基于互联网网页的小型分布式文件系统

可行性报告

王珺 夏昊珺 滕思洁 郑值

2017 年 4 月 9 日星期日

摘要

本报告首先提出了一个基于互联网网页的小型分布式文件系统设计并讨论了其可行性；接着介绍了上述分布式文件系统的设计中用到的范德蒙编码、Java 虚拟机等技术与理论，分析了基于这些技术设计的分布式文件系统在特定的场景下相较于传统的分布式文件系统的创新与优势。最后，本文调研了文件索引服务器和浏览器对 DFS 的支持作为对此前的调研报告的补充。

目录

一、实现方案与可行性分析

在这一部分中，我们将分别阐述客户端、服务器、网页服务器以及动态网页四者的设计与初步实现并讨论这些实现方案的可行性。

客户端服务程序

本项目的 java 客户端是一个服务程序，与服务器基于 tcp 协议的 socket 通信建立连接。客户端通过配置文件来进行配置。配置文件会选择要进行共享的文件列表，并提供服务器的 IP 接口。

可使用 PropertiesLoaderUtils 工具类来进行配置文件的读取和修改，Spring 提供的 PropertiesLoaderUtils 允许直接通过基于类路径的文件地址加载属性资源，最大的好处就是：实时加载配置文件，修改后立即生效，不必重启。

Java 客户端提供以下几种功能：

1. 获取本地目录：

服务器需要知道本地机器参与共享的文件列表
使用 java.io.File 类即可操作获取

```
class FileList
{
    File file;
    String[] fileList;

    public FileList(String fileRoute)
    {
        //通过指定路径，构造 file 对象
        file = new File(fileRoute);
        //获取指定路径下的文件列表，保存至 fileList 中
        fileList = file.list();
    }
}
```

2. 对文件进行分块

当服务器发来需要分块的文件时，客户端需要对文件应用 erasure code 算法进行分块，分块算法已经基于 backblaze 开源实现。

3. 向服务器发送文件碎片、接收服务器传来的碎片

当文件分块完成后，客户端便向服务器发出 post 请求，服务器开通数据传输端口，客户端将文件碎片发送向客户端。若有服务器发出获取碎片请求，客户端则将存在本地的碎片发向客户端。

收发文件已经有很成熟的实现方法，以发送一张照片为例：

```
public class Main {
```

```
  public static void main(String[] args) throws IOException {
    ServerSocket servsock = new ServerSocket(12345);
    File myFile = new File("qq.jpg");
    while (true) {
      Socket sock = servsock.accept();
      byte[] mybytearray = new byte[(int) myFile.length()];
      BufferedInputStream      bis      =      new      BufferedInputStream(new
FileInputStream(myFile));
      bis.read(mybytearray, 0, mybytearray.length);
      System.out.println(mybytearray.length);
      OutputStream os = sock.getOutputStream();
      os.write(mybytearray, 0, mybytearray.length);

      os.flush();
      sock.close();
    }
```

4. 响应服务器删除本地文件

```
public void testDelete(){
        try {
            ClientGlobal.init(conf_filename);

            TrackerClient tracker = new TrackerClient();
            TrackerServer trackerServer = tracker.getConnection();
            StorageServer storageServer = null;

            StorageClient storageClient = new StorageClient(trackerServer,
                    storageServer);
            int         i         =         storageClient.delete_file("group1",
"M00/00/00/wKgRcFV_O8OAK_KCAAAA5fm_sy874.conf");
            System.out.println( i==0 ? "删除成功" : "删除失败:"+i);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

服务器程序

服务器程序是分布式文件系统的核心组件，其管理着整个分布式文件系统的元数据并支持客户端服务程序和动态网页与分布式文件系统的交互。

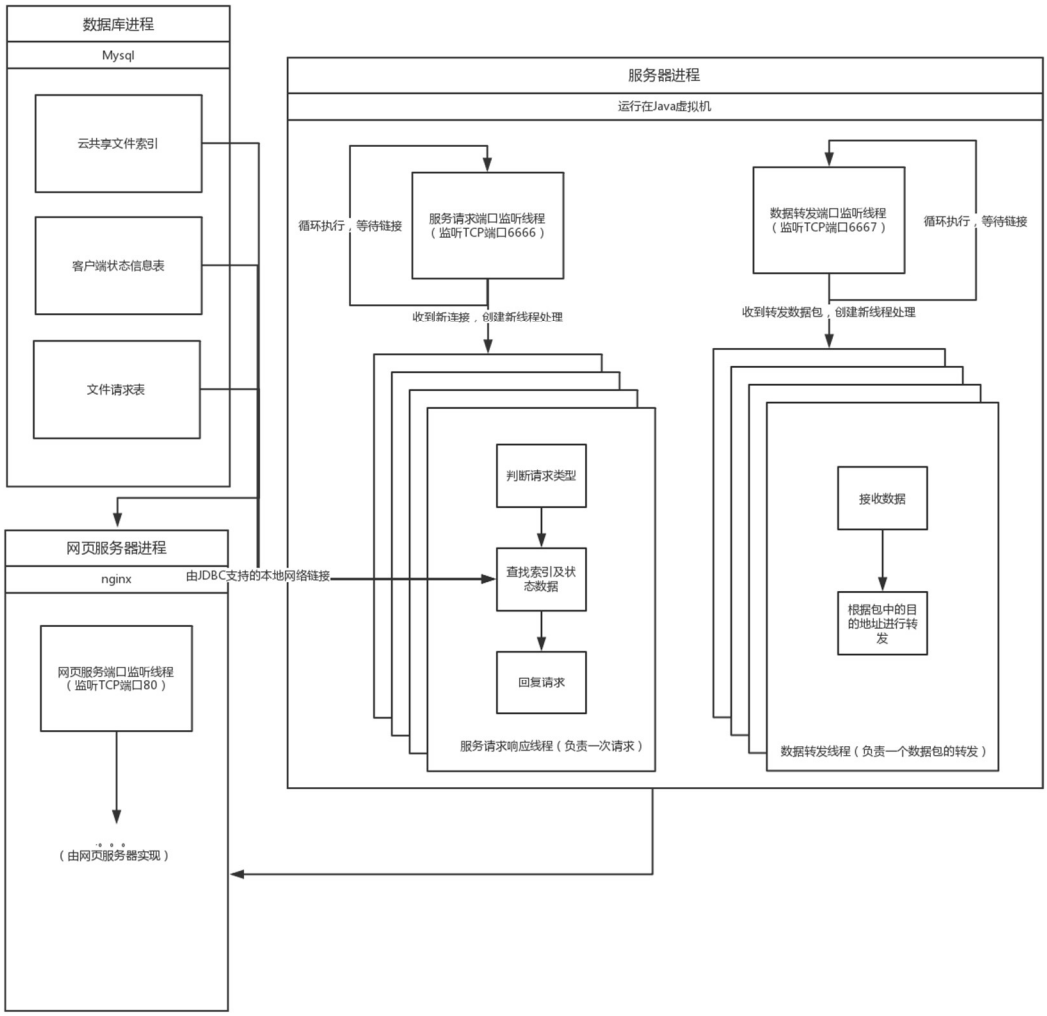
服务器程序的功能主要有：

连接类	数据管理类
接收、回复、转发服务请求与控制信息	维护云共享文件索引

转发数据	维护各个客户端的状态信息
支持动态网页	记录、处理当前等待响应的文件请求

（其中支持动态网页的部分将放在网页服务器中讨论）

服务器程序的结构如下：



服务器程序包括两个常开的线程用于监听两个不同的 TCP 端口分别用来回应服务请求和转发数据，对于每个 TCP 链接，服务器进程将创建一个线程进行处理，这将保证服务器可以在处理一个请求时不至于失去响应。

Java 提供了接口 Runable 供多线程编程使用，实现 Runable 的类只需重写 run 方法便可以在类中实例化一个线程（Thread 类）对象，并通过 start 方法将这个线程提交给 JVM 的线程调度队列，形如：

Main.java	HelloThread.java

public class Main{
public static void main(String[] args){
new HelloThread("thread #1").start();
new HelloThread("thread #2").start();
new HelloThread("thread #3").start();
	class HelloThread implements Runnable{
	private String name;
	public HelloThread(String name) {

