Hadoop Distributed File System(HDFS)

TR Evan.Z.Zhu

Agenda

- Hadoop介绍
- HDFS架构和设计
- Hadoop部署
- Hadoop常用命令
- Java API demo

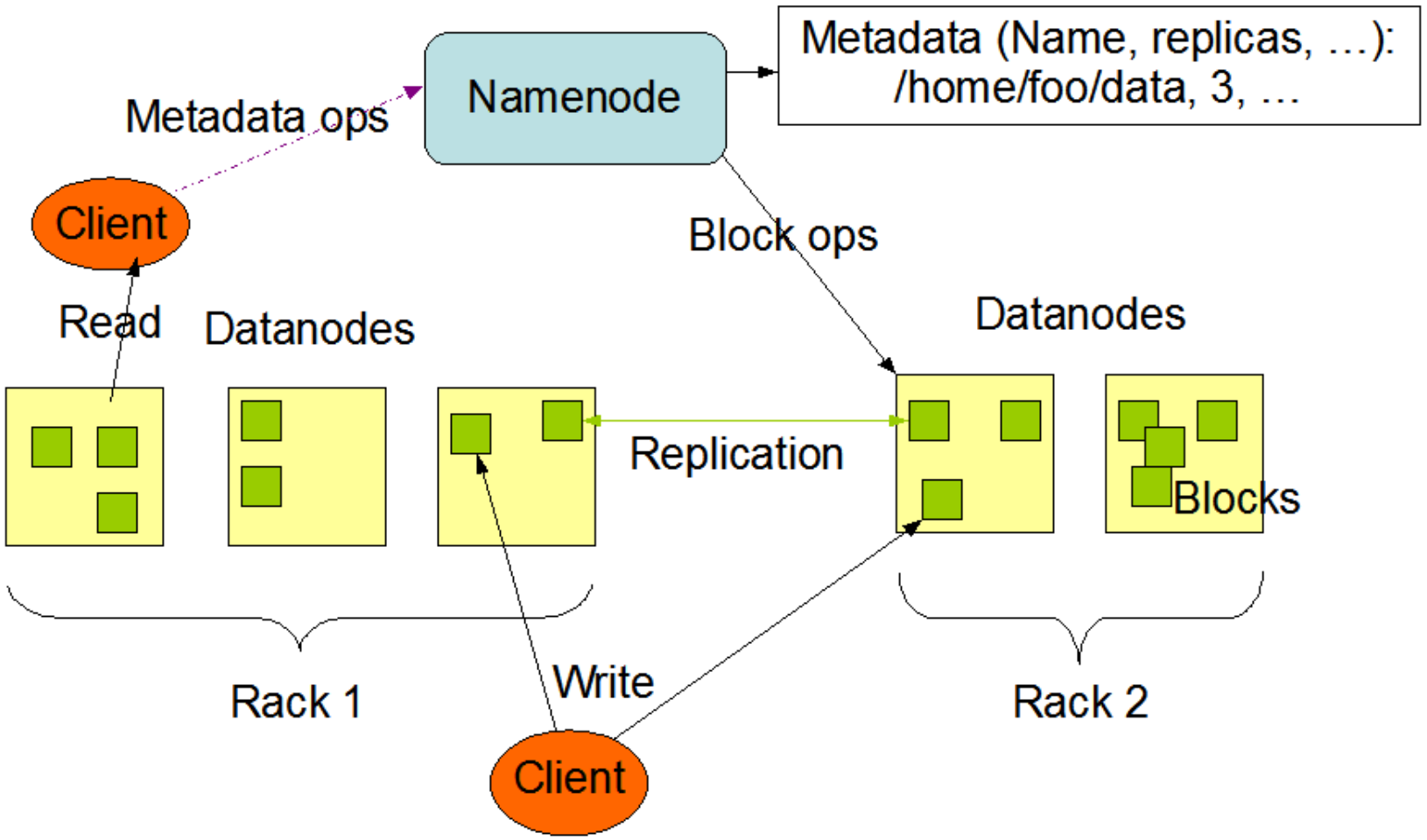
HDFS前提和设计目标

- 硬件错误
  - 硬件错误是常态而不是异常。面对的现实是构成系统的组件数目是巨大的，而且任一组件都有可能失效，即总有一部分HDFS的组件是不工作的。错误检测和快速、自动恢复是HDFS最核心的架构目标。
- 流式数据访问
  - 运行在HDFS上的应用和普通的应用不同，需要流式访问它们的数据集。HDFS的设计中更多的考虑到了数据批处理，而不是用户交互处理。比之数据访问的低延迟问题，更关键的在于数据访问的高吞吐量。POSIX标准设置的很多硬性约束对HDFS应用系统不是必需。为了提高数据的吞吐量，在一些关键方面对POSIX的语义做了一些修改。
- 大规模数据集
  - 运行在HDFS上的应用具有很大的数据集。HDFS上的一个典型文件大小一般都在GB到TB。因此，HDFS被调节以支持大文件存储。它应该能提供整体上高的数据传输带宽，能在一个集群里扩展到成千上万个节点。

