基于 Java Servlet 构建的图片服务器

背景需求

图床, 解决 github / 博客中插入图片的问题.

给定一个 url, 能够获得到图片内容.

可以参考搜狗图片.

重要知识点

1. 简单的Web服务器设计能力
2. Java 操作 MySQL 数据库
3. 数据库设计
4. Restful 风格 API
5. gson 的使用
6. 强化 HTTP 协议的理解
7. Servlet 的使用
8. 基于 md5 进行校验
9. Postman 工具的使用
10. 软件测试的基本思想和方法

整体架构

核心就是一个 HTTP 服务器, 提供对图片的增删改查能力.

同时搭配简单的页面辅助完成图片上传/展示.

数据库设计

创建文件 db.sql

```
create database if not exists image_system;
use image_system;

drop table if exists `image_table`
create table `image_table`(image_id int not null primary key auto_increment,
                           image_name varchar(50),
                           size bigint,
                           upload_time varchar(50),
                           md5 varchar(128),
                           content_type varchar(50) comment '图片类型',
                           path varchar(1024) comment '图片所在路径')
```

可以通过 `mysql -uroot < db.sql` 来导入 sql 内容.

> 什么是 md5?
>
> 这是一种常见字符串 hash 算法, 具有三个特性:
>
> 1. 不管源字符串多长, 得到的最终 md5 值都是固定长度
> 2. 源字符串稍微变化一点点内容, md5 值会变化很大(降低冲突概率)
> 3. 通过原字符串很容易计算得到 md5 值, 但是根据 md5 推导出原字符串很难(几乎不可能).

服务器 API 设计

Json 是一种常见是数据格式组织方式. 源于 JavaScript, 是一种键值对风格的数据格式.

Java 中可以使用 Gson 库来完成 Json 的解析和构造.

在 Maven 中新增 Gson 的依赖

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.2</version>
</dependency>
```

简单示例 (创建一个 TestGson 类)

```
public class TestGson {
    public static void main(String[] args) {
        HashMap<String, Object> data = new HashMap<>();
        data.put("name", "曹操");
        data.put("skill1", "剑气");
        data.put("skill2", "三段跳");
        data.put("skill3", "吸血加攻击");
        data.put("skill4", "释放技能加攻速");
        Gson gson = new GsonBuilder().create();
        String jsonData = gson.toJson(data);
        System.out.println(jsonData);
    }
}
```

新增图片

写一个简单的 html 来上传图片

upload.html (参考 https://www.jianshu.com/p/7636d5c60a8d)

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
</head>
<body>
    <form method="POST" enctype="multipart/form-data" action="/java_image_server/image">
        <table>
            <tr>
                <td>文件上传:</td>
                <td><input type="file" name="filename"/></td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="上传"/></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

直接抓个包看看, 上传图片的 http 请求是啥样的.

> 抓包需要使用 IDEA 内置的服务器来展示 html 才能抓到.
>
> 也就是点击 html 文件右上角的浏览器图标.

设定接口如下.

请求:

```
POST /image
Content-Type: application/x-www-form-urlencoded

------WebKitFormBoundary5muoelvEmAAVUyQB
Content-Disposition: form-data; name="filename"; filename="图标.jpg"
Content-Type: image/jpeg

......[图片正文].....

响应:
HTTP/1.1 200 OK
{
    "ok": true,
}
```

查看所有图片元信息

```
请求:
GET /image

HTTP/1.1 200 OK
[
    {
        "imageId": 1,
        "imageName": "1.png",
        "contentType": "image/png",
        "md5": "[md5值]"
    }
]
```

查看指定图片元信息

```
请求:
GET /image?imageId=1

响应:
HTTP/1.1 200 OK
{
    "imageId": 1,
    "imageName": "1.png",
    "contentType": "image/png",
    "md5": "[md5值]"
}
```

删除图片

```
请求:
DELETE /image?imageId=1

响应:
HTTP/1.1 200 OK
{
    "ok": true
}
```

查看图片内容

```
请求:
GET /imageShow?imageId=1

响应:
HTTP/1.1 200 OK
content-type: image/png

[响应 body 中为 图片内容 数据]
```

创建一个 JavaWeb 项目

过程略(项目最好提前准备好).

封装数据库操作

dao 对应到 MVC 中的 Model 层. 是对数据库的封装.

创建包结构

```
com.bit.java_image_server.dao
```

创建 DBUtil 类

创建一个单例类辅助创建连接.

其中 URL 为数据库链接字符串. 用户名, 密码都是固定的.