```
    }
}
```

```
        this.name=name;
    }

    public void run(){
      System.out.println("helloworld from "+name);
    }

    public void start() {
        Thread thread;
        thread = new Thread(this,name);
        thread.start();
    }
}
```

为了避免复杂的线程间同步与内存管理，本报告设计的分布式文件系统采用 MySQL 数据库保存文件索引信息和各客户端的状态信息。MySQL 原本是一个开放源代码的关系数据库管理系统，原开发者为瑞典的 MySQL AB 公司。2008 年 MySQL AB 公司被昇阳微系统收购，2009 年，甲骨文公司收购了昇阳微系统公司，故当前 MySQL 为 Oracle 旗下产品。同其他数据库一样，MySQL 可以通过标准查询语言（SQL）进行查询。

为了避免在数据库上进行漫长的线性查找，MYSQL 支持为数据库添加索引。引擎为 InnoDB 或 MyISAM 的数据库支持基于 B-Tree 的索引，其可以将查询时间减少到 O(log(n))；引擎为 MEMORY 的数据库支持基于 Hash 的索引，其可以将查询时间减少到 O(1))。添加索引可以通过 SQL 命令

alter table *table_name* add index *index_name* using {hash, BTREE} (*column_name*);

执行；具体数据库引擎的选择则还要考虑到数据库的大小（如 MEMORY 引擎的数据库大小不能超过 RAM 大小）和并发性的要求。

MySQL 数据库支持并发操作，为了在并发操作中保持数据的一致性，MySQL 提供锁机制与事务机制。MySQL 的锁机制包括表级锁（锁定整张表）和行级锁（锁定特定行，并发程度更高）；MySQL 的事务机制保证一系列的 SQL 可以被原子的执行且不被其他事务干扰。不同引擎的表对并发性的支持是不同的，如 InnoDB 实现了行级锁而 MyISAM 存储引擎不支持行级锁。MySQL 中 InnoDB 引擎支持具有事务、回滚和崩溃修复能力的事务安全型表，对并发性有最好的支持。因此，考虑到索引的速度、数据库大小的支持与对并发性的支持，本报告中设计的分布式文件系统采用基于 InnoDB 引擎的数据库。

Java 提供了 Java 数据库连接（JDBC）作为规范客户端程序如何来访问数据库的应用程序接口并提供了诸如查询和更新数据库中数据的方法。只需加载 MySQL 的驱动，就可以在 Java 程序中链接 MySQL 数据库并通过 SQL 查询（或修改）数据库中的内容。

下面介绍服务器程序提供的功能。

接收、回复、转发服务请求与控制信息：服务器与客户端通过非持续的 TCP 链接传递 HTTP 报文来交换信息。这些信息包括了

控制类	服务类
客户端上线报告	客户端上传共享文件请求
客户端剩余空间报告	客户端删除共享文件请求
客户端心跳报告	客户端创建共享文件夹请求
删除保存在客户端的碎片的请求	客户端删除共享文件夹请求

Java 提供了 ServerSocket 类和 Socket 类来进行基于 TCP 链接的网络通信。ServerSocket 类可以用于在服务器端创建一个欢迎套接字，Socket 类可以在客户端或服务器端创建链接套接字。

转发数据：由于各个客户端之间未必可以直接连接（如客户端可能隐藏在 NAT 后），客户端有时需要通过服务器转发消息，这种情况可能有两种：
（1）当一个客户端上传了文件，其需要发送碎片到另一个客户端
（2）当网页请求文件，其需要接收各客户端的碎片

对于以上的任何一种情况，转发数据时将随数据附带目的地址，故服务器只需接收数据并将数据发送到目的地址即可。

转发数据的实现有赖于老师提供的接口。

维护云共享文件索引
云共享文件索引以 MySQL 数据库中的一系列 table 的形式保存。每台共享了文件的电脑均在数据库中对应两个 table，其一记录了这台电脑上共享的每个文件的唯一标识符和其逻辑位置；其二记录了这台电脑上各个文件的碎片的物理位置和其唯一标识符。

维护各个客户端的状态信息
客户端状态信息以 MySQL 数据库中的一个 table 的形式保存。这个 table 中包括了客户端的唯一标识符，在线情况，剩余空间及当前复杂维持与这个客户端的控制链接（TCP 链接）的线程的编号。

记录、处理当前等待响应的文件请求
由于服务器并不时时连接各客户端，必须在 MySQL 数据库中使用一个 table 记录当前网页提出的文件请求。当服务器收到来自客户端的心跳连接时，其将查询文件请求表，如果发现有对客户端上文件的请求，则在回复心跳连接时将文件请求发给客户端并令其（通过服务器）将文件发往请求方。

