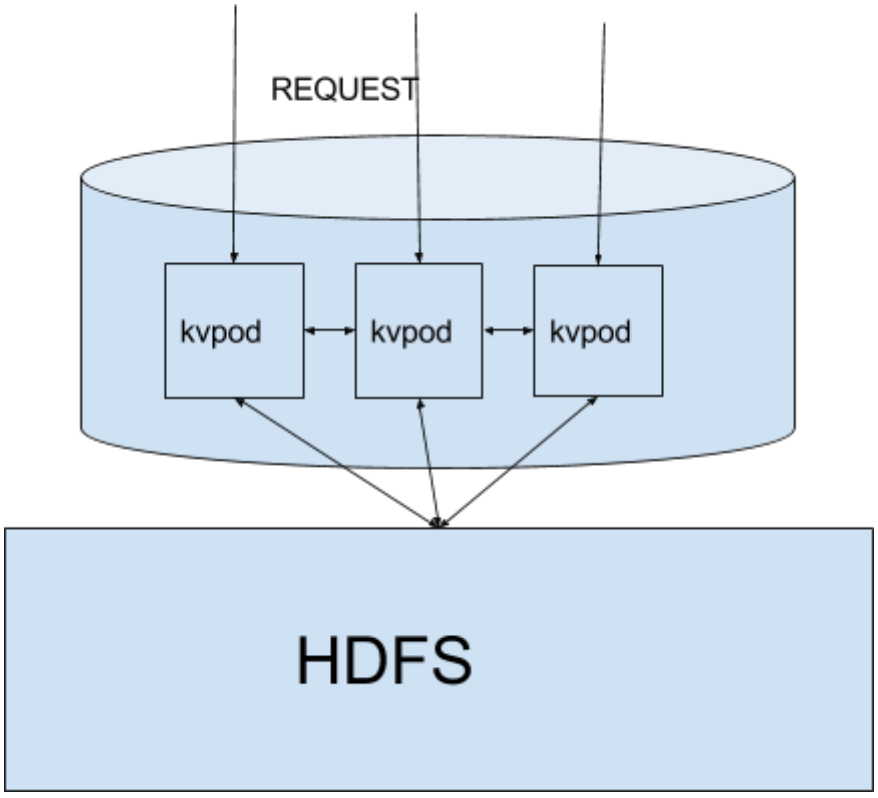
Key-Value Store 开发帮助

为了简化大家的工作量，我们提供了一个sdk来帮助大家完成作业，创建自己的Key-Value Store。下面，我将详细介绍sdk的使用，以及作业提交。 sdk可以在TSS下载。

作业介绍

首先，大家完成的key-value store，是一个基于hdfs的，分布式的key-value store。我们在测试的时候，会起三个java进程，分布在不同的机器上，为了方便，就把java 进程起名叫kvpod。pod大家可以理解为node。简而言之，就是多个kvpod协同合作，使用hdfs作底端存储，从而完成了整个kv-store的功能。



我们sdk的依赖的pom文件会放在tss上,大家把我们的pom放入自己的项目，在这基础上开发，不允许导入新的依赖。然后，大家注意，kvpod重启后，本地文件都会丢失。而我们在测试的时候，会涵盖重启pod的情况。

开发

接下来，要说明的是，sdk中我们提供了一个接口processor，如下图所示。为了完成key-value store，大家需要继承并实现这个接口。

```
public interface Processor {
 Map<String,String> get(String key);
 boolean put(String key, Map<String,String> value);
 boolean batchPut(Map<String, Map<String,String>> records);
 byte[] process(byte[] inupt);
}
```

