Confluent笔记

# 脚本

目录： ./bin

## confluent

### 脚本的用法

|  |
| --- |
| usage() {  cat <<EOF  ${command\_name}: A command line interface to manage Confluent services  Usage: ${command\_name} <command> [<subcommand>] [<parameters>]  These are the available commands:  acl Specify acl for a service.  config Configure a connector.  current Get the path of the data and logs of the services managed by the current confluent run.  destroy Delete the data and logs of the current confluent run.  list List available services.  load Load a connector.  log Read or tail the log of a service.  start Start all services or a specific service along with its dependencies  status Get the status of all services or the status of a specific service along with its dependencies.  stop Stop all services or a specific service along with the services depending on it.  top Track resource usage of a service.  unload Unload a connector.  '${command\_name} help' lists available commands. See '${command\_name} help <command>' to read about a  specific command.  EOF  exit 0  } |

./bin/confluent 脚本是Confluent 服务管理的命令行接口，包含的可执行命令在commands 和 enterprise\_commands 变量的定义中。

另外提供了help命令，可以查看该命令行接口及其支持的各个具体命令的用法，对应函数名为command\_usage。对应个各个具体的命令和用法参考下面章节。

## 目录

设 CONFLUENT\_HOME 为当前Confluent 部署目录。

目录布局：

* $CONFLUENT\_HOME/bin
* $CONFLUENT\_HOME/etc
* tmp\_dir=/tmp/
* confluent\_current\_dir ： 当前 confluent 所有数据存放的根目录，默认为tmp\_dir

|  |  |  |  |
| --- | --- | --- | --- |
| 目录 | 默认 | 备注 |  |
| confluent\_bin | $CONFLUENT\_HOME/bin |  |  |
| confluent\_home | $CONFLUENT\_HOME |  |  |
| confluent\_conf | $CONFLUENT\_HOME/etc |  |  |
| tmp\_dir | $TMPDIR 或 /tmp/ |  |  |
| confluent\_current\_dir | CONFLUENT\_CURRENT 或$TMPDIR | Confluent工作目录所在从父目录 |  |

目录的环境变量：

* TMPDIR
* CONFLUENT\_CURRENT

## 服务

对应 services 和 rev\_ services 变量，包含的服务：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 服务名 | 说明 | 配置文件 | 备注 |  |
| "zookeeper" |  |  |  |  |
| "kafka" |  |  |  |  |
| "schema-registry" |  |  |  |  |
| "kafka-rest" |  |  |  |  |
| "connect" |  |  |  |  |

服务层面的管理

1. status\_service
2. start\_service
3. config\_service
4. stop\_service
5. start\_or\_stop\_service ：
6. service\_exists

每个具体服务层面的步骤

1. start\_X
2. config\_X
3. export\_X
4. stop\_X
5. wait\_X

启动服务的流程：

|  |
| --- |
| start\_service() {  local service="${1}"  local start\_command="${2}"  local service\_dir="${confluent\_current}/${service}"  is\_running "${service}" "false" \  && echo "${service} is already running. Try restarting if needed"\  && return 0  mkdir -p "${service\_dir}"  config\_"${service}"  echo "Starting ${service}"  # TODO: decide whether to persist logs on stdout / stderr between runs.  ${start\_command} "${service\_dir}/${service}.properties" \  2> "${service\_dir}/${service}.stderr" \  1> "${service\_dir}/${service}.stdout" &  echo $! > "${service\_dir}/${service}.pid"  local service\_pid="$( cat "${service\_dir}/${service}.pid" 2> /dev/null )"  wait\_"${service}" "${service\_pid}"  is\_running "${service}"  } |

1. 通过pid判断是否已经运行
2. 创建服务的工作目录
3. 配置服务：构建服务的数据目录、 将指定的属性文件拷贝到服务的工作目录等

配置服务的接口，提供了指定一个key-value的设置 —— 有利于设置服务唯一性标识的配置参数，进而在单机上构建不冲突的多个同类服务的进程。

1. 用指定的启动脚本和指定的配置属性，启动服务，并重定向输出
2. 保存 pid 到指定文件，并等待服务运行….

