压缩结果文件展示下载模块。  
    1. 文件上传，引用ant Design vue组件的upload组件，通过axios进行 异步处理，通过向后端服务器发送http请求，上传文件到后端，然后展示 出上传需要压缩的文件。  
    2. 压缩设置模块用于选择压缩算法  
    3. 算法动态展示模块,用于对压缩算法感兴趣的进行动态类动画展示算法 实现过程  
    4. 压缩结果文件展示下载模块，从后端服务器请求获取压缩的数据，动 态展示压缩后的文件，压缩比，提供文件下载链接。  
    在footer模块进行版权展示。
3. 后端压缩算法处理：采用Spring Boot快速构建框架，引入junit工具来测试算法正确性。  
    1. Controller 接口开发：接收前端客户端请求处理

2.服务开发：编写 Service 接口和实现类，完善业务逻辑

1. 图像处理及Huffman编码算法实现  
    1. 图像读取和预处理  
    使用Java的BufferedImage类和ImageIO库来读取图像文件，获取图像的像素数据。然后为了进行Huffman，采用了灰度图像压缩，将彩色图像转换为灰度，将RGB值转化为灰度值，根据每个像素的灰度值编码成字符序列。

2.频率统计   
 通过遍历图像的每一个像素，统计每个灰度值的出现频率。这一步是霍夫曼编码的前提，因为霍夫曼编码依赖于频率来构建最优前缀编码。频率数据存储在一个HashMap中，其中键是灰度值，值是该灰度值出现的次数。  
3.Huffman树的构建  
 使用频率统计数据，构建霍夫曼树。使用一个优先队列（PriorityQueue），根据频率的升序排列节点。每次从队列中取出两个频率最低的节点，创建一个新节点作为它们的父节点，其频率是两个子节点频率之和，然后将新节点重新加入优先队列。重复这个过程，直到队列中只剩一个节点，这个节点就是霍夫曼树的根节点  
4.哈夫曼编码及编码图像数据生成  
 从霍夫曼树中生成霍夫曼编码。这一步涉及从根节点开始遍历树，到达每个叶节点时，记录从根到叶的路径（左子节点表示0，右子节点表示1）。这样，每个叶节点的路径就是对应灰度值的霍夫曼编码。根据哈夫曼编码表，将图像中的每个像素灰度值转换为对应的霍夫曼编码，形成一个长的编码字符串。  
5.数据写入  
压缩后的文件的扩展名为.hfmc ，压缩后文件结构为：

* + 文件头（54字节）：原图像相关数据
  + 编码后图像数据总位数dataBitCount（4字节，即一个32为无符号整数）
  + 原图像文件中不同的数据单元的数目（不同灰度值的数量）weightMapValCount（4字节，即一个32为无符号整数）
  + 权重映射表（表中每一项包含1个字节的字符（灰度值）本身和4个字节即32位无符号整数表示的该字符（灰度值）的频率，一共5个字节，表的总大小为weightMapValCount \* 5）
  + 编码数据，所有像素的霍夫曼编码转换为位流后的数据
  + 实际的编码数据位数为dataBitCount，最后一个字节的数据如果是不满8位，余下的位用0作为padding

1. 图像处理及JAVA自带库压缩算法实现

1.读取图像：使用 ImageIO.read 方法从输入文件中读取图像并生成 BufferedImage 对象。这一步将图像数据加载到内存中，便于后续处理。

2. 获取JPEG图片写入器：通过 ImageIO.getImageWritersByFormatName("jpg").next() 获取一个支持JPEG格式的图片写入器 ImageWriter。Java中有多个写入器，使用这个方法可以确保我们获得JPEG的写入器。

3.配置图像写入参数:创建 ImageWriteParam 对象并通过 writer.getDefaultWriteParam() 获取默认的写入参数。

4.设置压缩模式为显式模式,设置压缩质量：param.setCompressionQuality(quality)，其中 quality 参数的取值范围为0.0f到1.0f，分别表示最低和最高质量。通过设置压缩质量，可以控制输出文件的大小和图像质量之间的平衡,我们选择0.5f