网页和web服务器以及web应用程序三者的交互

网页使用 JS：
JavaScript 是 Web 的编程语言，所有现代的 HTML 页面都使用 JavaScript。
使用 JS 可以实现网页背后的逻辑，让网页在用户的点击下产生一系列反应。

使用 JS 时包含 JQuery 库：
jQuery 是一个 JavaScript 库，"写的少，做的多"，极大地简化了 JavaScript 编程。

网页 UI 设计采用 Bootstrap 框架：

Bootstrap，来自 Twitter，是目前最受欢迎的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的，它简洁灵活，使得 Web 开发更加快捷。

1. 考虑到它适用于快速开发+可视化开发+提供可重用组件：进度条等+UI 美观，所以值得使用。
2. 还有一点是，它秉承移动设备优先。Bootstrap 的响应式 CSS 能够自适应于台式机、平板电脑和手机，可以自动适应于屏幕大小调节网页布局。
3. 浏览器支持：所有的主流浏览器都支持 Bootstrap。

网页和服务器的交互采用 AJAX：

AJAX = 异步 JavaScript 和 XML（Asynchronous JavaScript and XML）

AJAX 是与服务器交换数据的技术，它在不重载全部页面的情况下，实现了对部分网页的更新。

jQuery 提供多个与 AJAX 有关的方法。（如果没有 jQuery，AJAX 编程还是有些难度的。编写常规的 AJAX 代码并不容易，因为不同的浏览器对 AJAX 的实现并不相同。这意味着你必须编写额外的代码对浏览器进行测试。不过，jQuery 团队为我们解决了这个难题，我们只需要一行简单的代码，就可以实现 AJAX 功能。）

jQuery AJAX 方法

jQuery AJAX 方法

AJAX 是一种与服务器交换数据的技术，可以在不重新载入整个页面的情况下更新网页的一部分。

下面的表格列出了所有的 jQuery AJAX 方法：

方法	描述
$.ajax()	执行异步 AJAX 请求
$.ajaxPrefilter()	在每个请求发送之前且被 $.ajax() 处理之前，处理自定义 Ajax 选项或修改已存在选项
$.ajaxSetup()	为将来的 AJAX 请求设置默认值
$.ajaxTransport()	创建处理 Ajax 数据实际传送的对象
$.get()	使用 AJAX 的 HTTP GET 请求从服务器加载数据
$.getJSON()	使用 HTTP GET 请求从服务器加载 JSON 编码的数据
$.getScript()	使用 AJAX 的 HTTP GET 请求从服务器加载并执行 JavaScript
$.param()	创建数组或对象的序列化表示形式（可用于 AJAX 请求的 URL 查询字符串）
$.post()	使用 AJAX 的 HTTP POST 请求从服务器加载数据
ajaxComplete()	规定 AJAX 请求完成时运行的函数
ajaxError()	规定 AJAX 请求失败时运行的函数
ajaxSend()	规定 AJAX 请求发送之前运行的函数
ajaxStart()	规定第一个 AJAX 请求开始时运行的函数
ajaxStop()	规定所有的 AJAX 请求完成时运行的函数
ajaxSuccess()	规定 AJAX 请求成功完成时运行的函数
load()	从服务器加载数据，并把返回的数据放置到指定的元素中
serialize()	编码表单元素集为字符串以便提交
serializeArray()	编码表单元素集为 names 和 values 的数组

通讯格式：

网页和后台 通信发送的内容是 JSON 字符串（JavaScript JSON，英文全称 JavaScript Object Notation）：

JSON 是用于存储和传输数据的格式，通常用于服务端向网页传递数据 ，是一种轻量级的数据交换格式。

JSON 是独立的语言，而且易于理解。

相关函数

函数	描述
JSON.parse()	用于将一个 JSON 字符串转换为 JavaScript 对象。
JSON.stringify()	用于将 JavaScript 值转换为 JSON 字符串。

Apache+Tomcat+Servlet+java web 应用

1. Web 服务器采用：Apache

Apache HTTP 服务器是一个模块化的服务器，可以运行在几乎所有广泛使用的计算机平台上。其属于应用服务器。Apache 支持支持模块多，性能稳定，Apache 本身是静态解析，适合静态 HTML、图片等，但可以通过扩展脚本、模块等支持动态页面等。