**待 —— Broke – broke id 以及 该broke 的数据存放的路径 —— 至少两个属性冲突 – 考虑采用不同配置文件的方式实现伪分布式集群部署。**

注意： 当前PID存放的文件路径 --- 同其他集群，tmp默认路径，可能定期清除，清除后无法通过命令查看进程状态，或stop该进程。

## 命令

命令根据作用的对象，可以分为两种：

1. 服务级别的命令 ： 参考主入口
2. Connector级别的命令：参考 connect\_subcommands 函数

命令支持的服务定义在services 和 rev\_ services 变量（参考**服务**章节），

主入口支持的命令：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 主入口 |  |  |  | 备注 |
| help |  |  |  |  |
| config | connect\_subcommands |  |  |  |
| connect | connect\_subcommands |  |  |  |
| current |  |  |  |  |
| destroy |  |  |  |  |
| list |  |  |  |  |
| load | connect\_subcommands |  |  |  |
| log |  |  |  |  |
| start |  |  |  |  |
| status |  |  |  |  |
| stop |  |  |  |  |
| top |  |  |  |  |
| unload | connect\_subcommands |  |  |  |
| acl |  |  |  |  |

主入口中，对应直接调用connect\_subcommands函数的命令，表示只支持connect级别的命令。其他命令级别都调用了*cmdName*\_command() 格式的函数，这些函数中也有支持connect级别的。

connect\_subcommands支持的子命令：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 子命令 | 具体函数 |  |  | 备注 |
| list | connect\_*cmdName*\_command |  |  |  |
| load |  |  |  |
| unload |  |  |  |
| status |  |  |  |
| config |  |  |  |
| restart | 未实现 |  |  |  |

对应 commands 和 enterprise\_commands 变量，包含的命令：

|  |
| --- |
| # 命令列表  declare -a commands=(  "list"  "start"  "stop"  "status"  "current"  "destroy"  "top"  "log"  "load"  "unload"  "config"  )  declare -a enterprise\_commands=(  "acl"  ) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 命令名 | 服务级别 | connector级别 | Enterprise版命令 | 备注 |
| "list" | Y | Y |  |  |
| "start" | Y |  |  |  |
| "stop" | Y |  |  |  |
| "status" | Y | Y |  |  |
| "current" |  |  |  |  |
| "destroy" | Y |  |  |  |
| "top" | Y |  |  |  |
| "log" | Y |  |  |  |
| "load" |  | Y |  |  |
| "unload" |  | Y |  |  |
| "config" |  | Y |  |  |
| "acl" | Y |  | Y |  |

### list命令

|  |
| --- |
| list\_usage() {  cat <<EOF  Usage: ${command\_name} list [ plugins | connectors ]  Description:  List all the available services or plugins.  Without arguments it prints the list of all the available services.  Given 'plugins' as subcommand, prints all the connector-plugins which are  discoverable in the current Confluent Platform deployment.  Given 'connectors' as subcommand, prints a list of connector names that map to predefined  connectors. Their configuration files can be found under 'etc/' directory in Confluent Platform.  Examples:  confluent list  Prints the available services.  confluent list plugins  Prints all the connector plugins (connector classes) discoverable by Connect runtime.  confluent list connectors  Prints a list of predefined connectors.  EOF  exit 0  } |

list 命令不带参数时，列出当前支持的所有服务，带参数时，对应为 connect 相关的子命令，包含列出connect运行是所有发现的plugins，列出所有预定义的 connectors。

confluent list plugins对应的 curl 命令：

### start/stop/destroy命令

start命令：

|  |
| --- |
| start\_usage() {  cat <<EOF  Usage: ${command\_name} start [<service>]  Description:  Start all services. If a specific <service> is given as an argument, it starts this service  along with all of its dependencies.  Output:  Prints status messages after starting each service to indicate successful startup or error.  Examples:  confluent start  Starts all available services.  confluent start kafka  Starts kafka and zookeeper as its dependency.  EOF  exit 0  } |

stop命令：

|  |
| --- |
| stop\_usage() {  cat <<EOF  Usage: ${command\_name} stop [<service>]  Description:  Stop all services. If a specific <service> is given as an argument it stops this service  along with all of its dependencies.  Output:  Prints status messages after stopping each service to indicate successful shutdown or error.  Examples:  confluent stop  Stops all available services.  confluent stop kafka  Stops kafka and zookeeper as its dependency.  EOF  exit 0  } |

