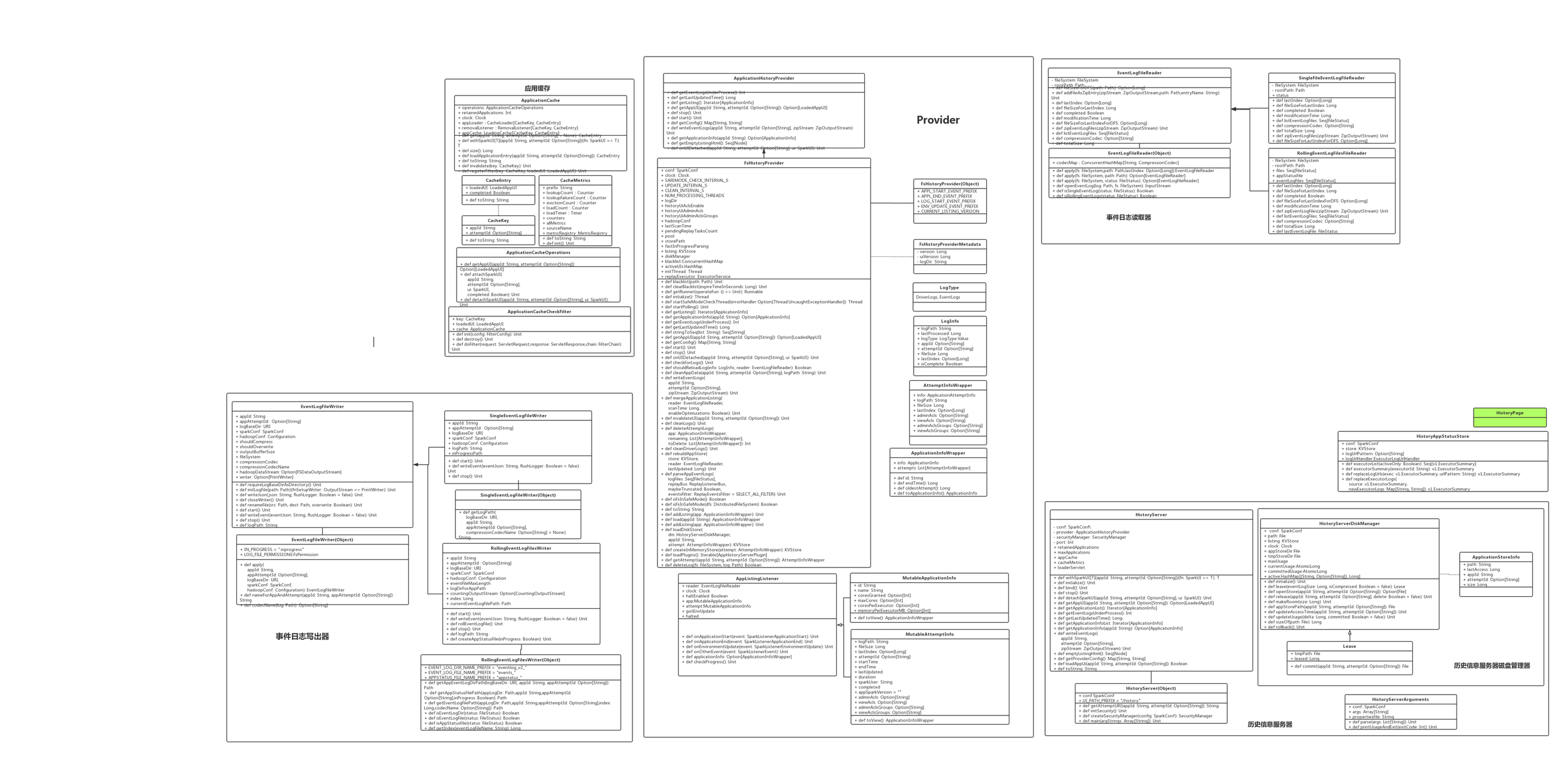
应用缓存
ApplicationCache
+ operations: ApplicationCacheOperations
+ retainedApplications: Int
+ clock: Clock
+ appLoader : CacheLoader[CacheKey, CacheEntry]
+ removalListener : RemovalListener[CacheKey, CacheEntry]
+ appCache: LoadingCache[CacheKey, CacheEntry]
+ def this(conf: SparkConf, operations: ApplicationCacheOperations, clock: Clock)
+ def withSparkUI[T](appId: String, attemptId: Option[String])(fn: SparkUI => T): T
+ def size(): Long
+ def loadApplicationEntry(appId: String, attemptId: Option[String]): CacheEntry
+ def toString: String
+ def invalidate(key: CacheKey): Unit
+ def registerFilter(key: CacheKey, loadedUI: LoadedAppUI): Unit
CacheEntry
+ loadedUI: LoadedAppUI
+ completed: Boolean
+ def toString: String
CacheMetrics
+ prefix: String
+ lookupCount : Counter
+ lookupFailureCount : Counter
+ evictionCount : Counter
+ loadCount : Counter
+ loadTimer : Timer
+ counters
+ allMetrics
+ sourceName
+ metricRegistry: MetricRegistry
+ def toString: String
+ def init(): Unit
CacheKey
+ appId: String
+ attemptId: Option[String]
+ def toString: String
ApplicationCacheOperations
+ def getAppUI(appId: String, attemptId: Option[String]): Option[LoadedAppUI]
+ def attachSparkUI(
appId: String,
attemptId: Option[String],
ui: SparkUI,
completed: Boolean): Unit
+ def detachSparkUI(appId: String, attemptId: Option[String], ui: SparkUI): Unit
ApplicationCacheCheckFilter
+ key: CacheKey
+ loadedUI: LoadedAppUI
+ cache: ApplicationCache
+ def init(config: FilterConfig): Unit
+ def destroy(): Unit
+ def doFilter(request: ServletRequest,response: ServletResponse,chain: FilterChain): Unit
Provider
ApplicationHistoryProvider
+ def getEventLogsUnderProcess(): Int
+ def getLastUpdatedTime(): Long
+ def getListing(): Iterator[ApplicationInfo]
+ def getAppUI(appId: String, attemptId: Option[String]): Option[LoadedAppUI]
+ def stop(): Unit
+ def start(): Unit
+ def getConfig(): Map[String, String]
+ def writeEventLogs(appId: String, attemptId: Option[String], zipStream: ZipOutputStream): Unit
+ def getApplicationInfo(appId: String): Option[ApplicationInfo]
+ def getEmptyListingHtml(): Seq[Node]
+ def onUIDetached(appId: String, attemptId: Option[String], ui: SparkUI): Unit
FsHistoryProvider
+ conf: SparkConf
+ clock: Clock
+ SAFEMODE_CHECK_INTERVAL_S
+ UPDATE_INTERVAL_S
+ CLEAN_INTERVAL_S
+ NUM_PROCESSING_THREADS
+ logDir
+ historyUiAclsEnable
+ historyUiAdminAcls
+ historyUiAdminAclsGroups
+ hadoopConf
+ lastScanTime
+ pendingReplayTasksCount
+ pool
+ storePath
+ fastInProgressParsing
+ listing: KVStore
+ diskManager
+ blacklist:ConcurrentHashMap
+ activeUIs:HashMap
+ initThread: Thread
+ replayExecutor: ExecutorService
+ def blacklist(path: Path): Unit
+ def clearBlacklist(expireTimeInSeconds: Long): Unit
+ def getRunner(operateFun: () => Unit): Runnable
+ def initialize(): Thread