```
private static final String URL = "jdbc:mysql://127.0.0.1:3306/image_server?
characterEncoding=utf8&useSSL=true"
private static final String USERNAME = "root";
private static final String PASSWORD = "";
```

这个类主要包含三个方法

```
// 这是一个获取单例的方法
public static DataSource getDataSource() { }
// 获取链接
public static Connection getConnection() { }
// 关闭链接
public static void close(Connection conn, PreparedStatement statement,
                         ResultSet resultSet) {}
```

类的实现代码

```
public class DBUtil {
    private static final String URL = "jdbc:mysql://127.0.0.1:3306/image_server?
characterEncoding=utf8&useSSL=true";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "";

    private static volatile DataSource dataSource = null;

    public static DataSource getDataSource() {
        if (dataSource == null) {
            // 此处的 synchronized 其实就是给当前这一小块加锁.
            // synchronized 要么修饰一个方法, 要么得传个对象. 此处修饰整个方法显然不合适(粒度太大)
            synchronized (DBUtil.class) {
                if (dataSource == null) {
                    dataSource = new MysqlDataSource();
                    ((MysqlDataSource)dataSource).setUrl(URL);
                    ((MysqlDataSource)dataSource).setUser(USERNAME);
                    ((MysqlDataSource)dataSource).setPassword(PASSWORD);
                }
            }
        }
        return dataSource;
    }

    // 每次收到请求重新获取数据库链接. DataSource 内置连接池, 所以效率也还行.
    // 后面的 close 也不是真的销毁链接. 只是回收到池子里了, 后面还能用
    public static Connection getConnection() {
        try {
            return getDataSource().getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    public static void close(Connection conn, PreparedStatement statement,
                             ResultSet resultSet) {
        // 关闭顺序必须是先关 resultSet, 再关 statement, 最后关 conn
        try {
            if (resultSet != null) {
                resultSet.close();
            }
```

```
            if (statement != null) {
                statement.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

创建 Image 类

```
public class Image {
    private int imageId;
    private String imageName;
    private int size;
    private String uploadTime;
    private String md5;
    private String contentType;
    private String path;
}
```

创建 ImageDao 类

```
public class ImageDao {
    public boolean insert(Image image) {
        return false;
    }

    public Image[] selectAll() {
        return null;
    }

    public Image selectOne(int imageId) {
        return null;
    }

    public boolean delete(int imageId) {
        return false;
    }
}
```

实现 ImageDao.insert 方法

```
public boolean insert(Image image) {
    // 1. 获取连接
    Connection connection = DBUtil.getConnection();
    PreparedStatement statement = null;
    String sql = "insert into image_table values(null, ?, ?, ?, ?, ?, ?)";
    try {
        // 2. 拼装 PreparedStatement
        statement.setString(1, image.getImageName());
        statement.setInt(2, image.getSize());
        statement.setString(3, image.getUploadTime());
        statement.setString(4, image.getMd5());
        statement.setString(5, image.getContentType());
        statement.setString(6, image.getPath());
        System.out.println(statement.toString());

        // 3. 执行语句，返回值表示影响了几行表格
        int ret = statement.executeUpdate();
        return ret > 0;
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtil.close(connection, statement, null);
    }
    return false;
}
```

测试一下 insert 方法

```
public static void main(String[] args) {
    // 测试 ImageDao 的代码
    Image image = new Image("测试.png", 100, "2019/12/03", "AABBCCDD",
                            "image/png", "./data/测试.png");
    ImageDao imageDao = new ImageDao();
    boolean ret = imageDao.insert(image);
    System.out.println(ret);
}
```

打一个 jar 包上传到 Linux 服务器上, 并运行

> 注意, 打 jar 包的时候要把 MF 文件生成到项目的根目录, 否则 jar 包可能存在问题.

实现 ImageDao.selectAll

```
public ArrayList<Image> selectAll() {
    // 1. 获取数据库链接
    Connection connection = DBUtil.getConnection();
    PreparedStatement statement = null;
    String sql = "select * from image_table";
    ArrayList<Image> result = new ArrayList<>();
    ResultSet resultSet = null;
```

```
    try {
        // 2. 执行 sql，并获取结果集合
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();
        // 3. 遍历结果集合
        while (resultSet.next()) {
            Image image = new Image();
            image.setImageId(resultSet.getInt("image_id"));
            image.setImageName(resultSet.getString("image_name"));
            image.setSize(resultSet.getInt("size"));
            image.setUploadTime(resultSet.getString("upload_time"));
            image.setMd5(resultSet.getString("md5"));
            image.setContentType(resultSet.getString("content_type"));
            image.setPath(resultSet.getString("path"));

            result.add(image);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtil.close(connection, statement, resultSet);
    }
    return result;
}
```