如上图所示，processor接口里，有4个方法，这里详细介绍4个方法的功能。

get()方法，通过传进来的参数查询获取相应的数据并返回。这里的参数key，就是key-value store里面的key，显然，返回的map，就是查询到的value。

put()方法，将传进来的数据写入hdfs。当然，这里怎么做索引，怎么缓存都需要大家自己考虑，最终目标当然是提高性能。这里，说一下我们的数据格式，key是一个字符串，而value是一个map，可以表示为map<column, value>。例如，下面有一条数据，

name	age	tel
choke	18	12345678

那么我们的valueMap则是{name=choke，age=18, tel=12345678};

batchPut()方法，顾名思义就是批量的写入数据，这里接收的参数Map<String,Map<String,String>>records中的key，value，就是我们put()方法中接收的参数，key，value。

process()方法，当kvpod接收到其他的kvpod传来的消息时，调用的方法，用来完成各个kvpod间的交流。这里的输入输出都是byte[]，大家可以自由创建自己的方便灵活的消息格式。

同时有件事大家要注意，因为使用了反射来注入大家所实现的继承processor接口的class,因此这个类一定要有一个无参数的构造器。

下面是一个数据全部存在内存中的Demo。

```
import cn.helium.kvstore.processor.Processor;
import java.util.HashMap;
```

```java
import java.util.Map;

public class MockProcessor implements Processor {
 Map<String, Map<String,String>> store = new HashMap<>();
 @Override
 public Map<String,String> get(String key) {
   Map<String,String> record = store.get(key);
   return record;
 }

 @Override
 public synchronized boolean put(String key, Map<String,String>
value) {
   store.put(key,value);
   return true;
 }

 @Override
 public synchronized boolean batchPut(
        Map<String, Map<String, String>> records) {
   store.putAll(records);
   return true;
 }

 @Override
 public byte[] process(byte[] inupt) {
   System.out.println("receive info: "+new String(inupt));
   return "received!".getBytes();
 }

}
```

相关API

当你实现了这个processor接口后，整个key-value store就算完成了。下面介绍一些我们提供的大家可能会用到的api。

Class：cn.helium.kvstore.rpc.RpcServer
Method: public static int getRpcServerId()

我们会对各个kvpod进行标号，而大家通过RpcServer.getRpcServerId()即可获得当前kvpod的编号。

Class: cn.helium.kvstore.rpc.RpcClientFactory
Method：public static byte[] inform(int index, byte[] info)

大家可以通过RpcClientFactory.inform()方法方便地与向其他kvpod发送消息。这里的参数，index就是接受你发出的消息的kvpod的编号,而info自然就是要发送的消息。这里，大家要注意的是，当与目标kvpod的网络连接出现问题，连不通的时候，会抛出IOException。大家要处理一下这个异常。

Class: cn.helium.kvstore.common.KvStoreConfig
Method: public static int getServersNum()

同时，当前kvstore的kvpod数量可以通KvStoreConfig.getServersNum()获得。

Class: cn.helium.kvstore.common.KvStoreConfig
Method: public static String getHdfsUrl()

当然，大家做的是基于hdfs的，key-value store，大家可以通KvStoreConfig.getHdfsUrl()获得hdfs的url。

本地调试

接下来介绍一下大家写完后，本地调试的方法。大家将自己的程序打包后，（依赖也要一并打包）通过java -cp yours.jar:ourSdk.jar:dependency.jar

cn.helium.kvstore.main.Main [--paramName paramValue]可以将自己的kvpod起起来。下面介绍下相关参数。

variable	paramNames	type	default
restPort	--restful-server-port	int	8500
kvPodId	--kvpod-id	int	0
processorClass	--processor-class	String	cn.helium.kvstore.processor.MockProcessor。
hdfsPort	--hdfs-port	int	9000
hdfsHost	--hdfs-host	String	127.0.0.1
kvPodsHosts	--kvpods-hosts	String	localhost,localhost,localhost
kvPodsRpcPorts	--kvpods-rpc-ports	String	8090,8091,8092