HDFS前提和设计目标

- 简单的一致性模型
  - HDFS应用需要一个“一次写入多次读取”的文件访问模型。一个文件经过创建、写入和关闭之后就不需要改变。这一假设简化了数据一致性问题，并且是高吞吐量的数据访问成为可能。
- “移动计算比移动数据更划算”
  - 一个应用请求的计算，离它操作的数据越近就越高效，在数据达到海量级别的时候更是如此。因为这样就能降低网络阻塞的影响，提高系统数据的吞吐量。将计算移动到数据附近，比之将数据移动到应用所在显然更好。HDFS为应用提供了将它们移动到数据附近的接口。
- 异构软硬件平台间的可移植性
  - HDFS在设计的时候就考虑到平台的可移植性。这种特性方便了HDFS作为大规模数据应用平台的推广。

HDFS Architecture



Hadoop架构

- HDFS采用Master/Slave架构
  - HDFS集群是由一个NameNode和一定数目DataNode组成
- NameNode
  - 是一个中心服务器
  - 管理文件系统的名字空间(namespace)及客户端对文件的访问
  - 执行文件系统的名字空间操作，比如打开、管理、重命名文件和目录
  - 负责确定数据块到具体DataNode节点的映射
  - 单一NameNode结构大大简化了系统的结构
  - 是所有HDFS元数据的仲裁者和管理者，这样，用户数据永远不会流过NameNode

Hadoop架构

- DataNode
  - 一般是一个节点一个
  - 管理它所在节点上的存储
  - 负责处理文件系统客户端的读写请求
  - 在NameNode的统一调度下进行数据块的创建、删除和复制
- Editlog(日志)
  - 文件以及目录创建、删除日志
- Fsimage(镜像)
  - 整个文件系统的名字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为FsImage的文件
- Checkpoint(检查点)
  - NameNode启动时执行
  - 将Editlog中变化应用至读取的FsImage

Hadoop架构

- Secondary NameNode
  - 为了不断的将NameNode中的Editlog应用到FsImage中，减少下次NameNode的启动时间
    - NameNode响应Secondary NameNode请求，将Editlog推送给Secondary NameNode，开始重新写一个新的Editlog
    - Secondary NameNode收到来自NameNode的FsImage和Editlog
    - Secondary NameNode将FsIamge加载到内存，应用Editlog，并生成一个新的FsImage文件
    - Secondary NameNode将新的FsImage推送给NameNode
    - NameNode用新的FsImage取代旧的FsImage，在fstime文件中记下检查点发生的时间
  - 不是NameNode的备份

Hadoop特点

- 扩容能力(Scalable)

  能可靠存储和处理千兆字节数据(PB)
- 低成本(Economical)

  可以通过普通机器组成的服务器群来分发和处理数据
- 高效率(Efficient)

  通过分发数据，可以在数据所在的节点上并行地处理它们，这使得处理非常的快速
- 可靠性(Reliable)

  能自动维护数据的多份复制，并且在任务失败后能自动重新部署计算任务

HDFS几个设计特点

- Block的放置：默认不配置
  一个Block会根据配置会有多份备份，一般情况是三份。
- 心跳检测DataNode状态
  DataNode每间隔一定时间向NameNode发送状态；NameNode使用心跳侦测DataNode故障
- 数据复制
  DataNode失败、需要平衡DataNode的存储利用率和需要平衡DataNode数据交互压力
- 数据校验：CRC32
  在文件Block写的时候处理写入数据还会写入校验信息；在读取的时候需要校验后在读取
- NameNode是单点
  在失败的时候，任务处理信息会记录在本地文件系统和远端的文件系统中
- 数据管道性的写入
- 安全模式

Hadoop DataNode增加和删除

- dfs.hosts
  - 允许连入NameNode的DataNode清单
- dfs.hosts.exclude
  - 禁止连入NameNode的DataNode清单
- hadoop dfsadmin –refreshNodes
  - 批量增加和删除DataNode