+ def startSafeModeCheckThread(errorHandler:Option[Thread.UncaughtExceptionHandler]): Thread
+ def startPolling(): Unit
+ def getListing(): Iterator[ApplicationInfo]
+ def getApplicationInfo(appId: String): Option[ApplicationInfo]
+ def getEventLogsUnderProcess(): Int
+ def getLastUpdatedTime(): Long
+ def stringToSeq(list: String): Seq[String]
+ def getAppUI(appId: String, attemptId: Option[String]): Option[LoadedAppUI]
+ def getConfig(): Map[String, String]
+ def start(): Unit
+ def stop(): Unit
+ def onUIDetached(appId: String, attemptId: Option[String], ui: SparkUI): Unit
+ def checkForLogs(): Unit
+ def shouldReloadLog(info: LogInfo, reader: EventLogFileReader): Boolean
+ def cleanAppData(appId: String, attemptId: Option[String], logPath: String): Unit
+ def writeEventLogs(
appId: String,
attemptId: Option[String],
zipStream: ZipOutputStream): Unit
+ def mergeApplicationListing(
reader: EventLogFileReader,
scanTime: Long,
enableOptimizations: Boolean): Unit
+ def invalidateUI(appId: String, attemptId: Option[String]): Unit
+ def cleanLogs(): Unit
+ def deleteAttemptLogs(
app: ApplicationInfoWrapper,
remaining: List[AttemptInfoWrapper],
toDelete: List[AttemptInfoWrapper]): Int
+ def cleanDriverLogs(): Unit
+ def rebuildAppStore(
store: KVStore,
reader: EventLogFileReader,
lastUpdated: Long): Unit
+ def parseAppEventLogs(
logFiles: Seq[FileStatus],
replayBus: ReplayListenerBus,
maybeTruncated: Boolean,
eventsFilter: ReplayEventsFilter = SELECT_ALL_FILTER): Unit
+ def isFsInSafeMode(): Boolean
+ def isFsInSafeMode(dfs: DistributedFileSystem): Boolean
+ def toString: String
+ def addListing(app: ApplicationInfoWrapper): Unit
+ def load(appId: String): ApplicationInfoWrapper
+ def addListing(app: ApplicationInfoWrapper): Unit
+ def loadDiskStore(
dm: HistoryServerDiskManager,
appId: String,
attempt: AttemptInfoWrapper): KVStore
+ def createInMemoryStore(attempt: AttemptInfoWrapper): KVStore
+ def loadPlugins(): Iterable[AppHistoryServerPlugin]
+ def getAttempt(appId: String, attemptId: Option[String]): AttemptInfoWrapper
+ def deleteLog(fs: FileSystem, log: Path): Boolean
FsHistoryProvider(Object)
+ APPL_START_EVENT_PREFIX
+ APPL_END_EVENT_PREFIX
+ LOG_START_EVENT_PREFIX
+ ENV_UPDATE_EVENT_PREFIX
+ CURRENT_LISTING_VERSION
FsHistoryProviderMetadata
- version: Long
- uiVersion: Long
- logDir: String
LogType
DriverLogs, EventLogs
LogInfo
+ logPath: String
+ lastProcessed: Long
+ logType: LogType.Value
+ appId: Option[String]
+ attemptId: Option[String]
+ fileSize: Long
+ lastIndex: Option[Long]
+ isComplete: Boolean
AttemptInfoWrapper
+ info: ApplicationAttemptInfo
+ logPath: String
+ fileSize: Long
+ lastIndex: Option[Long]
+ adminAcls: Option[String]
+ viewAcls: Option[String]
+ adminAclsGroups: Option[String]
+ viewAclsGroups: Option[String]
ApplicationInfoWrapper
+ info: ApplicationInfo
+ attempts: List[AttemptInfoWrapper]
+ def id: String
+ def endTime(): Long
+ def oldestAttempt(): Long