服务的启动与停止，对应的函数start\_command 和 stop\_command 都调用了 start\_or\_stop\_service函数：

start\_or\_stop\_service "start" "services" "${@}"

start\_or\_stop\_service "stop" "rev\_services" "${@}"

start\_or\_stop\_service函数：

1. 参数 ： start/stop 启停， 服务列表变量 ， service
2. 对应服务的启动与停止，根据服务在指定变量中的定义顺序，逐个执行，直到执行完指定的服务后，退出 —— 对应定义的顺序，表示了服务启动/停止操作的依赖关系。

### destroy命令

|  |
| --- |
| destroy\_usage() {  cat <<EOF  Usage: ${command\_name} destroy  Description:  Delete an existing confluent run. Any running services are stopped. The data and the log  files of all services are deleted.  Examples:  confluent destroy  Confirms that every service is stopped and finally prints the filesystem path that is deleted.  EOF  exit 0  } |

使用 stop\_command 停止所有服务，并删除运行时根目录下的所有相关文件，包含工作目录和存放当前工作目录的文件。

### status命令

|  |
| --- |
| status\_usage() {  cat <<EOF  Usage: ${command\_name} status [ <service> | connectors | <connector-name> ]  Description:  Return the status of services or connectors.  Without arguments it prints the status of all the available services.  If a specific <service> is given as an argument the status of the requested service is returned  along with the status of its dependencies.  Given 'connectors' as subcommand, prints a list with the connectors currently loaded in Connect.  If a specific <connector-name> is given, then the status of the requested connector is returned.  Examples:  confluent status  Prints the status of the available services.  confluent status kafka  Prints the status of the 'kafka' service.  confluent status connectors  Prints a list with the loaded connectors at any given moment.  confluent status file-source  Prints the status of the connector with the given name.  EOF  exit 0  } |

包含服务的状态和connect的状态：

1. 服务状态：支持全部的服务状态和指定的服务的状态；
2. connect状态：支持获取所有已加载的connectors的状态和指定名字的connector的状态。

### current命令

|  |
| --- |
| current\_usage() {  cat <<EOF  Usage: ${command\_name} current  Description:  Return the filesystem path of the data and logs of the services managed by the current  confluent run. If such a path does not exist, it will be created.  Output:  The filesystem directory path to the current confluent run.  Examples:  confluent current  /tmp/confluent.SpBP4fQi  EOF  exit 0  } |

打印当前confluent运行时的工作目录。

### log命令

|  |
| --- |
| log\_usage() {  cat <<EOF  Usage: ${command\_name} log <service> [optional arguments to tail]  Description:  Read or tail the log of a service. If no arguments are given, a snapshot of the log is opened with  a viewer ('less' command). If any arguments are given, 'tail' is used instead and the arguments  are passed to the tail command.  Examples:  confluent log connect  Opens the connect log using 'less'.  confluent log kafka -f  Tails the kafka log and waits to print additional output until the log command is interrupted.  EOF  exit 0  } |

### top命令

|  |
| --- |
| top\_usage() {  cat <<EOF  Usage: ${command\_name} top [<service>]  Description:  Track resource usage of a service.  EOF  exit 0  } |

### load命令

|  |
| --- |
| load\_usage() {  cat <<EOF  Usage: ${command\_name} load [<connector-name> [-d <connector-config-file>]]  Description:  Load a bundled connector with a predefined name or custom connector with a given configuration.  EOF  exit 0  } |

Use '${command\_name} load ${connector} -d <connector-config-file.[json|properties]' to load the connector's configuration."

加载connector, 包含 ：

1. 指定名字的预定义 connector
2. 自定义的 connector ，必须带 -d 参数，指定该connector 对应的配置文件

当前配置文件支持两种格式： json和properties， 对应两个转换函数。

备注： 自定义connector时， connector 的名字从配置文件名中提取，而不是传入的名字。

### unload命令

|  |
| --- |
| unload\_usage() {  cat <<EOF  Usage: ${command\_name} unload [<connector-name>]  Description:  Unload a connector with the given <connector-name>.  EOF  exit 0  } |