（Apche 可以支持 PHPcgiperl,但是要使用 Java 的话，你需要 Tomcat 在 Apache 后台支撑，将 Java 请求由 Apache 转发给 Tomcat 处理。）

2. 配合 Apache 使用 Tomcat（应用服务器）

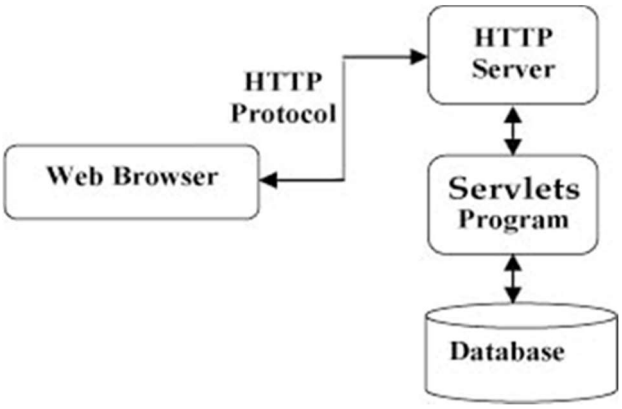
Tomcat 是应用（Java）服务器，它只是一个 Servlet(JSP 也翻译成 Servlet)容器，可以认为是 Apache 的扩展，但是可以独立于 Apache 运行。

3. 使用 Servlat 实现 web 应用程序和其他进程间的通讯

Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

使用 Servlet，您可以收集来自网页表单的用户输入，呈现来自数据库或者其他源的记录，还可以动态创建网页。

只需要在我们的 web 应用程序中导入一些 java 库就可以通过库函数实现对 http 端口的请求监听和响应。



Servlet 执行以下主要任务：
读取客户端（浏览器）发送的显式的数据。这包括网页上的 HTML 表单，或者也可以是来自 applet 或自定义的 HTTP 客户端程序的表单。
读取客户端（浏览器）发送的隐式的 HTTP 请求数据。这包括 cookies、媒体类型和浏览器能理解的压缩格式等等。
处理数据并生成结果。这个过程可能需要访问数据库，执行 RMI 或 CORBA 调用，调用 Web 服务，或者直接计算得出对应的响应。
发送显式的数据（即文档）到客户端（浏览器）。该文档的格式可以是多种多样的，包括文本文件（HTML 或 XML）、二进制文件（GIF 图像）、Excel 等。
发送隐式的 HTTP 响应到客户端（浏览器）。这包括告诉浏览器或其他客户端被返回的文档类型（例如 HTML），设置 cookies 和缓存参数，以及其他类似的任务。

4. 服务器端网络服务程序基于 JAVA
我们已有基于 java 实现的一些模块，所以使用 java 可以减少任务量（不用重写模块）；
Java 的网络功能很强大，写起来也方便；
客户端服务程序使用 java，两个互相交互的服务程序实现上尽量保持统一。

动态网页设计

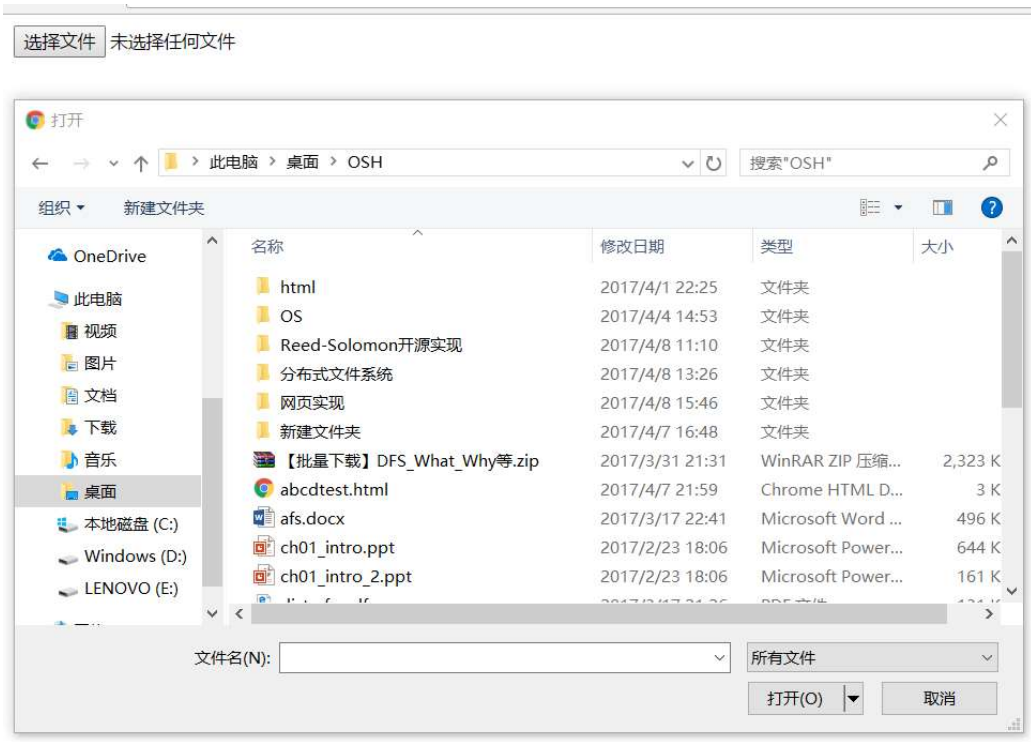
网页对文件系统的支持：
JavaScript 对本地文件操作的支持：

Js 提供如下文件操作函数：
CopyFile() 复制文件
CopyFolder() 复制目录
CreateFolder() 创建新目录
CreateTextFile() 生成一个文件
DeleteFile() 删除一个文件
DeleteFolder() 删除一个目录
DriveExists() 检验盘符是否存在
Drives 返回盘符的集合
FileExists() 检验文件是否存在
FolderExists 检验一个目录是否存在
GetAbsolutePathName() 取得一个文件的绝对路径
GetBaseName() 取得文件名
GetDrive() 取得盘符名
GetDriveName() 取得盘符名
GetExtensionName() 取得文件的后缀
GetFile() 生成文件对象
GetFileName() 取得文件名
GetFolder() 取得目录对象
GetParentFolderName 取得文件或目录的父目录名
GetSpecialFolder() 取得特殊的目录名
GetTempName() 生成一个临时文件对象
MoveFile() 移动文件

MoveFolder() 移动目录

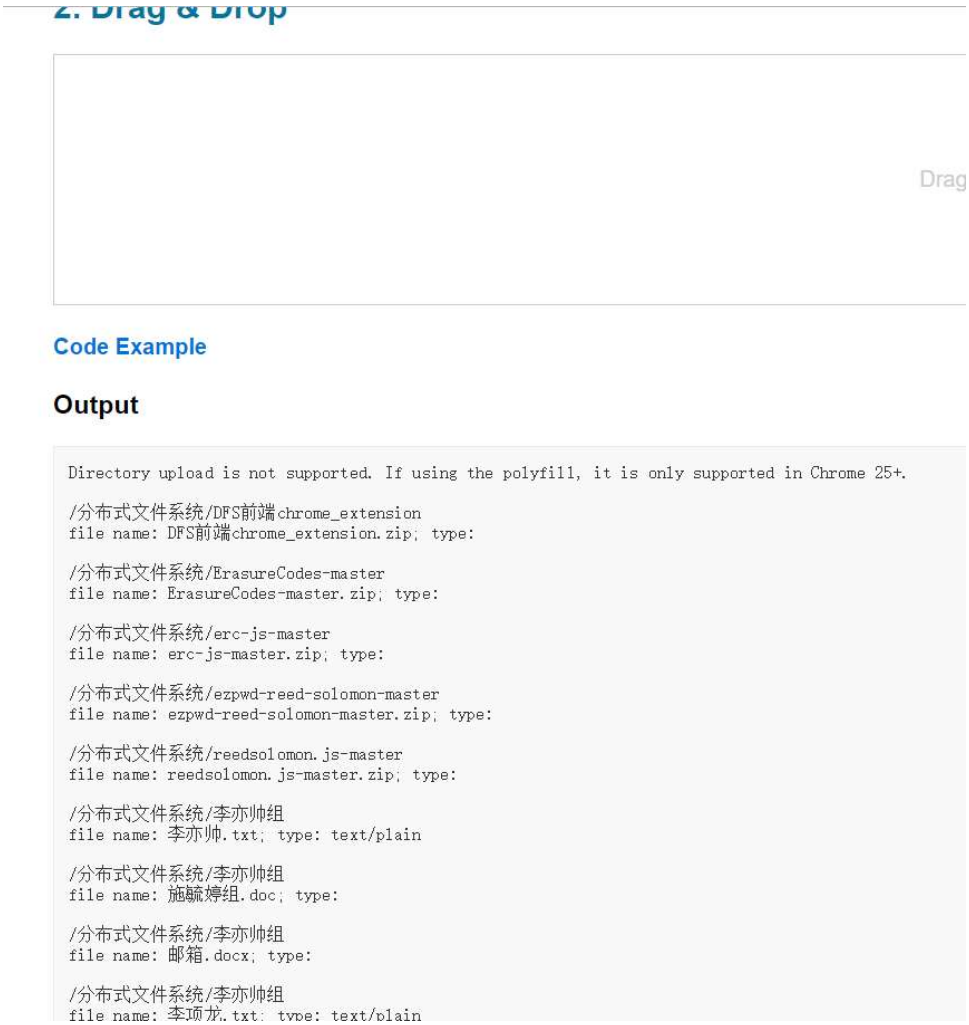
可以利用 js 实现在网页上获取本地文件目录，并上传本地文件。我们尝试了利用 js 选择并打开本地文件：

如图：



可以利用网页获取本地文件目录：

如：



Eursure code 的 Javascript 实现：

Reed-solomom 的开源 js 实现库 erc-js：

###Classes

ReedSolomon

Public-facing interface providing encode/decode functionality

ReedSolomon.Codec

Implements the Reed-Solomon error correction algorithm

ReedSolomon.GaloisField

Implements a Galois field GF(p^n) over p = 2

ReedSolomon.Utils

Implements string and array manipulation methods

库的使用：