+ def toApplicationInfo(): ApplicationInfo
事件日志读取器
EventLogFileReader
- fileSystem: FileSystem
- rootPath: Path
- lastIndex: Option[Long]
+ def addFileAsZipEntry(zipStream: ZipOutputStream,path: Path,entryName: String): Unit
+ def lastIndex: Option[Long]
+ def fileSizeForLastIndex: Long
+ def completed: Boolean
+ def modificationTime: Long
+ def fileSizeForLastIndexForDFS: Option[Long]
+ def zipEventLogFiles(zipStream: ZipOutputStream): Unit
+ def listEventLogFiles: Seq[FileStatus]
+ def compressionCodec: Option[String]
+ def totalSize: Long
EventLogFileReader(Object)
+ codecMap : ConcurrentHashMap[String, CompressionCodec]
+ def apply(fs: FileSystem,path: Path,lastIndex: Option[Long]):EventLogFileReader
+ def apply(fs: FileSystem, path: Path): Option[EventLogFileReader]
+ def apply(fs: FileSystem, status: FileStatus): Option[EventLogFileReader]
+ def openEventLog(log: Path, fs: FileSystem): InputStream
+ def isSingleEventLog(status: FileStatus): Boolean
+ def isRollingEventLogs(status: FileStatus): Boolean
SingleFileEventLogFileReader
- fileSystem: FileSystem
- rootPath: Path
+ status
+ def lastIndex: Option[Long]
+ def fileSizeForLastIndex: Long
+ def completed: Boolean
+ def modificationTime: Long
+ def listEventLogFiles: Seq[FileStatus]
+ def compressionCodec: Option[String]
+ def totalSize: Long
+ def zipEventLogFiles(zipStream: ZipOutputStream): Unit
+ def fileSizeForLastIndexForDFS: Option[Long]
RollingEventLogFilesFileReader
- fileSystem: FileSystem
- rootPath: Path
+ files: Seq[FileStatus]
+ appStatusFile
+ eventLogFiles: Seq[FileStatus]
+ def lastIndex: Option[Long]
+ def fileSizeForLastIndex: Long
+ def completed: Boolean
+ def fileSizeForLastIndexForDFS: Option[Long]
+ def modificationTime: Long
+ def zipEventLogFiles(zipStream: ZipOutputStream): Unit
+ def listEventLogFiles: Seq[FileStatus]
+ def compressionCodec: Option[String]
+ def totalSize: Long
+ def lastEventLogFile: FileStatus
事件日志写出器
EventLogFileWriter
+ appId: String
+ appAttemptId : Option[String]
+ logBaseDir: URI
+ sparkConf: SparkConf
+ hadoopConf: Configuration
+ shouldCompress
+ shouldOverwrite
+ outputBufferSize
+ fileSystem
+ compressionCodec
+ compressionCodecName
+ hadoopDataStream: Option[FSDataOutputStream]
+ writer: Option[PrintWriter]
+ def requireLogBaseDirAsDirectory(): Unit
+ def initLogFile(path: Path)(fnSetupWriter: OutputStream => PrintWriter): Unit
+ def writeJson(json: String, flushLogger: Boolean = false): Unit
+ def closeWriter(): Unit
+ def renameFile(src: Path, dest: Path, overwrite: Boolean): Unit
+ def start(): Unit
+ def writeEvent(eventJson: String, flushLogger: Boolean = false): Unit
+ def stop(): Unit
+ def logPath: String
EventLogFileWriter(Object)
+ IN_PROGRESS = ".inprogress"
+ LOG_FILE_PERMISSIONS:FsPermission
+ def apply(
appId: String,
appAttemptId: Option[String],
logBaseDir: URI,
sparkConf: SparkConf,
hadoopConf: Configuration): EventLogFileWriter