测试一下 selectAll 方法

```
public static void main(String[] args) {
    ImageDao imageDao = new ImageDao();
    ArrayList<Image> images = imageDao.selectAll();
    System.out.println(images.toString());
}
```

实现 ImageDao.selectOne

```
public Image selectOne(int imageId) {
    // 1. 获取数据库连接
    Connection connection = DBUtil.getConnection();
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String sql = "select * from image_table where image_id = ?";
    try {
        // 2. 执行 SQL 语句
        statement = connection.prepareStatement(sql);
        statement.setInt(1, imageId);
        resultSet = statement.executeQuery();

        // 3. 遍历结果集合(这个结果中应该只有一个)
        if (resultSet.next()) {
            Image image = new Image();
```

```
            image.setImageId(resultSet.getInt("image_id"));
            image.setImageName(resultSet.getString("image_name"));
            image.setSize(resultSet.getInt("size"));
            image.setUploadTime(resultSet.getString("upload_time"));
            image.setMd5(resultSet.getString("md5"));
            image.setContentType(resultSet.getString("content_type"));
            image.setPath(resultSet.getString("path"));
            return image;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtil.close(connection, statement, resultSet);
    }
    return null;
}
```

测试一下 selectOne

```
public static void main(String[] args) {
    ImageDao imageDao = new ImageDao();
    Image image = imageDao.selectOne(2);
    System.out.println(image);
}
```

实现 ImageDao.delete

```
public boolean delete(int imageId) {
    // 1. 创建数据库连接
    Connection connection = DBUtil.getConnection();
    PreparedStatement statement = null;
    String sql = "delete from image_table where image_id = ?";
    try {
        // 2. 执行 SQL 语句
        statement = connection.prepareStatement(sql);
        statement.setInt(1, imageId);
        int ret = statement.executeUpdate();
        return ret > 0;
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtil.close(connection, statement, null);
    }
    return false;
}
```

测试 delete 方法

```
public static void main(String[] args) {
    ImageDao imageDao = new ImageDao();
    boolean ret = imageDao.delete(2);
    System.out.println(ret);
}
```

实现 Servlet

首先在项目根目录下创建一个 servlet 包

在这个包中创建两个 Servlet 类.

一个用来完成图片的增删改查(ImageServlet), 一个用来展示图片的详细内容(ImageShowServlet)

创建 ImageServlet

这个类的 doPost 对应插入图片, doGet 对应查看图片信息, doDelete 对应删除图片.

```
public class ImageServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        super.doGet(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        super.doPost(req, resp);
    }

    @Override
    protected void doDelete(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        super.doDelete(req, resp);
    }
}
```

同时要记得把这个类加到 web.xml 中.

注意, 其中的类名要写完整的带包名的名字.

```
<servlet>
    <servlet-name>ImageServlet</servlet-name>
    <servlet-class>com.bit.java_image_server.servlet.ImageServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ImageServlet</servlet-name>
    <url-pattern>/image</url-pattern>
</servlet-mapping>
```

使用 Alibaba Cloud Toolkit 插件把项目部署到云服务器上

将刚才的代码简单修改成一个 hello world 程序, 然后部署到云服务器上测试一下.

参考 https://blog.csdn.net/a455368951/article/details/102037508

注意, 部署过程需要有一定时间. tomcat 解压缩 war 包也需要一定时间. 稍等片刻才能操作, 否则就会 404

实现 ImageServlet.doPost

这个方法对应上传图片

这里需要用到 Commons FileUpload, 可以在 Maven 仓库中找到这个包, 并且使用 maven 下载之.

https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload/1.4

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.4</version>
</dependency>
```

实现 doPost 方法

实现上传图片过程可以参考 https://blog.csdn.net/weixin_38437243/article/details/78871242

```
// 1. 获取到图片相关的元信息(Image对象), 并写入数据库
//  a) 创建 factory 对象和 upload 对象
//  b) 使用 upload 对象解析请求
//  c) 对请求信息进行解析, 转换成 Image 对象
//  d) 将 Image 对象写入数据库中
// 2. 获取到图片内容, 写入到磁盘中
// 3. 设置返回的响应结果
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    // 上传图片
    resp.setContentType("application/json; charset=utf8");
    PrintWriter writer = resp.getWriter();
    // 1. 获取到图片相关的元信息(Image对象), 并写入数据库
```