```
// n is the length of a codeword
var rs = new ReedSolomon(n);
var enc = rs.encode('hello world');
var msg = rs.decode(enc);
```

二、理论依据与技术依据

本报告设计的分布式文件系统中主要涉及到了以下的理论与技术：

网络数据传输：为了使分布在不同地点的各种设备可以共同维护、使用我们设计的分布式文件系统，基于网络的数据交换是不可避免的。目前在世界范围内应用最为广泛的网络是因特网，其使用TCP/IP协议互联了世界上数以亿计的各类设备。因此，我们设计的分布式文件系统将使用因特网进行数据的传输。

JVM：我们设计的分布式文件系统使用Java编程，因此其将运行在Java虚拟机（JVM）上。JVM是一种用于计算设备的规范，它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。JVM是Java语言实现与平台的无关性的关键：一般的高级语言如果要在不同的平台上运行需要编译成不同的目标代码，而JVM屏蔽了与具体平台相关的信息，使得Java语言只要编译为字节码就可以在多种平台上不加修改地运行。

Erasure Code：我们设计的分布式文件系统使用Erasure Code来实现副本管理。Erasure Code（又称纠删码）是一种编码技术，它可以将n份原始数据，增加m份数据，并能通过n+m份中的任意n份数据，还原为原始数据。

网盘：我们设计的分布式文件系统使用商业网盘来增强可用性。网盘是由互联网公司推出的在线存储服务，利用这些公司的服务器为用户提供文件的存储、访问、备份、共享等功能。由于目前的商业网盘都由专业人员维护并采用了复杂的副本机制，其可靠性与稳定性均有较好的保证。同时，大多数网盘也都提供了API接口和SDK的支持，这使得利用网盘服务来实现新服务变得非常简单。

对于上述大多数内容，我们在之前的调研报告中已经进行了介绍。下面，我们将着重补充如何使用范德蒙码（Erasure Code的一种）进行文件的编码与解码并简要讨论JVM的原理。

用范德蒙码编码及解码文件

编码：

若用d来表示data块，c来表示code块，则采用下图所示范德蒙生成矩阵由data块生成code块：

$$\begin{bmatrix} 1 & 0 & 0\cdots & 0 \\ 0 & 1 & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 1 \\ 1 & 1 & 1\cdots & 1 \\ 1 & 2 & 3\cdots & n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1}\cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

4个data块，2个Code块情况下，编码过程如下：

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\1&1&1&1\\1&2&4&8\end{bmatrix}\cdot\begin{bmatrix}D_0\\D_1\\D_2\\D_3\end{bmatrix}=\begin{bmatrix}D_0\\D_1\\D_2\\D_3\\C_0\\C_1\end{bmatrix}$$

其中Code块是：

$$C_0 = D_0 + D_1 + D_2 + D_3$$
$$C_1 = D_0 + 2 \times D_1 + 4 \times D_2 + 8 \times D_3$$

解码：

解码过程原理是：未损失的数据块和校验块乘以编码矩阵的逆矩阵可以得到原来的数据。

以4个data块，2个Code块的情况的解码来解释，当code的块数为2时，最多坏掉两块数据块（按照解方程就是四元一次方程，至少4个才能解出来四个元的值）。此处假设一个数据块D1和一个code块C0丢失。

解码过程分为两步：

1）顺序遍历编码矩阵的前n行，顺序选取没有损坏的前k行（意思是该行对应的数据块或者校验块没有损坏）。生成k*k的矩阵M。本例中M矩阵如下：

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&0&0&1\\1&2&4&8\end{bmatrix}$$

编码encode的时候这几行发生了下面的事情：

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&0&0&1\\1&2&4&8\end{bmatrix}*\begin{bmatrix}D_0\\D_1\\D_2\\D_3\end{bmatrix}=\begin{bmatrix}D_0\\D_1\\D_2\\C_1\end{bmatrix}$$

所以解码的时候，有 C_1, D_0, D_1, D_2 以及M，很显然可以通过求M的逆矩阵来求出 D_0, D_1, D_2, D_3：