+ def nameForAppAndAttempt(appId: String, appAttemptId: Option[String]): String
+ def codecName(log: Path): Option[String]
SingleEventLogFileWriter
+ appId: String
+ appAttemptId : Option[String]
+ logBaseDir: URI
+ sparkConf: SparkConf
+ hadoopConf: Configuration
+ inProgressPath
+ def start(): Unit
+ def writeEvent(eventJson: String, flushLogger: Boolean = false): Unit
+ def stop(): Unit
SingleEventLogFileWriter(Object)
+ def getLogPath(
logBaseDir: URI,
appId: String,
appAttemptId: Option[String],
compressionCodecName: Option[String] = None): String
RollingEventLogFilesWriter
+ appId: String
+ appAttemptId : Option[String]
+ logBaseDir: URI
+ sparkConf: SparkConf
+ hadoopConf: Configuration
+ eventFileMaxLength
+ logDirForAppPath
+ countingOutputStream: Option[CountingOutputStream]
+ index: Long
+ currentEventLogFilePath: Path
+ def start(): Unit
+ def writeEvent(eventJson: String, flushLogger: Boolean = false): Unit
+ def rollEventLogFile(): Unit
+ def stop(): Unit
+ def logPath: String
+ def createAppStatusFile(inProgress: Boolean): Unit
RollingEventLogFilesWriter(Object)
+ EVENT_LOG_DIR_NAME_PREFIX = "eventlog_v2_"
+ EVENT_LOG_FILE_NAME_PREFIX = "events_"
+ APPSTATUS_FILE_NAME_PREFIX = "appstatus_"
+ def getAppEventLogDirPath(logBaseDir: URI, appId: String, appAttemptId: Option[String]): Path
+ def getAppStatusFilePath(appLogDir: Path,appId: String,appAttemptId: Option[String],inProgress: Boolean): Path
+ def getEventLogFilePath(appLogDir: Path,appId: String,appAttemptId: Option[String],index: Long,codecName: Option[String]): Path
+ def isEventLogDir(status: FileStatus): Boolean
+ def isEventLogFile(status: FileStatus): Boolean
+ def isAppStatusFile(status: FileStatus): Boolean
+ def getIndex(eventLogFileName: String): Long
AppListingListener
+ reader: EventLogFileReader
+ clock: Clock
+ hadEnabled: Boolean
+ appMutableApplicationInfo
+ attempt:MutableApplicationInfo
+ gotEnvUpdate
+ halted
+ def onApplicationStart(event: SparkListenerApplicationStart): Unit
+ def onApplicationEnd(event: SparkListenerApplicationEnd): Unit
+ def onEnvironmentUpdate(event: SparkListenerEnvironmentUpdate): Unit
+ def onOtherEvent(event: SparkListenerEvent): Unit
+ def applicationInfo: Option[ApplicationInfoWrapper]
+ def checkProgress(): Unit
MutableApplicationInfo
+ id: String
+ name: String
+ coresGranted: Option[Int]
+ maxCores: Option[Int]
+ coresPerExecutor: Option[Int]
+ memoryPerExecutorMB: Option[Int]
+ def toView(): ApplicationInfoWrapper
MutableAttemptInfo
+ logPath: String
+ fileSize: Long
+ lastIndex: Option[Long]
+ attemptId: Option[String]
+ startTime
+ endTime
+ lastUpdated
+ duration
+ sparkUser: String
+ completed
+ appSparkVersion = ""
+ adminAcls: Option[String]
+ viewAcls: Option[String]
+ adminAclsGroups: Option[String]
+ viewAclsGroups: Option[String]
+ def toView(): ApplicationInfoWrapper
HistoryPage
HistoryAppStatusStore
+ conf: SparkConf