5.JPEG介绍

（1） 颜色空间转换：将图像从RGB颜色空间转换到YCrCb颜色空间，其中Y代表亮度，Cr和Cb代表色度。这一步的目的是将图像数据分离为视觉上更重要的亮度信息和较不重要的色度信息。

（2）分块处理：将图像分成8x8像素的块。每个块分别处理，减少计算复杂度。

（3）离散余弦变换（DCT）：对每个8x8块进行离散余弦变换，将空间域的数据转换为频域数据。DCT可以将图像块中的大部分信息集中到较少的几个频率成分中。

（4）量化：对DCT系数进行量化。根据人的视觉系统对高频和低频成分的敏感度不同，使用量化表将DCT系数除以不同的量化因子，从而减少高频成分的精度。这一步是JPEG压缩中最主要的有损部分。

java中的 ImageIO 和 ImageWriteParam 类允许我们调整JPEG图像的压缩质量。这里的压缩实现主要利用了JPEG标准的量化和熵编码部分

1. **本实验的设计性和创新性体现**（如果是算法设计，说明算法的改进和创新之处及其依据；如果是处理技术，说明新颖的处理效果，及其对应的方法介绍）
2. 所有html界面，程序都属于自己设计编写，界面设计符合人们常用习惯。采用vue来实现页面，vue的响应式系统可以自动追踪依赖和管理状态更新，当数据变化时自动更新视图，无需手动介入，简化了数据管理并提高了界面的响应性。他的组件化可以复用，简化代码，让界面更为简洁。
3. 引入图像预处理步骤，我尝试过使用整个像素点来作为一个字符，也尝试过将每个像素RGB分解成三个色分别作为一个字符，最终决定将图像转换为灰度图，因为灰度图只有一个颜色通道，而彩色图像有三个颜色通道（红、绿、蓝）。灰度图像的数据量仅为彩色图像的三分之一，处理和分析的复杂度会降低。灰度会让频率更为集中，有利于huffman实现。
4. hffman中使用优先队列的优化算法，提升霍夫曼树的构建效率（最优二进制树）
5. 在前端展示中，通过动画动态演示编码过程，直观展示压缩原理。
6. **实现步骤**（如果是代码实现，给出关键功能模块的代码；如果是处理技术，给出处理步骤）