### config命令

|  |
| --- |
| config\_usage() {  cat <<EOF  Usage: ${command\_name} config <connector-name> [ -d <connector-config-file> ]  Description:  Get or set a connector's configuration properties.  Given only the connector's name, prints the connector's configuration if such a connector is  currently loaded in Connect.  Additionally, given a filename with the option '-d', it configures the connector '<connector-name>'.  The file needs to be in a valid JSON or java properties format and has to contain a correct  configuration for a connector with the same name as the one given in the command-line.  Examples:  confluent config s3-sink  Prints the current configuration of the predefined connector with name 's3-sink'  confluent config wikipedia-file-source  Prints the current configuration of a custom connector with name 'wikipedia-file-source'  confluent config wikipedia-file-source -d ./wikipedia-file-source.json  Configures a connector named 'wikipedia-file-source' by passing its configuration properties in  JSON format.  confluent config wikipedia-file-source -d ./wikipedia-file-source.properties  Configures a connector named 'wikipedia-file-source' by passing its configuration properties as  java properties.  EOF  exit 0  } |

查看、修改 connector 的配置信息

### acl命令

|  |
| --- |
| acl\_usage() {  if [[ -z "${2}" ]]; then  cat <<EOF  Usage: ${command\_name} acl <service> [<parameters>]  Description:  Specify ACL to a service. Use help acl <service> to get further details about  [<parameters>]. Currently schema-registry is the only supported service for acl.  EOF  exit 0  else  acl\_command "${2}" "--help"  fi  } |

### Kafka Connect相关的命令

支持的命令参考 connect\_subcommands 函数。

注意：当前Confluent默认启动的是分布式的Kafka Connect，因此对应的相关的命令都是采用 rest ful api来访问， 即通过curl 发送请求。 在confluent脚本中，curl 指定了 -s选项，并且设置的max-time 为10s，因此，当需要返回结果时，可以考虑去除-s选项，并且在Kafka Connect响应慢时，适当增加 max-time默认值。

#### restart命令

## Connector

### 预定义connector

对应 connector \_properties 变量。

该变量定义了"load"/"unload" 命令可以加载/卸载的 connector的名字及其对应的配置属性名

|  |
| --- |
| "elasticsearch-sink=kafka-connect-elasticsearch/quickstart-elasticsearch.properties"  "file-source=kafka/connect-file-source.properties"  "file-sink=kafka/connect-file-sink.properties"  "jdbc-source=kafka-connect-jdbc/source-quickstart-sqlite.properties"  "jdbc-sink=kafka-connect-jdbc/sink-quickstart-sqlite.properties"  "hdfs-sink=kafka-connect-hdfs/quickstart-hdfs.properties"  "s3-sink=kafka-connect-s3/quickstart-s3.properties" |

“=”左边对应 "load"/"unload" 所需的参数，表示启动的connector 的名字，同时“ = ”右边给出了该connector 对应的配置属性文件相对与。