```
    //  a) 创建 factory 对象和 upload 对象
    FileItemFactory factory = new DiskFileItemFactory();
    ServletFileUpload upload = new ServletFileUpload(factory);
    List<FileItem> items = null;
    //  b) 使用 upload 对象解析请求
    try {
        items = upload.parseRequest(req);
    } catch (FileUploadException e) {
        e.printStackTrace();
        resp.setStatus(500);
        writer.println("{ \"ok\" : false, \"reason\" : \"请求解析失败\"}");
        return;
    }
    //  c) 对请求信息进行解析，转换成 Image 对象
    //     一个请求可以上传一个文件，也能上传多个文件
    //     当前只考虑上传一个的情况
    FileItem item = items.get(0);
    Image image = new Image();
    image.setImageName(item.getName());
    image.setSize((int)item.getSize());
    SimpleDateFormat df = new SimpleDateFormat("yy-MM-dd HH:mm:ss");
    image.setUploadTime(df.format(new Date()));
    // TODO 计算 MD5 的方法暂时不写
    image.setMd5("AABBCCDD");
    image.setContentType(item.getContentType());
    // 引入时间戳，让文件名唯一
    image.setPath(PATH_BASE + System.currentTimeMillis() + "_" + image.getImageName());
    //  d) 将 Image 对象写入数据库中
    ImageDao imageDao = new ImageDao();
    imageDao.insert(image);

    // 2. 获取到图片内容，写入到磁盘中
    File file = new File(image.getPath());
    try {
        item.write(file);
    } catch (Exception e) {
        e.printStackTrace();
        resp.setStatus(500);
        // 注意，此处可能出现脏数据的情况，比如数据库插入成功，但是这里文件写入失败
        writer.println("{ \"ok\" : false, \"reason\" : \"文件写入失败\"}");
        return;
    }

    // 3. 设置返回的响应结果
    resp.setStatus(200);
    // 先不使用 GSON 来构建 Json 字符串
    writer.println("{ \"ok\": true }");
}
```

验证该方法, 可以使用刚才的 upload.html, 上传一个图片试试, 检查服务器响应是否正确, 数据库是否写入成功, 图片文件是否上传成功.

实现 ImageServlet.doGet

这里要分成两种情况, 一个是获取所有图片信息, 一个是获取单个图片信息.

根据请求中是否带有 image_id 参数来决定

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    // 这个一定要放在上面(否则不生效)
    resp.setContentType("application/json;charset=UTF-8");

    String image_id = req.getParameter("image_id");
    if (image_id == null || image_id.equals("")) {
        // 获取所有图片信息
        doSelectAll(req, resp);
    } else {
        // 获取指定图片信息
        doSelectOne(Integer.parseInt(imageId), resp);
    }
}
```

实现 doSelectAll

```
// 1. 创建 ImageDao 对象并从数据库查找数据
// 2. 将查找到的数据转换成 JSON 格式的字符串
// 3. 结果字符串写入到 resp 对象中
private void doSelectAll(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
    ImageDao imageDao = new ImageDao();
    List<Image> images = imageDao.selectAll();
    String respString = gson.toJson(images);
    resp.setContentType("application/json; charset=utf8");
    resp.getWriter().write(respString);
}
```

验证 doSelectAll 需要使用 Postman 工具来构造一个 Get 请求.

实现 doSelectOne

需要考虑 id 不存在的情况.

```
// 1. 创建 ImageDao 对象并从数据库查找数据
// 2. 将查找到的数据转换成 JSON 格式的字符串
// 3. 结果字符串写入到 resp 对象中
private void doSelectOne(Integer imageId, HttpServletResponse resp) throws IOException {
    ImageDao imageDao = new ImageDao();
    Image image = imageDao.selectOne(imageId);
```

```
    if (image == null) {
        resp.setStatus(404);
        resp.getWriter().write("{ \"ok\": false, \"reason\":\"imageId 不存在\"}");
        return;
    }
    resp.setStatus(200);
    String respString = gson.toJson(image);
    resp.getWriter().write(respString);
}
```

验证过程同上.