如上表所示，我们提供了一些参数，方便大家在本地测试自己的程序。下面对于各个参数进行说明

variable	description
restPort	配置kvpod启动时，restful service监听的端口
kvPodId	配置当前要启动的kvpod的编号
processorClass	继承processor接口实现的类（包名+类名）
hdfsPort	配置使用的hdfs的端口
hdfsHost	配置使用的hdfs的ip
kvPodHosts	kvStore中各个kvpod的ip

kvPodRpcPorts	各个kvpod上的rpc service端口，

使用这些参数，大家就可以在测试一下自己的程序。

Example：

java -cp yours.jar:ourSdk.jar cn.helium.kvstore.main.Main --kvPodId 2

注意点：

1. kvPodsHosts，与kvPodsRpcPorts，是整个kvstore中所有的kvPod的ip和端口，按kvPodId从小到大列出，以英文逗号隔开。例如，我们2个节点的kvpod,kvPodId 为0的节点ip为10.19.139.5, kvPodId为1 的节点ip为10.19.139.6，那么kvPodHosts配置应该是：----kvpods-hosts 10.19.139.5,10.19.139.5
2. 配置的参数，kvPodId不能大于等于kvPod的数量。例如当kvstore里有3个kvpod的时候，3个kvpod的kvPodId只能是0，1，2；

Restful Service 接口

下面是我们提供的restful接口。需要注意的是，这里我们只是提供了一个接口，接到请求后就直接调用了大家实现processor的类的相应方法，所以读写中的异常需要大家自己hold住。然后就是每个kvpod都有自己的restful接口，我们测试的时会随机调用某个kvpod的restful接口。

Put

DESCRIPTION	向kvstore 写入数据
METHOD	POST
URL	/process
URL Params	None
DATA PARAMS	{ “key”: “100”, “value”: {

	“column1” : “value1”, “column2” : “value2” } }
RESPONSE	Code:200
	Content: “success!” || ”fail”

Get

DESCRIPTION	向kvstore 读取数据
METHOD	GET
URL	/process
URL Params	key=[string]
DATA PARAMS	None
RESPONSE	Code:200
	Content: “{ column1=value1,column2=value2}” || “find nothing.”

BatchPut

DESCRIPTION	批量向kvstore 写入数据
METHOD	POST
URL	/batchProcess
URL Params	None

DATA PARAMS	{
	“100”: {
	“column1” : ”value1”,
	“column2” : “value2”

	},
	“110”: {
	“column1” : ”value3”,
	“column2” : ”value4”
	}
	}

RESPONSE	Code:200
	Content: “success!” || ”fail”

TestCase

我们一共准备了五条testCase，测试大家的程序。我们在部署大家的程序的时候，会限定资源。具体如下：

1. 对于每个kvpod，我们会给8G内存，1个core的cpu
2. hdfs一共会起3个datanode,1个namenode,每个datanode有4G内存，1个core的cpu，以及40G的存储空间。namenode 有4G内存，1个core的cpu。

接下来给大家说一下我们的testcase，我们测试的时候，主要是测试功能性，会向kvstore读写数据。具体来说，会先写入一定数量的数据，然后会重启所有的kvpod，再去尝试读取数据，看看能否读到，读到的数据是否正确。请注意：重启pod后，所有的local disk以及 内存都会丢失。但是，五条case在1个半小时内希望跑完。超过1个半小时，我们会终止任务。

具体case如下：

1. 向kvstore写1万条数据，记录写数据的速度，然后我们会重启所有的kvpod，测试能否读取到正确的数据。
2. 我们会断掉某两个kvpod之间的网络连接，让它们不能相互通信，然后测试case1。
3. 我们会干脆杀掉一个kvpod，然后测试case1。
4. 向kvstore写100万条数据，记录写数据的速度，重启kvpod，测试能否读取到正确的数据，并测试读取数据的速度。
5. 同case2，这次是1亿条数据。

Testcase	数据量	异常	测试	分数
1	1万		正确读写	20
2	1万	节点间通信异常	正确读写	20
3	1万	节点挂掉	正确读写	20
4	100万		正确读写	20
5	1亿		正确读写	20