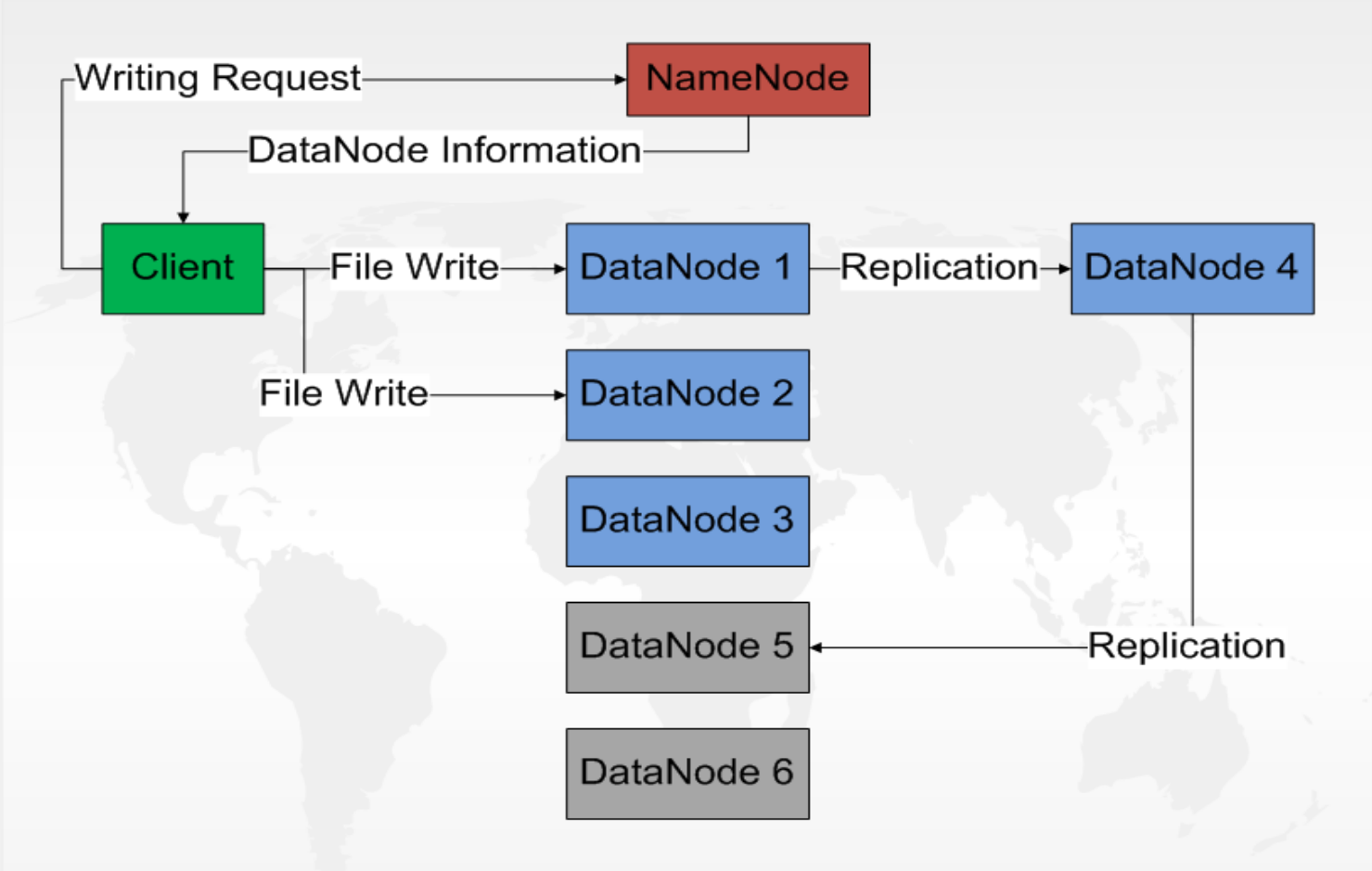
Hadoop DataNode增加

- dfs.hosts文件中增加ip, mapred.hosts也类似。Slave中增加NameNode的IP
- Hadoop dfsadmin –refreshNodes
- 启动新的DataNode和TaskTracker
- 如果mapred.host(NameNode)变化
  - 重新启动jobTracker
  - Mapred.jobtracker.restart.recover = true
  - 重新启动后将恢复运行JobTracker