实现 ImageServer.doDelete

```
// 1. 获取请求中的 imageId
// 2. 创建 ImageDao 对象, 查找对应的 Image 对象
// 3. 删除数据库中的数据
// 4. 删除磁盘上的文件
// 5. 写回响应
protected void doDelete(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    resp.setContentType("application/json; charset=utf8");
    // 1. 获取请求中的 imageId
    String imageId = req.getParameter("imageId");
    if (imageId == null || "".equals(imageId)) {
        resp.setStatus(404);
        resp.getWriter().write("{ \"ok\": false, \"reason\": \"imageId 解析失败\" }");
        return;
    }
    // 2. 创建 ImageDao 对象, 查找对应的 Image 对象
    ImageDao imageDao = new ImageDao();
    Image image = imageDao.selectOne(Integer.parseInt(imageId));
    if (image == null) {
        resp.setStatus(404);
        resp.getWriter().write("{ \"ok\": false, \"reason\": \"imageId 不存在\" }");
        return;
    }
    // 3. 删除数据库中的数据
    imageDao.delete(Integer.parseInt(imageId));
    // 4. 删除磁盘上的文件
    File file = new File(image.getPath());
    file.delete();
    // 5. 写回响应
    resp.getWriter().write("{ \"ok\": true }");
}
```

验证方式同上.

实现 ImageShowServlet

实现展示图片详细内容的接口.

```
// 1. 解析请求中的 imageId
// 2. 查找数据库，找到对应的 path
// 3. 从文件中读图片数据，别忘记设置 content-type
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    // 1. 解析请求中的 imageId
    String imageId = req.getParameter("imageId");
    if (imageId == null || "".equals(imageId)) {
        resp.setContentType("application/json; charset=utf8");
        resp.setStatus(404);
        resp.getWriter().write("{ \"ok\": false, \"reason\": \"解析 imageId 失败\" }");
        return;
    }
    // 2. 查找数据库，找到对应的 path
    ImageDao imageDao = new ImageDao();
    Image image = imageDao.selectOne(Integer.parseInt(imageId));
    if (image == null) {
        resp.setContentType("application/json; charset=utf8");
        resp.setStatus(404);
        resp.getWriter().write("{ \"ok\": false, \"reason\": \"imageId 不存在\" }");
        return;
    }
    // 3. 从文件中读图片数据，别忘记设置 content-type
    resp.setContentType(image.getContentType());
    File file = new File(image.getPath());
    // 使用字节流读取文件(图片是二进制文件，不适合使用字符流)
    FileInputStream fileInputStream = new FileInputStream(file);
    byte[] buffer = new byte[1024];
    OutputStream outputStream = resp.getOutputStream();
    while (true) {
        int len = fileInputStream.read(buffer);
        if (len == -1) {
            break;
        }
        outputStream.write(buffer);
    }
    fileInputStream.close();
    outputStream.close();
}
```

修改 web.xml

```
<servlet>
    <servlet-name>ImageShowServlet</servlet-name>
    <servlet-class>com.bit.java_image_server.api.ImageShowServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ImageShowServlet</servlet-name>
    <url-pattern>/imageShow</url-pattern>
</servlet-mapping>
```

验证程序

写前端页面

使用 HTML 模板

直接在百度上搜索 "免费网页模板", 能找到很多免费模板网站. 可以直接基于现成的漂亮的页面进行修改.

> tips: 做减法比做加法更容易.

将网页模板解压缩, 拷贝到项目的 webapp 目录中.

我们课堂上使用的模板是 amaze ui.

http://tpl.amazeui.org/

基于模板进行删减

基于模板中的 lw-img.html 进行修改, 删除不必要的部分.

修改之后的效果为:

修改之后 html 代码为(只截取部分关键代码):

导航栏实现上传

修改过程:

- 文件上传和提交按钮: 把 input 标签及其父 div 拷贝一份. 原来的 input 标签 type 改成 file, 增加 name="filename". 新的 input 标签 type="submit"
- 修改 form 标签属性, 新增 method="POST" enctype="multipart/form-data" action="/java_image_server/image"

```
<div class="am-collapse am-topbar-collapse" id="blog-collapse">
    <ul class="am-nav am-nav-pills am-topbar-nav">
    </ul>
    <form class="am-topbar-form am-topbar-right am-form-inline" method="POST"
enctype="multipart/form-data" action="/java_image_server/image">
        <div class="am-form-group">
            <input type="file" name="filename" class="am-form-field am-input-sm">
        </div>
        <div class="am-form-group">
            <input type="submit" class="am-form-field am-input-sm"  style="height:41px"
value="上传"/>
        </div>
    </form>
</div>
```

注意, 为了让 "上传按钮" 和前面一样高, 这里加了一个 `style="height:41px"`

> 此处我们直接使用 style 属性调整样式, 这是一个简单粗暴的做法. 事实上专业的前端开发很少这样做, 代码的可维护性不好~~.
>
> PS: 咱们目前不是专业前端开发, 所以先不管那么多, 先简单粗暴做出来效果就行~

页面主题展示图片预览

```
<div class="am-g am-g-fixed blog-fixed blog-content">
    <figure data-am-widget="figure" class="am am-figure am-figure-default "   data-am-
figure="{  pureview: 'true' }">
        <div id="container">
            <div><img src="images/01.jpg" style="height:200px; width:200px"><h3>Agfa</h3>
</div>
        </div>
    </figure>
</div>
```

类似的, 给 img 标签关联上高度和宽度.