+ store: KVStore
+ logUrlPattern: Option[String]
+ logUrlHandler:ExecutorLogUrlHandler
+ def executorList(activeOnly: Boolean): Seq[v1.ExecutorSummary]
+ def executorSummary(executorId: String): v1.ExecutorSummary
+ def replaceLogUrls(exec: v1.ExecutorSummary, urlPattern: String): v1.ExecutorSummary
+ def replaceExecutorLogs(
source: v1.ExecutorSummary,
newExecutorLogs: Map[String, String]): v1.ExecutorSummary
HistoryServer
- conf: SparkConf\
- provider: ApplicationHistoryProvider
- securityManager: SecurityManager
- port: Int
+ retainedApplications
+ maxApplications
+ appCache
+ cacheMetrics
+ loaderServlet
+ def withSparkUI[T](appId: String, attemptId: Option[String])(fn: SparkUI => T): T
+ def initialize(): Unit
+ def bind(): Unit
+ def stop(): Unit
+ def detachSparkUI(appId: String, attemptId: Option[String], ui: SparkUI): Unit
+ def getAppUI(appId: String, attemptId: Option[String]): Option[LoadedAppUI]
+ def getApplicationList(): Iterator[ApplicationInfo]
+ def getEventLogsUnderProcess(): Int
+ def getLastUpdatedTime(): Long
+ def getApplicationInfoList: Iterator[ApplicationInfo]
+ def getApplicationInfo(appId: String): Option[ApplicationInfo]
+ def writeEventLogs(
appId: String,
attemptId: Option[String],
zipStream: ZipOutputStream): Unit
+ def emptyListingHtml(): Seq[Node]
+ def getProviderConfig(): Map[String, String]
+ def loadAppUi(appId: String, attemptId: Option[String]): Boolean
+ def toString: String
HistoryServer(Object)
+ conf:SparkConf
+ UI_PATH_PREFIX = "/history"
+ def getAttemptURI(appId: String, attemptId: Option[String]): String
+ def initSecurity(): Unit
+ def createSecurityManager(config: SparkConf): SecurityManager
+ def main(argStrings: Array[String]): Unit
HistoryServerDiskManager
+ conf: SparkConf
+ path: File
+ listing: KVStore
+ clock: Clock
+ appStoreDir:File
+ tmpStoreDir:File
+ maxUsage
+ currentUsage:AtomicLong
+ committedUsage:AtomicLong
+ active:HashMap[String, Option[String], Long]
+ def initialize(): Unit
+ def lease(eventLogSize: Long, isCompressed: Boolean = false): Lease
+ def openStore(appId: String, attemptId: Option[String]): Option[File]
+ def release(appId: String, attemptId: Option[String], delete: Boolean = false): Unit
+ def makeRoom(size: Long): Unit
+ def appStorePath(appId: String, attemptId: Option[String]): File
+ def updateAccessTime(appId: String, attemptId: Option[String]): Unit
+ def updateUsage(delta: Long, committed: Boolean = false): Unit
+ def sizeOf(path: File): Long
+ def rollback(): Unit
Lease
+ tmpPath: File
+ leased: Long
+ def commit(appId: String, attemptId: Option[String]): File
ApplicationStoreInfo
+ path: String
+ lastAccess: Long
+ appId: String
+ attemptId: Option[String]
+ size: Long
HistoryServerArguments
+ conf: SparkConf
+ args: Array[String]
+ propertiesFile: String
+ def parse(args: List[String]): Unit
+ def printUsageAndExit(exitCode: Int): Unit
历史信息服务器
历史信息服务器磁盘管理器