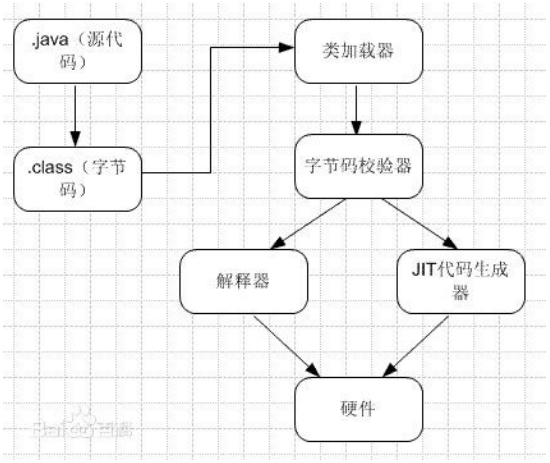
$$M^{-1} * \begin{bmatrix}D_0\\D_1\\D_2\\C_1\end{bmatrix}=\begin{bmatrix}D_0\\D_1\\D_2\\D_3\end{bmatrix}$$

2）求出损失的数据。由于（1）中已经求出来了所有的数据块的内容，而且编码矩阵是知道的，因此可以求出所有的数据，对于本例子，其实是在第三个式子两边同时乘以一个矩阵来求出 C_0，得到的就是相应的编码矩阵的部分，于是就求出来了丢失的数据 D_1 和 Code C_0。

JVM

JVM是Java Virtual Machine（Java虚拟机）的缩写，JVM是一种用于计算设备的规范，它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。Java语言的一个非常重要的特点就是与平台的无关性。而使用Java虚拟机是实现这一特点的关键。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的目标代码。而引入Java语言虚拟机后，Java语言在不同平台上运行时不需要重新编译。Java语言使用Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。这就是Java的能够“一次编译，到处运行”的原因。

JVM是java的核心和基础，在java编译器和os平台之间的虚拟处理器。它是一种基于下层的操作系统和硬件平台并利用软件方法来实现的抽象的计算机，可以在上面执行java的字节码程序。java编译器只需面向JVM，生成JVM能理解的代码或字节码文件。Java源文件经编译器，编译成字节码程序，通过JVM将每一条指令翻译成不同平台机器码，通过特定平台运行。Jvm运行原理如下图



JVM指令系统同其他计算机的指令系统极其相似。Java指令也是由操作码和操作数两部分组成。操作码为8位二进制数，操作数紧随在操作码的后面，其长度根据需要而不同。JVM只设置了4个最为常用的寄存器。它们是：

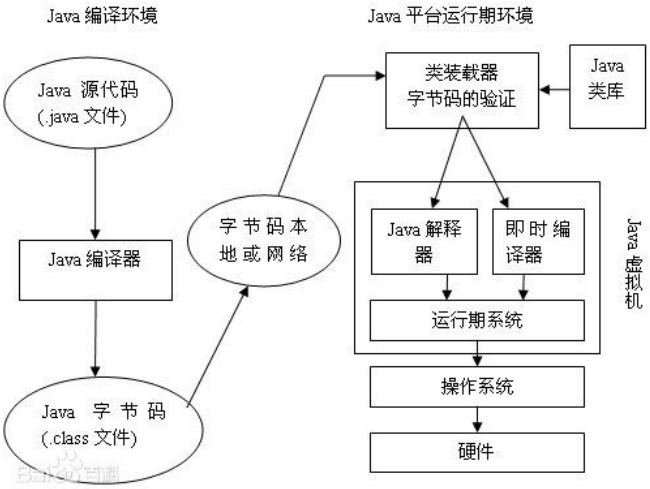
pc程序计数器、optop操作数栈顶指针、frame当前执行环境指针、vars指向当前执行环境中第一个局部变量的指针

所有寄存器均为32位。pc用于记录程序的执行。optop,frame和vars用于记录指向Java栈区的指针。

Java栈是JVM存储信息的主要方法。当JVM得到一个Java字节码应用程序后，便为该代码中一个类的每一个方法创建一个栈框架，以保存该方法的状态信息。每个栈框架包括以下三类信息：

局部变量、执行环境、操作数栈

JVM体系结构如下图


Java 编译环境
Java 平台运行期环境
Java 源代码
(.java 文件)
Java 编译器
Java 字节码
(.class 文件)
字节码本地或网络
类装载器
字节码的验证
Java
类库
Java 解释器
即时编译器
运行期系统
Java 虚拟机
操作系统
硬件

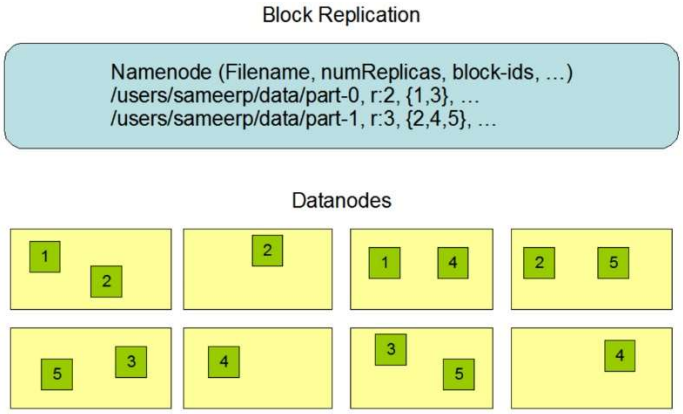
三、创新点

在这一部分中，我们会从备份方式、可用性、访问方式和部署难度五个方面将本报告中设计的分布式文件系统与目前已有的分布式文件系统进行对比并分析其具有的优势与创新。

备份机制：Erasure code

为了提供容错功能并保证在数据节点（无论是传统的数据服务器还是P2P机制中的对等方）崩溃或离线时数据的可用性，大多数分布式文件系统都实现了备份机制，即通过保存文件的多个副本保证文件的安全性。

目前包括GFS、HDFS在内的主流分布式文件系统往往采用了完全复制副本[3.3][3.4]，即将一个文件（文件块）完整地复制为多份完全相同的副本并保存在不同的位置。以HDFS为例，HDFS的备份机制如下图[3.3]所示：



HDFS允许用户为不同的文件设置不同的复制级别并根据文件的复制级别将文件复制为相同的多份并保存在不同的服务器上。HDFS的默认复制级别为3，这时其会将一个副本存放在本地机架的节点上，另一个副本放在同一机架的另一个节点上，最后一个副本放在不同机架的节点上。

完全复制副本的方式无疑能提供文件的安全性，同时其也允许客户端获取离其最近的副本以加快访问速度，但完全复制副本的缺点也是明显的：其消耗了过多的存储空间，以HDFS默认的3备份为例，这意味着系统只能提供其物理存储空间1/3容量的存储服务。此外，在采用P2P机制的分布式文件系统中，由于对等方的服务较专用的服务器更不稳定，其往往需要设置更大的复制级别，也就意味着可用的存储空间更小。