使用 Vue.js

参考 Vue 的文档 https://cn.vuejs.org/v2/guide/

创建 Vue 对象

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

在页面中找个位置标注为 id="app", 演示插值表达式的基本用法.

并且使用控制台演示.

实现展示图片

构造数据

修改 js 代码

```
var app = new Vue({
    el:'#app',
    data: {
        images: [
          {
            imageId: 1,
            imageName: "1.png",
            contentType: "image/png",
            md5: "aabbccdd",
          },
          {
            imageId: 2,
            imageName: "2.png",
```

```
            contentType: "image/png",
            md5: "aabbccdd",
          }
        ]
    },
    methods: {

    },
});
```

修改 html 代码, 和数据关联

- 使用 v-bind:src 把图片的 src 通过 imageShow 接口获取到.
- 使用 {{image.imageName}} 表示图片标题

```
<div class="am-g am-g-fixed blog-fixed blog-content">
  <figure data-am-widget="figure" class="am am-figure am-figure-default "   data-am-figure="
{  pureview: 'true' }">
      <div id="container">
        <div v-for="image in images">
          <img v-bind:src=" 'imageShow?imageId=' + image.imageId "
style="height:200px;width:200px">
          <h3>{{image.imageName}}</h3>
        </div>
      </div>
  </figure>
</div>
```

从服务器获取数据

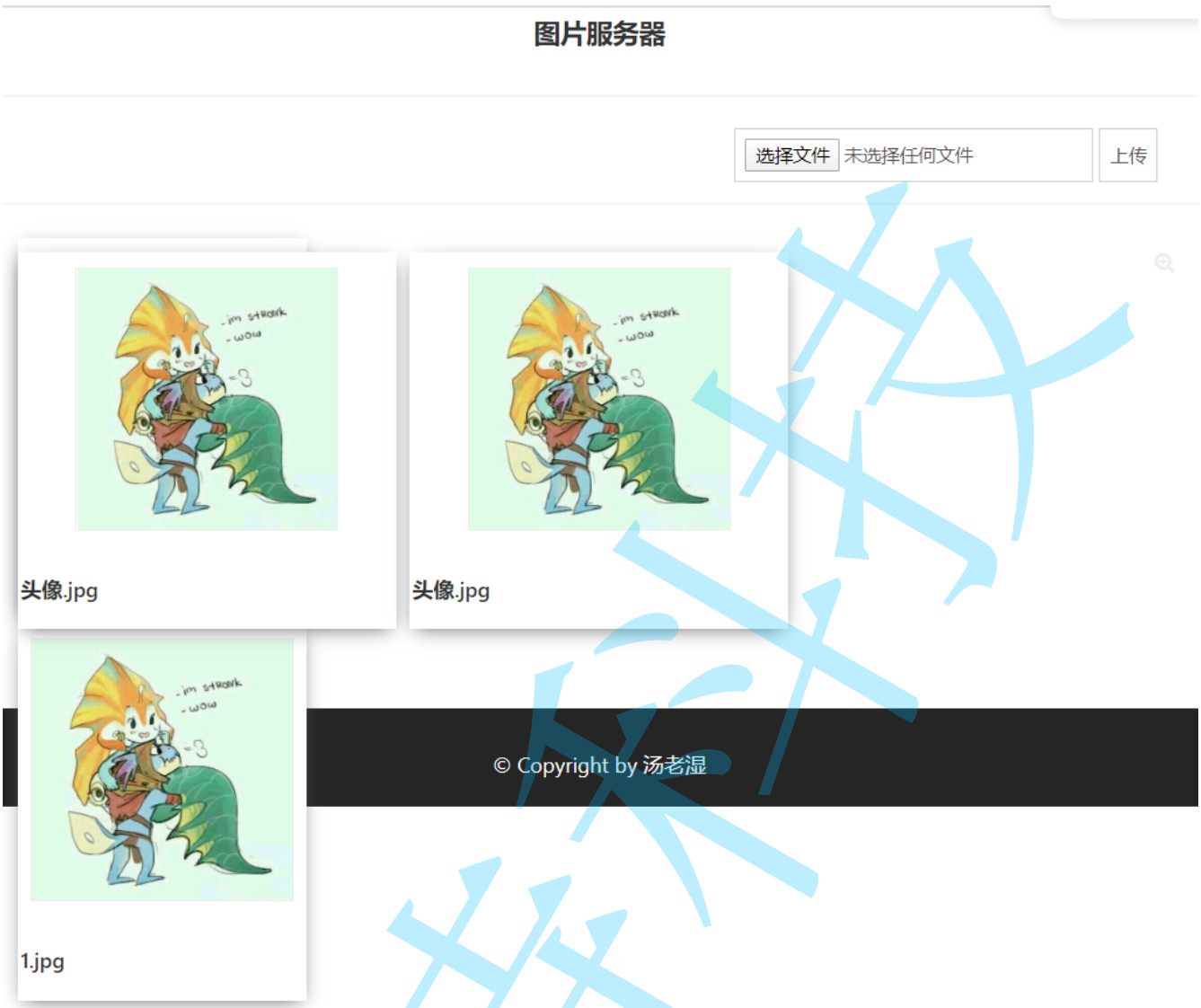
在 methods 中新增获取所有图片的方法.

```
getImages() {
    $.ajax({
        url: "image",
        type: "get",
        context: this,
        success: function(data, status) {
            this.images = data;
        }
    })
}
....
// 页面加载时调用
app.getImages();
```