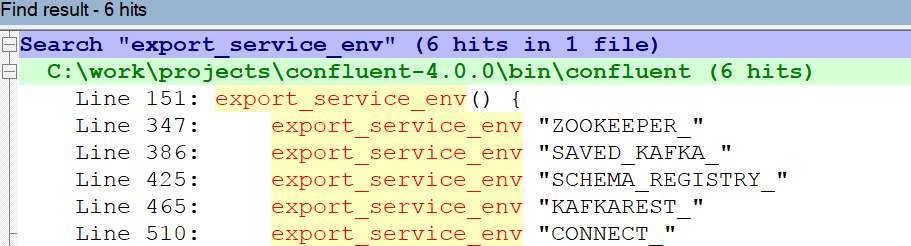
### 用到的curl命令

## 环境变量

### export\_service\_env 函数

传入的参数对应以启动的服务名作为环境变量的前缀标志，并转换为各个服务运行所需的通用的环境变量。

|  |
| --- |
| # 本意应该是，将所有特定的服务所指定的环境变量，转换为通用的 KAFKA\_ 作为前缀的环境变量。  # 两个可能存在的问题：  # 1. Kafka rest 服务 ： 通用的环境变量前缀为 KAFKAREST\_ ，而非 KAFKA\_ ---- 对应底层run-class 的脚本也不同，因此 假设的前缀并不是通用的  # 尤其是，通用的 KAFKA\_前缀的环境变量不能控制 Kafka rest 底层run-class的运行  # 2. 和前缀对应的 Kafka 服务，实际使用时传入的是 SAVED\_KAFKA\_  export\_service\_env() {  # The prefix needs to include any delimiters (e.g. '\_').  local prefix="${1}"  local var="${prefix}LOG4J\_OPTS"  export KAFKA\_LOG4J\_OPTS="${!var}"  var="${prefix}EXTRA\_ARGS"  export EXTRA\_ARGS="${!var}"  var="${prefix}HEAP\_OPTS"  export KAFKA\_HEAP\_OPTS="${!var}"  var="${prefix}JVM\_PERFORMANCE\_OPTS"  export KAFKA\_JVM\_PERFORMANCE\_OPTS="${!var}"  var="${prefix}GC\_LOG\_OPTS"  export KAFKA\_GC\_LOG\_OPTS="${!var}"  var="${prefix}JMX\_OPTS"  export KAFKA\_JMX\_OPTS="${!var}"  var="${prefix}DEBUG"  export KAFKA\_DEBUG="${!var}"  var="${prefix}OPTS"  export KAFKA\_OPTS="${!var}"  var="${prefix}CLASSPATH"  export CLASSPATH="${!var}"  } |



与服务列表对应，其中， 由于KAFKA 服务是转换后的通用环境变量的前缀，因此不在上图中。另外，SAVED\_KAFKA\_前缀是用于保存旧的KAFKA的环境变量，并不是服务名。

