Document-1

Contents

Architecture：

整个系统包括三个角色：1.Client，2.NameNode，3.DataNode。三种不同的角色分别运行在不同的机器上，分别对应着项目中的三个接口 ISimpleDistributedFileSystem、INameNode 和 IDataNode，各个接口的详细介绍如下：

Client（ISimpleDistributedFileSystem）：

作为使用 SDFS 系统的终端，主要包含三个方法，分别对应 open、create 和 mkdir。1.open 返回一个输入流对象，2.create 返回输出流对象，3.mkdir 执行创建文件目录操作。

NameNode（INameNode）：

维护着整个 SDFS 系统的继承自 Entity 的 DirNode 和 FIleNode 对象关系等信息，负责响应 Client 发起的操作，包含五个方法，分别对应 open、create、close、mkdir 和 addBlock。

1.open 返回指定路径的 FIleNode 对象，其中单个的 FIleNode 包括多个 BlockInfo，根据 BlockInfo 对象可以寻找到单个块对应的所有备份即 LocatedBlock 对象列表，从而根据不同的 LocatedBlock 中的 inetAddress 和 blockNumber 寻找到对应的 DataNode 节点上的数据块；

2.create 返回指定路径下新建的 FileNode 对象，并且在对应的 DirNode 的 contents 中添加新的 FIleNode 文件对象；

3.close 关闭已经打开的指定路径的 FileNode，检查对应的元信息是否有更改，并将改动后的元信息同步到持久化的磁盘上；

4.mkdir 在指定的路径下，即指定的 DirNode 的 contents 中创建对应的 DirNode 目录对象；

5.addBlock 用于申请指定路径的文件对象存放数据的 block 单元，同时返回新申请到的 LocatedBlock 对象，并且在此路径文件对应的的 FIleNode 中对应的 BlockInfo 新增此 LocatedBlock。

DataNode（IDataNode）：

作为 SDFS 系统中存储数据的终端，单台机器上维护着多个 Block 数据块，提供 read 和 write 方法来读写对应 blockNumber 标识的数据块，并且在读写的过程中可能会涉及到偏移量等参数。

The Execution Flow：

ISimpleDistributedFileSystem：

1.open(String fileUri)，首先 Client 将 fileUri 传回给 NameNode 节点，然后 NameNode 从 root 目录对应的 DirNode 中的 contents 开始，以'/'为分隔符，依次解析 fileUri 字符串包含的各个 DirNode，直到检索到不含目录的目标文件 FileNode。在找到目标文件的 FileNode 对象后，NameNode 将 FileNode 序列化后传回给 Client，然后 Client 根据 NameNode 传回的 FileNode 生成 SDFSInputStream 作为返回值。

2. create(String fileUri)，首先 Client 将 fileUri 传回给 NameNode 节点，NameNode 执行 create 操作，类似上述的 open 操作中的层次性寻找到最后一个 DirNode 目录，然后在这个目录对象的 contents 中新增一个 FileNode，新增的 FileNode 默认通过 addBlock 方法获取了一个 BlockInfo 及备份系数倍的 LocatedBlock。最后，NameNode 把这个新增的 FileNode 序列化后返回给 Client，Client 根据 FileNode 生成 SDFSOutputStream 作为返回值。

3. mkdir(String fileUri)，首先 Client 将 fileUri 传回给 NameNode 节点，NameNode 执行 mkdir 操作，类似上述层次性的查找目录操作，然后在最后一个已存在的 DirNode 的 contents 中新建一个 DirNode 节点，并且设置新建的 DirNode 相关属性。

SDFSInputStream：

1. int read(byte[] b)，根据 FileNode 中的 BlockInfo 选取备份中的一个或多个 LocatedBlock，然后去寻找对应 DataNode 上的 Block，将 Block 中的内容读取到 byte 数组中。整个读取过程是流式的，

根据 BlockInfo 列表上的 LocatedBlock 备份，顺次寻找 DataNode 上的 Block 内容，流式填写到 byte 数组中。

2. close()，表示读取操作结束，读取操作不会改变 SDFS 系统中的数据内容，所以只需要丢弃掉 read 过程中的 cursor 等临时变量，释放内存。

SDFSOutputStream:

1. write(byte[] b)，根据 FileNode 中已有的 BlockInfo 列表，从前往后写入每个 BlockInfo 对应的 LocatedBlock 列表中的第一个，当第一个 LocatedBlock 写满后，同一个 BlockInfo 中的其他 LocatedBlock 会顺次从上一个 LocatedBlock 处获取数据进行复制操作，从而实现流式写入。当 BlockInfo 用尽后，如果仍有数据未写入，则需要向 NameNode 申请一组备份系数的 LocatedBlock，即进行 addBlock 操作，将新申请到的这组 LocatedBlock 列表添加到 BlockInfo 中。

2. close()，当写入操作结束后，会释放写入过程中创建的临时变量，并且会通知 NameNode 写入操作结束。在收到 close 信号及各个 LocatedBlock 的写入完成确认后，NameNode 会将更改后的 FileNode 等信息会进行相关的持久化操作，序列化后存储在本地的硬盘中。

Difficulty:

1.NameNode 管理 DirNode 和 FileNode 的层次结构，开发过程中需要提取字符串中不同的目录结构；

2. SDFSInputStream 的 read 操作需要考虑 byte 数组长度、LocatedBlock 是否可达等信息，需要谨慎地进行读取操作，防止读取失败；

3. SDFSOutputStream 的 write 操作需要考虑已有的 FileNode 中的 BlockInfo 是否充足，并且需要协调 LocatedBlock 之间的数据传输。当存储空间不足时，需要另外向 NameNode 申请新的 LocatedBlock，并且在 FileNode 的 BlockInfo 中添加这些新增的 LocatedBlock，并且需要注意同步 NameNode 中的 FIleNode 和本地的 FileNode。当同一 BlockInfo 中的一个 LocatedBlock 写入完成后，需要向后面的 LocatedBlock 顺次执行写入复制操作，需要考虑写入中的网络故障，提供相应的保障机制，确保写入成功。

Bottleneck:

1.多个 Client 同时发起读写请求时，NameNode 和 DataNode 需要实现 synchronize 的方法确保关键属性的访问安全，同时扩展相应的多进程实现方法；

2.写入操作中出现意外，需要提供回滚机制，确保数据能够成功写入，同时确保系统安全不崩溃；

3.多地部署过程中，通信网络的带宽、试验等状况可能成为数据传输的瓶颈，可能需要单独设计网络架构。

4.NameNode 需要通过心跳检测、report 等机制来扫描已有的 DataNode 存储状态，对于出现意外数据损失的 Block，需要及时从其他备份中恢复，确保数据的安全可靠。

5.需要考虑整个 SDFS 系统的可扩展性，考虑 NameNode 是否具有新增 DataNode 的接口，提高整个系统的灵活性。

文进

14307110274