部署到服务器上, 测试效果.

控制页面重绘

多上传几个图片, 出现以下情况.



这个 bug 出现的原因和用到的图片展示插件有关. 此处的图片展示插件 (img.js) 会根据得到的图片来计算每个图片所处的位置. 但是我们此处页面绘制在前, 而获取图片顺序在后, 导致计算的图片位置出现错误.

经过观察发现, 只要调整浏览器宽度, img.js 就能自动进行重绘. 因此我们只要主动触发浏览器 resize 事件即可.

```
getImages() {
    $.ajax({
        url: "image",
        type: "get",
        context: this,
        success: function(data, status) {
            this.images = data;
            // 这个代码用来触发 resize 事件.
            $("#app").resize();
        }
    })
},
```

完善上传功能

当前的上传请求会返回一个 JSON 格式的数据. 而我们更需要的是直接能看到上传的效果.

修改上传接口的响应, 直接返回一个 302 响应, 重定向回主页.

修改 ImageServlet.doPost

在上传成功代码最后, 加上一个重定向

```
resp.sendRedirect("index.html");
```

实现删除图片

图片下方新增删除按钮

```
<button style="width:100%" class="am-btn am-btn-success" v-
on:click="remove(image.image_id)">删除</button>
```

实现事件处理函数

```
remove(image_id) {
    $.ajax({
        url:"image?image_id=" + image_id,
        type:"delete",
        context: this,
        success: function(data, status) {
            this.getImages();
            alert("删除成功");
        }
    })
}
```

验证删除效果.

阻止点击事件冒泡

此时发现个问题, 点击删除按钮之后, 会触发预览图片效果.

这是因为 JavaScript 的事件冒泡机制导致的. 一个标签接受到的事件会依次传给父级标签.

此处需要阻止 click 事件冒泡. Vue 中使用 v-on:click.stop 即可.

```
<button style="width:100%" class="am-btn am-btn-success" v-
on:click.stop="remove(image.image_id)">删除</button>
```

后续扩展点

实现基于白名单方式的防盗链

通过 HTTP 中的 refer 字段判定是否是指定网站请求图片.

修改 ImageShowServlet.doGet 方法

新增属性:

```
private static HashSet<String> whiteList = new HashSet<>();
static {
    whiteList.add("http://47.98.116.42:8080/java_image_server/index.html");
}
```

新增以下逻辑:

```
// 校验 refer 是否在白名单中
String referer = req.getHeader("Referer");
if (!whiteList.contains(referer)) {
    data.put("ok", false);
    data.put("reason", "未授权使用");
    String respData = gson.toJson(data);
    resp.getWriter().write(respData);
    return;
}
```

注意: 这种方式并非一劳永逸. 对方完全可以构造一个一模一样的 Referer. 其他防盗链方法参考 "其他扩展思路" 章节.

基于 MD5 实现相同内容图片只存一份

百度网盘的 "秒传" 功能就是类似的原理.