Erasure Code（中文名为抹除码或纠删码）是一种编码技术，它可以将n个原始数据块编码为n+m个数据块，并能通过在n+m个数据块任取n个进行解码还原出原始数据。Erasure Code有多种实现方式，Reed-Solomon Code[3.5]和 Tonardo Code 是最常见的 2 种编码方式。Reed-Solomon Code的编码方式如下图示，对于原始数据$d_1 \sim d_n$我们可以左乘一个编码矩阵（由单位矩阵与范德蒙矩阵组合而成）进行编码；而解码时只需在编码矩阵中删去未获得的m个数据对应的行，将剩余的n*n 阶矩阵求逆并左乘当前拥有的数据组成的列向量即可。

$$\begin{bmatrix} 1 & 0 & 0\cdots & 0 \\ 0 & 1 & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 1 \\ 1 & 1 & 1\cdots & 1 \\ 1 & 2 & 3\cdots & n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1}\cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

一般的，记编码效率$\mathrm{r} = \frac{n+m}{n}$,表示编码后总占用空间与原始数据大小的比。对于常见的应用，一般把 m 设置为$\frac{n}{2}$左右，即 r 约为 1.5，显然这个结果要远好于完全复制副本的实现方式。

我们计划使用 Erasure Code 作为我们的数据备份方式。目前已经有了一些基于 Erasure Code 的存储系统设计[3.6][3.7]，但正如前文所述，主流的分布式文件系统大多现在仍使用完全复制副本的方式。

可用性：结合对等方主机与网盘服务

除了备份机制，为了进一步提高文件的可用性，我们创新性的将目前已十分成熟的网盘资源整合进了本文设计的分布式文件系统，这种实现方式在目前常见的分布式文件系统中尚未出现。

当一个对等方需要上传云共享文件时，中心服务器会向其返回一个主机列表指示每个块要保存的位置，而主机列表中即包括其他对等方，也包括网盘；类似的，当一个对等方需要下载云共享文件时，中心服务器也会向其返回一个主机列表指示每个块当前保存的位置，对等方可以根据当前各主机的在线情况与网络情况自行决定在哪些对等方/网盘中下载块以组成完整的文件。

需要注意的是，由于访问网盘的协议是由网盘的提供方设计的，其往往与我们设计的对等方间的通讯协议不同。故在本文设计的分布式文件系统中，网盘与对等方间的差异是显式的，即请求块的对等方必须根据主机列表中的主机类型选择合适的通讯协议。

由于网盘往往由专业的数据存储企业运营、维护，其本身也使用了一套数据备份机制，故其几乎可以保证总是可用。因此，网盘的加入能够大大地提高分布式文件系统中文件的可用性。同时，与单纯依靠网盘进行存储相比，我们的设计只需在网盘中存储文件的部分块（而非整个文件），这意味着我们的设计可以利用相同空间的网盘资源存储更多的文件，进而减少了租赁网盘空间的成本。

访问方式：从 API、客户端到网页

以GFS为代表的主流分布式文件系统往往面向专业用户或程序设计者，其提供的访问接口往

往以库函数或POSIX标准的API为主。由于缺少专业的技术人员，一般的家庭、非IT领域的小微企业应用这类分布式文件系统是非常困难的。

我们设计的分布式文件系统提供了包括传统的API、客户端软件在内的多种访问方式。我们创新性的提供了网页访问方式，允许用户使用任何一台连接互联网的主机如同打开商业网盘一样打开我们的分布式文件系统。

部署难度：纯 Java 开发的客户端，可在任何 Java 虚拟机上运行

为了尽可能追求效率，包括GFS、NFS、FastDFS在内的大多数分布式文件系统往往选择使用C/C++进行实现，这意味着在各个服务器/对等方不同构时（如使用了不同的处理器、操作系统）往往需要多次编译以使分布式文件系统可以在所有机器上运行。

随着JIT技术的发展，目前Java的执行效率已得到了巨大的提升，同时，Java基于JVM的运行模式使其可以兼容各种体系结构的主机。我们的客户端完全使用Java编写，这使得任何可以允许Java虚拟机的主机理论上都可以运行我们的客户端。

附录：相关调研

在这一节中我们调研了DFS的浏览器支持和目录索引服务器作为对调研报告的补充。

目录索引服务器

文件结构 ：

目录项：一个文件系统维护的一个索引节点的数组。目录项中的每一项包括文件索引节点 i 的节点号和文件名，目录只是将文件的名称和它的索引节点号结合在一起的一张表。

索引节点 ： 又称 I 节点，在文件系统结构中，包含有关相应文件的信息的一个记录，这些信息包括文件权限、文件名、文件大小、存放位置、建立日期等。文件系统中所有文件的索引节点保存在索引节点表中。

数据 ： 文件的实际内容。（在服务器中不保存文件数据）

文件的记录形式：

文件系统使用索引节点(inode)来记录文件信息。一个文件系统维护了一个索引节点的数组，每个文件或目录都与索引节点数组中的唯一的元素对应。每个索引节点在数组中的索引号，称为索引节点号。文件系统将文件索引节点号和文件名同时保存在目录中，目录中每一对文件名称和索引节点号称为一个连接。