因此，可以通过带有不同服务名前缀的环境变量为特定的服务设置环境变量。

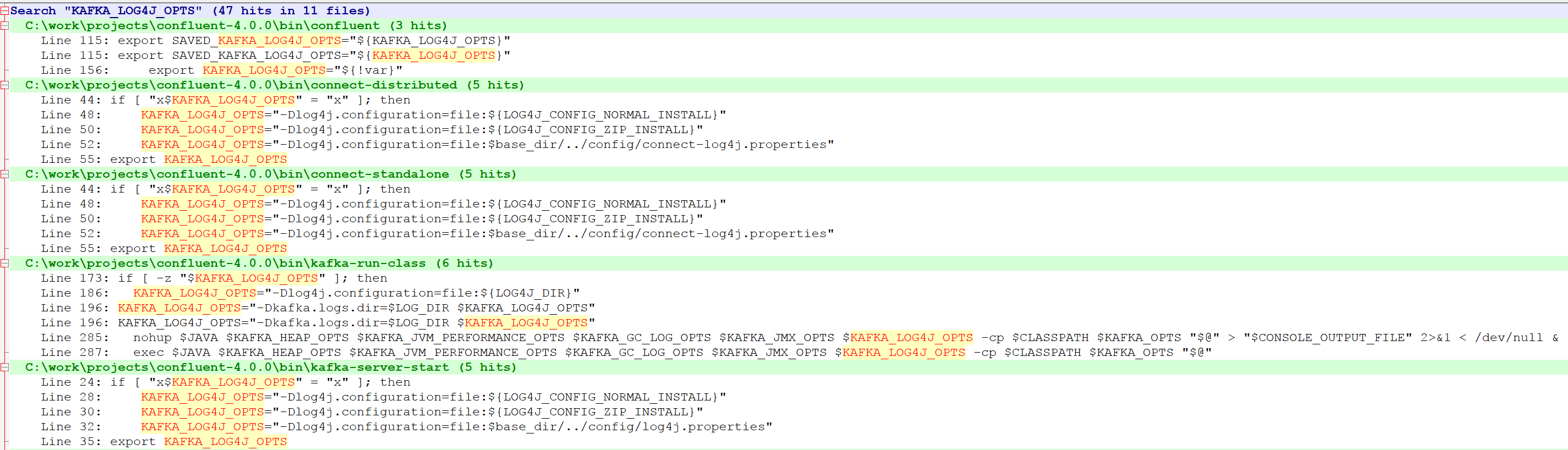
### 环境变量列表

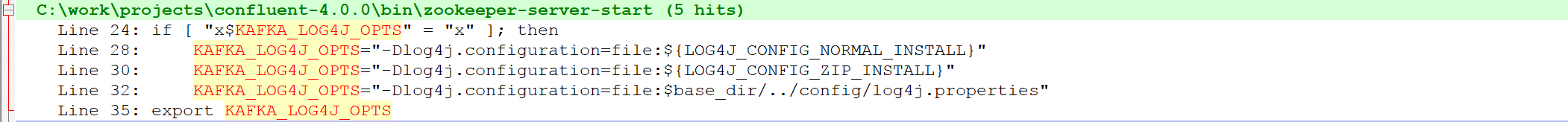
|  |  |  |  |
| --- | --- | --- | --- |
| 通用（或KAFKA）的环境变量 | 说明 | 服务的环境变量 | 备注 |
| KAFKA\_ LOG4J\_OPTS |  | ***SERVERNAME\_***LOG4J\_OPTS |  |
| KAFKA\_HEAP\_OPTS |  | **SAVED\_KAFKA\_**HEAP\_OPTS |  |
| KAFKA\_JVM\_PERFORMANCE\_OPTS |  | ***SERVERNAME\_***JVM\_PERFORMANCE\_OPTS |  |
| KAFKA\_GC\_LOG\_OPTS |  | ***SERVERNAME\_***GC\_LOG\_OPTS |  |
| KAFKA\_JMX\_OPTS |  | ***SERVERNAME\_***JMX\_OPTS |  |
| KAFKA\_ DEBUG |  | ***SERVERNAME\_***DEBUG |  |
| KAFKA\_OPTS |  | ***SERVERNAME\_***OPTS |  |
| CLASSPATH |  |  |  |
| EXTRA\_ARGS | 额外参数 |  |  |