整体思路

- 修改上传图片代码, 使用 md5 作为文件名.
- 修改 DAO 层代码, 在 DAO 层实现一个 selectByMD5 方法, 根据 MD5 来查找数据库中的图片信息.
- 修改上传图片代码, 存储文件时先判定, 该 md5 对应的文件是否存在, 存在就不必写磁盘了.
- 修改删除图片代码, 先删除数据库记录, 删除完毕后, 看数据库中是否存在相同 md5 的记录. 如果不存在, 就删除磁盘文件.

实现计算 MD5

修改 pom.xml, 引入依赖

```
<!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.13</version>
</dependency>
```

修改 ImageServlet.doPost 方法, 实现计算 MD5.

```
// 实现计算 MD5 的方式
image.setMd5(DigestUtils.md5Hex(item.get()));
// 修改文件路径的生成方式
image.setPath(PATH_BASE + image.getMd5());
```

修改 ImageDao

新增方法 selectByMD5

```
public Image selectByMD5(String md5) {
    Connection connection = DBUtil.getConnection();
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String sql = "select * from image_table where md5 = ?";
    try {
        statement = connection.prepareStatement(sql);
        statement.setString(1, md5);
        resultSet = statement.executeQuery();
        while (resultSet.next()) {
            Image image = new Image();
            image.setImageId(resultSet.getInt("image_id"));
            image.setImageName(resultSet.getString("image_name"));
            image.setSize(resultSet.getInt("size"));
            image.setUploadTime(resultSet.getString("upload_time"));
            image.setMd5(resultSet.getString("md5"));
            image.setContentType(resultSet.getString("content_type"));
            image.setPath(resultSet.getString("path"));
            return image;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtil.close(connection, statement, resultSet);
    }
    return null;
}
```

根据MD5决定写入文件

修改 ImageServlet.doPost 方法.

如果该 MD5 值的文件不存在, 才真的写入磁盘.

如果该 MD5 值的文件存在, 则直接使用原来的文件, 不必再写一次磁盘.

```
//  d) 将 Image 对象写入数据库中
ImageDao imageDao = new ImageDao();

// [这个代码位置很关键!!] 根据 md5 查询是否已经有 Image 存在
Image existImage = imageDao.selectByMD5(image.getMd5());

imageDao.insert(image);
// 2. 获取到图片内容, 写入到磁盘中. 如果图片存在就不写文件了
if (existImage == null) {
    File file = new File(image.getPath());
    try {
        item.write(file);
    } catch (Exception e) {
```

```
            e.printStackTrace();
            resp.setStatus(500);
            // 注意，此处可能出现脏数据的情况，比如数据库插入成功，但是这里文件写入失败
            resp.getWriter().println("{ \"ok\" : false, \"reason\" : \"文件写入失败\"}");
            return;
        }
    }

    // 3. 设置返回的响应结果
    resp.sendRedirect("index.html");
```

根据 MD5 决定删除文件

修改 ImageServlet.doDelete 方法.

其他代码不变, 只修改删除文件逻辑.

```
    //  b) 删除磁盘文件，此时说明已经不存在相同 MD5 的图片了，可以删除磁盘上的图片文件
    Image existImage = imageDao.selectByMD5(image.getMd5());
    if (existImage == null) {
        File file = new File(image.getPath());
        file.delete();
    }
```

其他扩展思路

1. 多个小图片拼接成一个大文件, 提高磁盘存储效率, 降低磁盘碎片.
2. 支持图片处理功能(例如返回指定大小的图片)
3. 防盗链其他方式 https://www.cnblogs.com/wangyongsong/p/8204698.html

附录

关于 MD5

MD5的特点

https://mp.weixin.qq.com/s?src=11×tamp=1579341359&ver=2104&signature=d94gRODg3uu5or1b5Eykt2jpEnx*llWYIpIN4gcvyYh*JlyT-9a0IZGBk3NXSw8GKzhyN5jLeGB8yGDnUN2xeMhWjHb5BRiW7Fb5-rVH6wxB-61qAQDCsGn05bp3wn9U&new=1

MD5算法原理

https://mp.weixin.qq.com/s?src=11×tamp=1579341305&ver=2104&signature=3Qrmz-tuZumRTSPrp9B*7MRJmUlE5znm3gOfDSePZH77jXCL-IrPiUjiGjEoVg8hIqiM73tyDHG3UCfBM*J3t0mCVryyvtpl2moF2RyB0qio3zEGRW6kwX5f3Zog8MlT&new=1

安装 Postman

官方下载地址

https://dl.pstmn.io/download/latest/win64

注意!!! 网上大多数关于 POSTMAN 的安装教程都过时了. 当前不用翻墙就能直接在官网上下载安装包.

比特科技