文件在服务器中的索引：

在我们的 dfs 设计中文件数据不会被保存到服务器中，也就是在服务器中只会储存目录项和索引节点。另外文件目录的部分会增加文件所在的主机以确定文件所在位置。文件索引服务器目的：通过文件名确定文件所在目录（及主机），已经在机器中具体储存位置。

索引文件：

索引文件由主文件和索引表构成。

主文件：文件本身；

索引表：在文件本身外建立的一张表，它指明逻辑记录和物理记录之间的一一对应关系；

索引文件的建立：按输入记录的先后次序建立数据区和索引表。其中索引表中关键字是无序的。待全部记录输入完毕后对索引表进行排序，排序后的索引表和主文件一起就形成了索引文件。

索引表结构选择：

由于文件系统经常需要更新或发生变动，以及考虑到访问的性能。选取 B 树作为其结构。理由如下：

① 插入、删除方便

② 本身是层次结构，无须建立多级索引

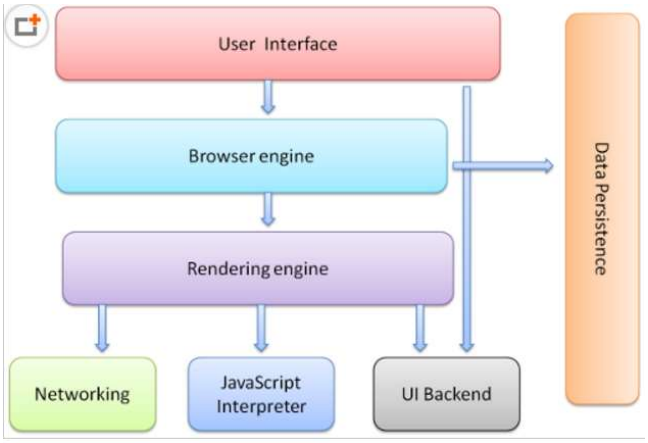
③ 建立索引表的过程即为排序过程。

④ B 树的查找效率高。

浏览器对 DFS 的支持

浏览器组件构成：

大多数浏览器的组件构成如图：



在最底层的三个组件分别是网络，UI后端和js解释器。作用如下：

（1）网络：用来完成网络调用，例如http请求，它具有平台无关的接口，可以在不同平台上工作。
（2）UI后端：用来绘制类似组合选择框及对话框等基本组件，具有不特定于某个平台的通用接口，底层使用操作系统的用户接口。
（3）JS解释器：用来解释执行JS代码。

Javaspcript执行原理：

JavaScript 是一种动态、弱类型、基于原型的语言，可以通过浏览器可以直接执行。如下图示，JavaScript 由核心（ECMAScript）、文档对象模型（DOM）和浏览器对象模型（BOM）三部分组成。

当浏览器遇到<script> 标记的时候，浏览器会执行之间的 javascript 代码。嵌入的 js 代码是顺序执行的，每个脚本定义的全局变量和函数，都可以被后面执行的脚本所调用。
ECMAScript 是一个描述，定义了脚本语言的所有属性、方法和对象。其他语言也可以实现 ECMAScript 来作为功能的基准。
DOM（文档对象模型）是 HTML 和 XML 的应用程序接口（API）。DOM 将把整个页面规划成由节点层级构成的文档 IE 3.0 和 Netscape Navigator 3.0 提供了一种特性 - BOM（浏览器对象模型），可以对浏览器窗口进行访问和操作。使用 BOM，开发者可以移动窗口、改变状态栏中的文本以及执行其他与页面内容不直接相关的动作。。HTML 或 XML 页面的每个部分都是一个节点的衍生物。
文件操作支持：
Html 定义了<input type="file" /> 用于文件上传。因此可在网页上实现上传文件。
Javascript 很多对文件的操作：

CopyFile() 复制文件
CopyFolder() 复制目录
CreateFolder() 创建新目录
CreateTextFile() 生成一个文件
DeleteFile() 删除一个文件
DeleteFolder() 删除一个目录
DriveExists() 检验盘符是否存在
Drives 返回盘符的集合
FileExists() 检验文件是否存在
FolderExists 检验一个目录是否存在
GetAbsolutePathName() 取得一个文件的绝对路径
GetBaseName() 取得文件名
GetDrive() 取得盘符名
GetDriveName() 取得盘符名
GetExtensionName() 取得文件的后缀
GetFile() 生成文件对象
GetFileName() 取得文件名
GetFolder() 取得目录对象
GetParentFolderName 取得文件或目录的父目录名
GetSpecialFolder() 取得特殊的目录名
GetTempName() 生成一个临时文件对象
MoveFile() 移动文件
MoveFolder() 移动目录

参考文献

[3.3] HADOOP官方文档. Hadoop分布式文件系统：架构和设计

[3.4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. *The Google File System*

[3.5] Reed I S, Solomon G. *Polynomial Codes Over Certain Finite Fields[J]*

[3.6] Alexandros G. Dimakis, Vinod Prabhakaran, Vinod Prabhakaran, *Decentralized erasure codes for distributed networked storage*

[3.7] 贾军博, 谷建华, 朱靖飞, 等. *网络环境下基于Erasure Code的高可靠性存储体系设计[J]*