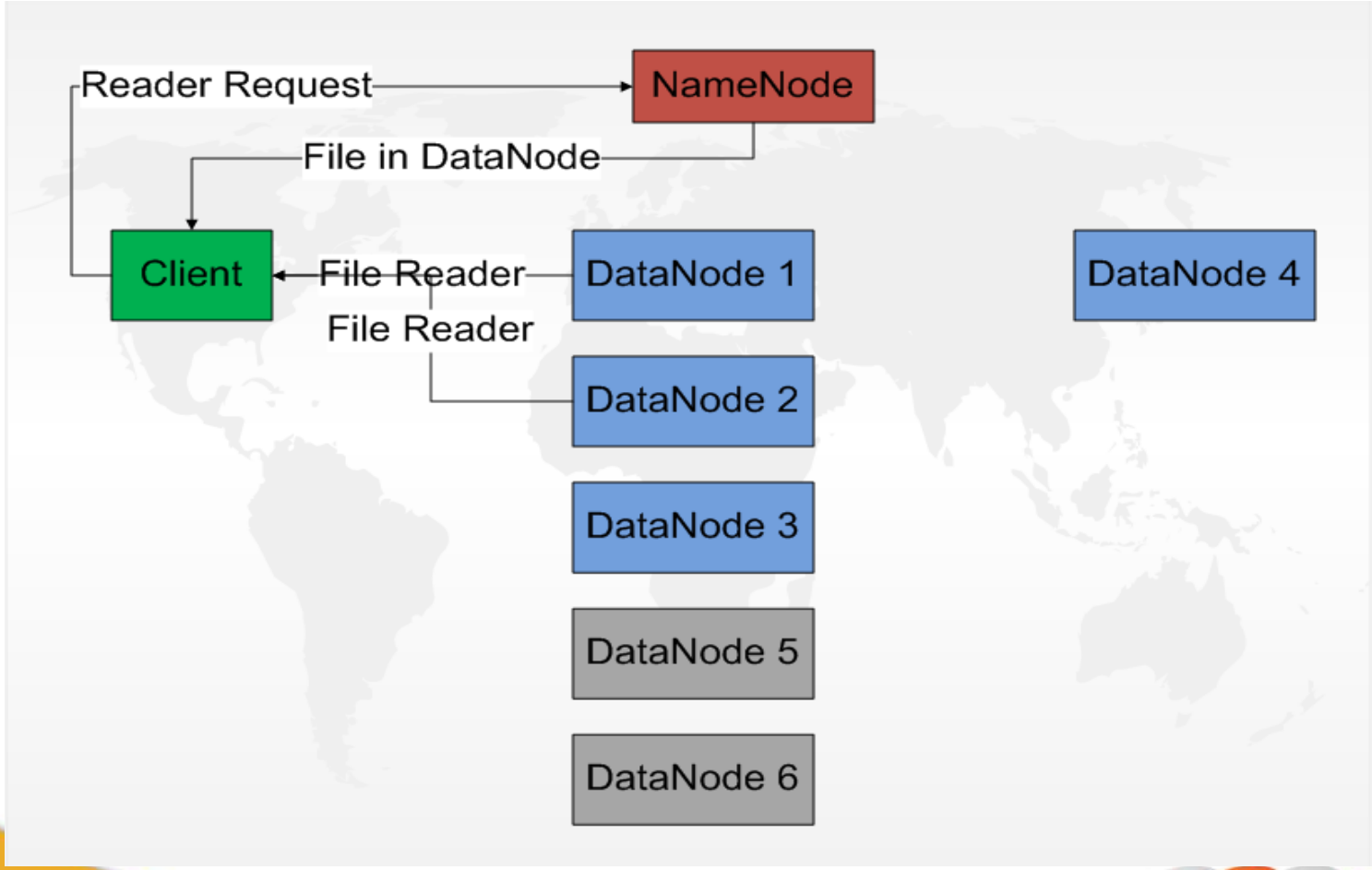
Hadoop文件读写

- Hadoop文件写入
  - Client向NameNode发起文件写入请求
  - NameNode根据文件大小和文件块配置情况，返回给Client它管理部分DataNode信息
  - Client将文件划分为多个Block，根据DataNode的地址信息，按顺序写入到每一个DataNode块中
- Hadoop文件读取
  - Client向NameNode发起文件读取请求
  - NameNode返回文件存储的DataNode的信息
  - Client从DataNode中读取文件信息
- Hadoop文件Block复制
  - NameNode发现部分文件的Block不符合最小复制数或部分DataNode失效
  - 通知DataNode相互复制Block
  - DataNode开始直接相互复制

Hadoop文件写


Writing Request
NameNode
DataNode Information
Client
File Write
DataNode 1
Replication
DataNode 4
File Write
DataNode 2
DataNode 3
DataNode 5
Replication
DataNode 6

Hadoop文件读



Hadoop部署类型

- 本地模式(Standalone)
  - 运行在同一个JVM，使用本地文件系统
- 伪分布模式(Pseudo-distributed)
  - 所有JVM进程运行在同一个操作系统中，每个JVM运行各自服务。使用HDFS存储，模拟Hadoop Cluster，便于程序开发和调试
- 完全分布模式(Full distributed)