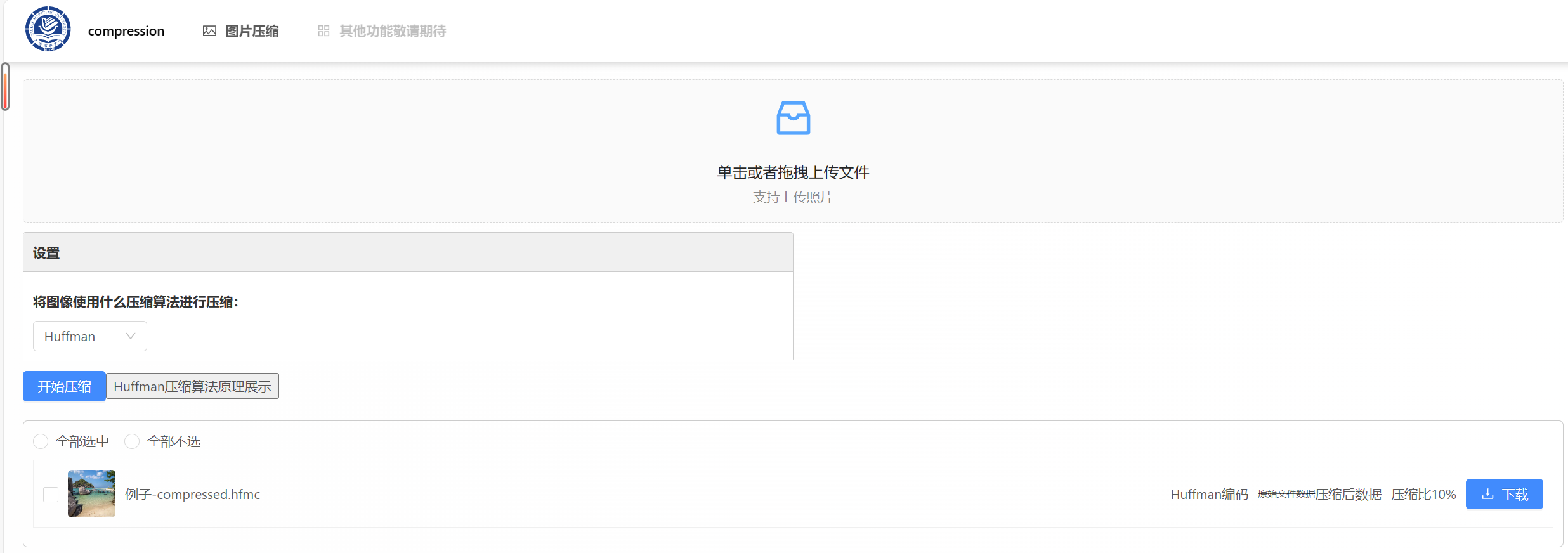
**后端主要模块**

1. //文件上传与压缩控制器
2. @RestController
3. @RequestMapping("/api/upload")
4. **public** **class** FileUploadController {
5. @PostMapping
6. **public** ResponseEntity<String> uploadFile(@RequestParam("file") MultipartFile file) {
7. // ... 代码实现
8. }
10. @PostMapping("/compress")
11. **public** ResponseEntity<Map<String, Object>> compressFile(@RequestBody Map<String, Object> request) {
12. // ... 代码实现
13. }
14. }
15. //Huffman
16. **public** **class** HuffmanCompression **implements** CompressionAlgorithm {
17. @Override
18. **public** Map<String, Object> compress(String inputFilePath, String outputFilePath) **throws** Exception{
19. //代码实现
20. }
21. // 生成霍夫曼编码
22. **private** Map<Character, String> generateHuffmanCodes(String input) {
23. }
24. // 霍夫曼树节点类
25. **private** **static** **class** HFNode **implements** Comparable<HFNode> {}}
26. //jpeg
27. **public** **class** DCTCompression **implements** CompressionAlgorithm {
29. @Override
30. **public** Map<String, Object> compress(String inputFilePath, String outputFilePath) **throws** Exception {
31. //调用库设置参数，压缩
32. }}