Hadoop部署前置条件

- 操作系统(CentOS)安装
- 运行环境(Java)安装
  - JDK1.6.0
- SSH环境配置
  - 无密码访问

Hadoop部署

- 准备工作（Hadoop安装包）
  - Hadoop-0.20.203.0rc1.tar.gz
- 修改Hadoop配置文件
  - Conf/hadoop-env.sh
  - Conf/core-site.xml
  - Conf/hdfs-site.xml
  - Conf/mapred-site.xml
  - Bin/hadoop
- 设置系统环境变量
  - HADOOP_HOME/CLASSPATH/PATH
- 运行和启动Hadoop
  - Hadoop namenode –format
  - Start-all.sh
- 验证安装
  - Hadoop version

Hadoop常用命令

- Hadoop
- Hadoop namenode
- Hadoop fs
- Hadoop dfsadmin
- Hadoop balancer
- Hadoop jar
- Hadoop job

Hadoop namenode

- Hadoop namenode –format
- Hadoop namenode –rollback
- Hadoop namenode –upgrade
- Hadoop namenode -importCheckppint

Hadoop dfsadmin

- Hadoop dfsadmin –report
- Hadoop dfsadmin –safemode [leave|get|…]
- Hadoop dfsadmin –refreshNodes
- Hadoop dfsadmin –metasave filename

Hadoop fs

- Hadoop fs –put(上传文件)
- Hadoop fs –get(获取文件)
- Hadoop fs –cat(显示文件内容)
- Hadoop fs –copyFromLocal
- Hadoop fs –copyToLocal
- Hadoop fs –cp(拷贝文件)
- Hadoop fs –ls(列出文件)
- Hadoop fs –stat(列出文件统计信息)
- Hadoop fs –rm(删除文件)
- Hadoop fs –mv(移动文件)

Java API demo

- 获取HDFS上的文件
- 获取HDFS上的文件内容
- 向HDFS上写文件内容
- 在HDFS上创建和删除文件夹

Demo:获取HDFS上的文件

```
public static void getFile(String src, String dst) throws Exception{
    Configuration config = new Configuration();
    config.set("fs.default.name", "hdfs://127.0.0.1:9000/");
    FileSystem fs = FileSystem.get(config);
    Path srcPath = new Path(src);
    Path dstPath = new Path(dst);
    fs.copyToLocalFile(srcPath, dstPath);
}
```

Demo:获取HDFS上的文件内容

```
public static void readFile(String dst) throws Exception{
        Configuration config = new Configuration();
        config.set("fs.default.name", "hdfs://127.0.0.1:9000/");
        FileSystem fs = FileSystem.get(config);
        Path path = new Path(dst);
        if (fs.exists(path)){
                FSDataInputStream is = fs.open(path);
                FileStatus status = fs.getFileStatus(path);
                byte[] buffer = new byte[Integer.parseInt(String.valueOf(status.getLen()))];
                is.readFully(0, buffer);
                is.close();
                fs.close();
                String value = new String(buffer);
                System.out.println(buffer.length);
                System.out.println(value);
        }
        else{
                throw new Exception("The file is not found!");
        }
}
```

Demo:向HDFS上写文件内容

```
public static void writeFileContent(String dst) throws IOException{
        Configuration config = new Configuration();
        config.set("fs.default.name","hdfs://127.0.0.1:9000/");
        FileSystem fs = FileSystem.get(config);
        FSDataOutputStream dos = fs.create(new Path(dst));
        String utf8 = "UTF-8";
        dos.write("I am testing.\r\n".getBytes(utf8));
        dos.write("This is a demo.\r\n".getBytes(utf8));
        dos.write("Verity program correctly.\r\n".getBytes(utf8));
        Date dt = new Date();
        dos.write((dt.toString() + "\r\n").getBytes(utf8));
        dos.close();
        fs.close();
}
```

Demo:在HDFS上创建删除文件夹

```
public static void createDirectory(String dir) throws IOException{
        Configuration config = new Configuration();
        config.set("fs.default.name","hdfs://127.0.0.1:9000/");
        FileSystem fs = FileSystem.get(config);
        fs.mkdirs(new Path(dir));
        fs.close();
}
public static void deleteDirectory(String dir) throws IOException{
        Configuration config = new Configuration();
        config.set("fs.default.name","hdfs://127.0.0.1:9000/");
        FileSystem fs = FileSystem.get(config);
        fs.delete(new Path(dir), true);
        fs.close();
}
```

Thank you!