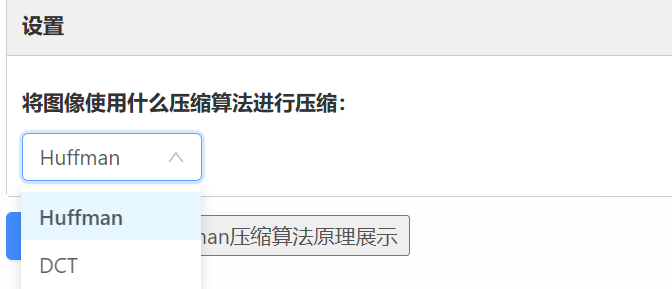
**前端主要模块**

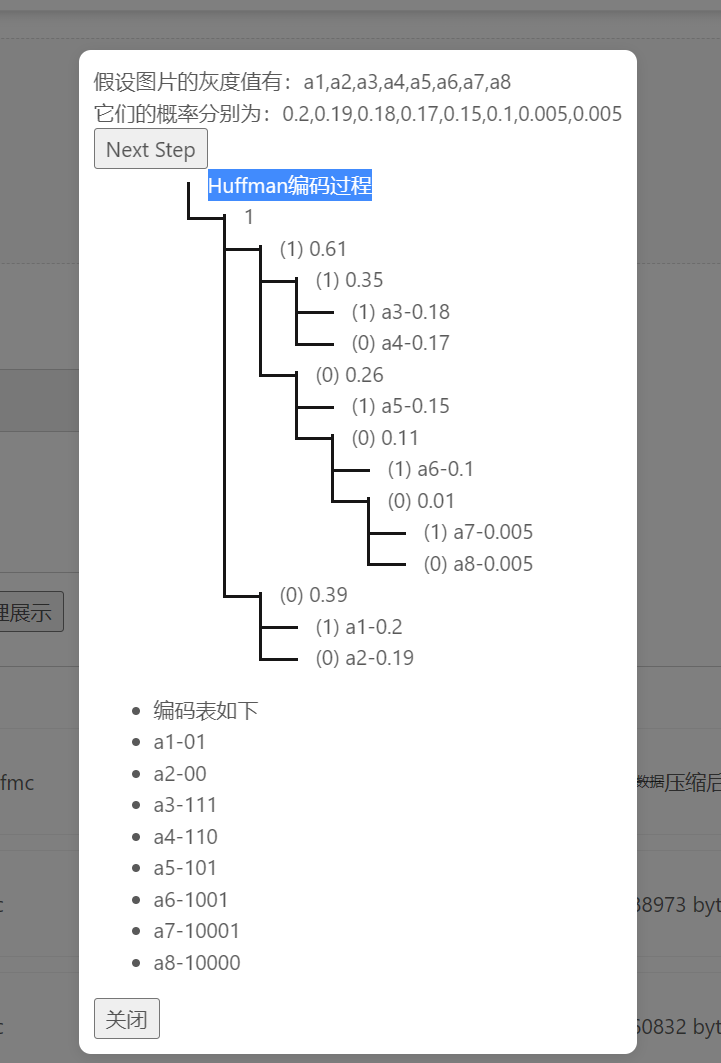
1. **<template>**
2. **<div** id="basic-layout"**>**
3. **<a-layout>**
4. **<a-layout-header><GlobalHeader** **/></a-layout-header>**
5. **<a-layout-content><ImageCompression/></a-layout-content>**
6. **<a-layout-footer>**本项目由zaki制作**</a-layout-footer>**
7. **</a-layout>**
9. **</div>**
10. **</template>**
12. **<script>**
13. import GlobalHeader from '../components/GlobalHeader.vue'
14. import ImageCompression from '../components/ImageCompression.vue'
15. export default {
16. components: {
17. GlobalHeader,
18. ImageCompression
19. }
20. }
21. **</script>**
23. **<style** scoped **>**
24. #basic-layout {
25. height: 100vh;
26. }
28. .ant-layout-header {
29. margin-bottom: 16px;
30. background: white;
31. box-shadow: #eee 1px 1px 5px;
32. position: fixed;
33. top: 0;
34. left: 0;
35. right: 0;
36. padding: 0 ;
37. z-index: 999;
38. }
40. .ant-layout-content{
41. margin-top: 64px;
42. padding: 20px;
43. background: linear-gradient(to right, #fefefe, #fff);
44. }
46. .ant-layout {
47. width: 100%;
48. min-height: 100%;
49. }
51. .ant-layout-footer {
52. background-color: #efefef;
53. padding: 16px;
54. width: 100%;
55. text-align:center;
56. }
57. **</style>**
58. 实验效果

1.主界面



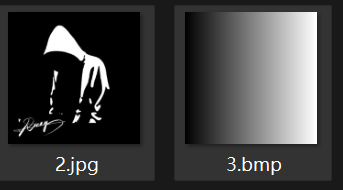
1. 其他功能





1. 压缩效果

Huffman

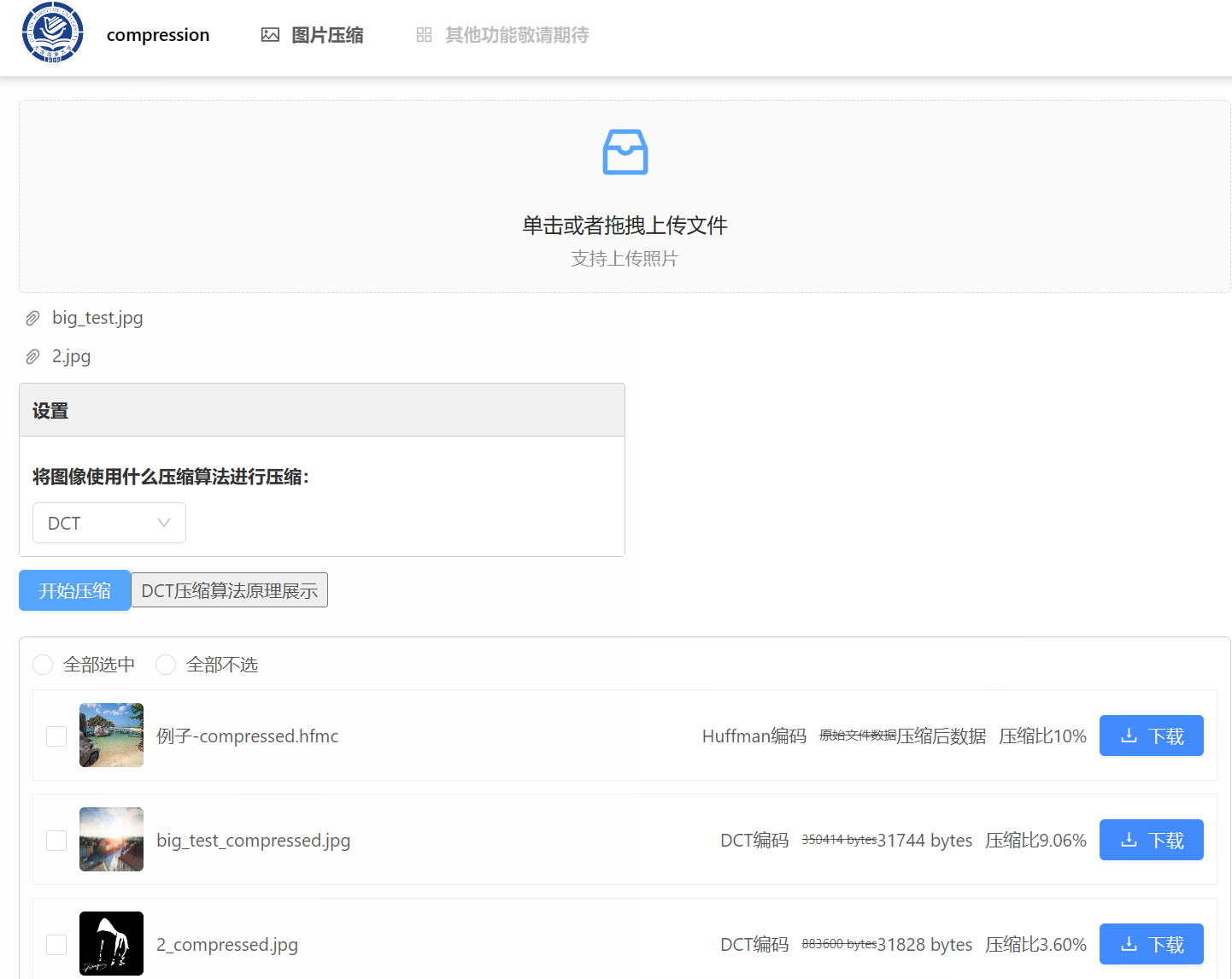
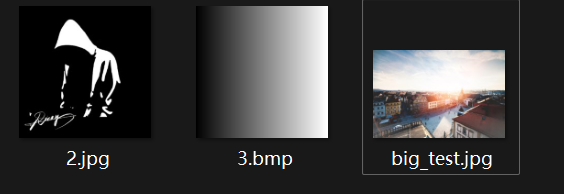




对灰度值少的分布不均匀的图像，权值映射表较小，压缩效果较好

对灰度值多的分布均匀的图像，权值映射表较大，压缩效果较差

java包DCT



现代这种新型组合式压缩，压缩大小很明显

1. 实验结论

通过本次实验，我们成功设计并实现了一个基于图片压缩算法的多媒体系统。该系统集成了无损压缩和有损压缩两种方法，具体实现了Huffman编码和JPEG压缩算法。前端采用Vue.js构建，后端基于Spring Boot框架，实现了前后端分离的动态交互界面，提供了文件上传、压缩设置、算法动态展示和压缩结果展示下载等功能模块。

实验结果表明：

Huffman编码 在灰度图像压缩中具有较高的压缩比，适用于对图像质量要求较高的场景。

JPEG压缩 则在图像质量和压缩比之间取得了较好的平衡，通过调整压缩质量参数，可以灵活地控制图像的质量和文件大小。

系统界面友好，操作便捷，用户可以直观地了解图像压缩过程及其效果。该系统的设计和实现，不仅验证了多种压缩算法的有效性，也展现了前后端技术的结合在实际应用中的优势。