### \*\_LOG4J\_OPTS环境变量

可以指定文件路径，更具体地控制日志打印。

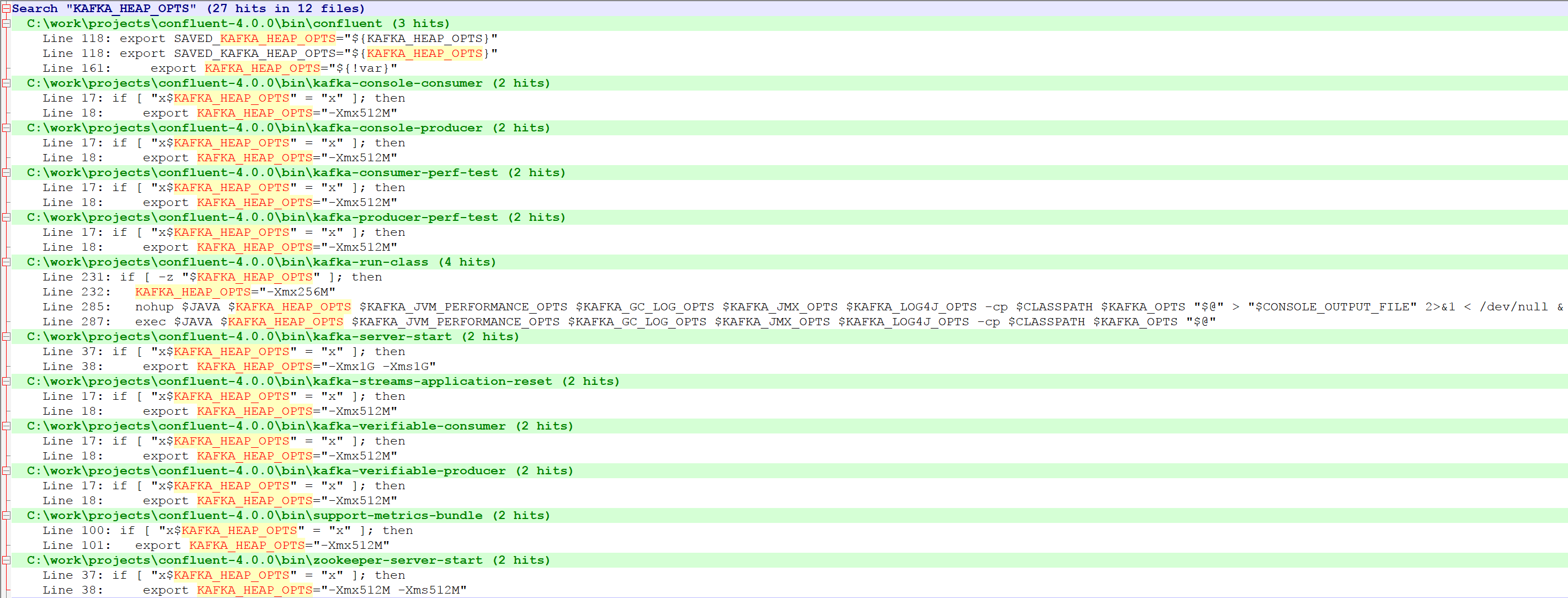
下面截图中可以看到当前脚本默认使用的不同的日志属性的文件路径。





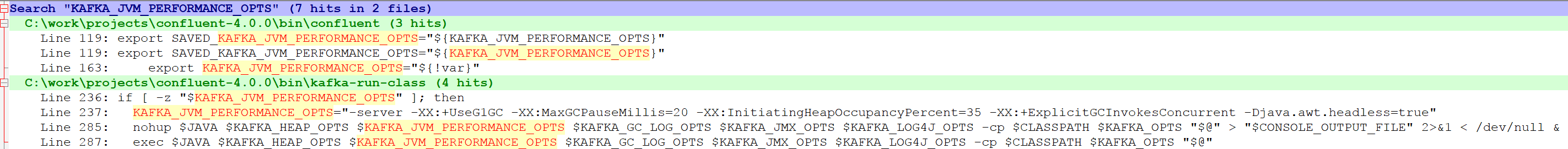
### \*\_HEAP\_OPTS环境变量

设置JVM进程启动的堆内存大小。



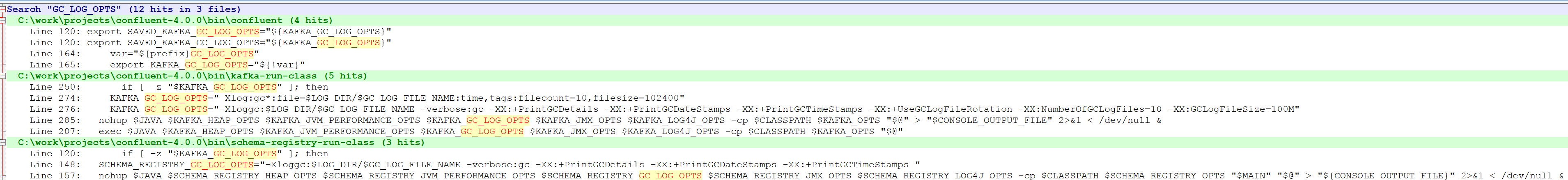
### \*\_JVM\_PERFORMANCE\_OPTS环境变量

JVM进程性能相关参数的设置。



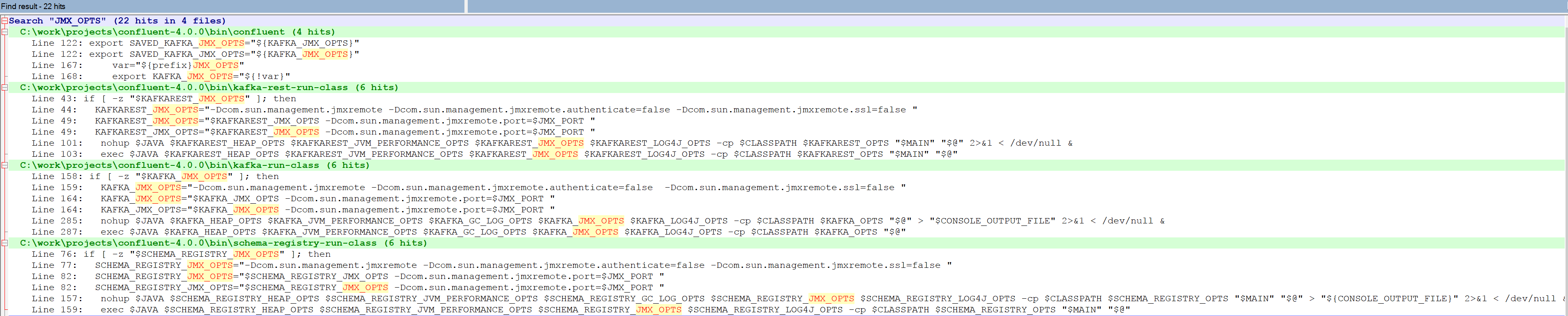
### \*\_GC\_LOG\_OPTS环境变量

设置JVM 的GC相关信息的日志属性。



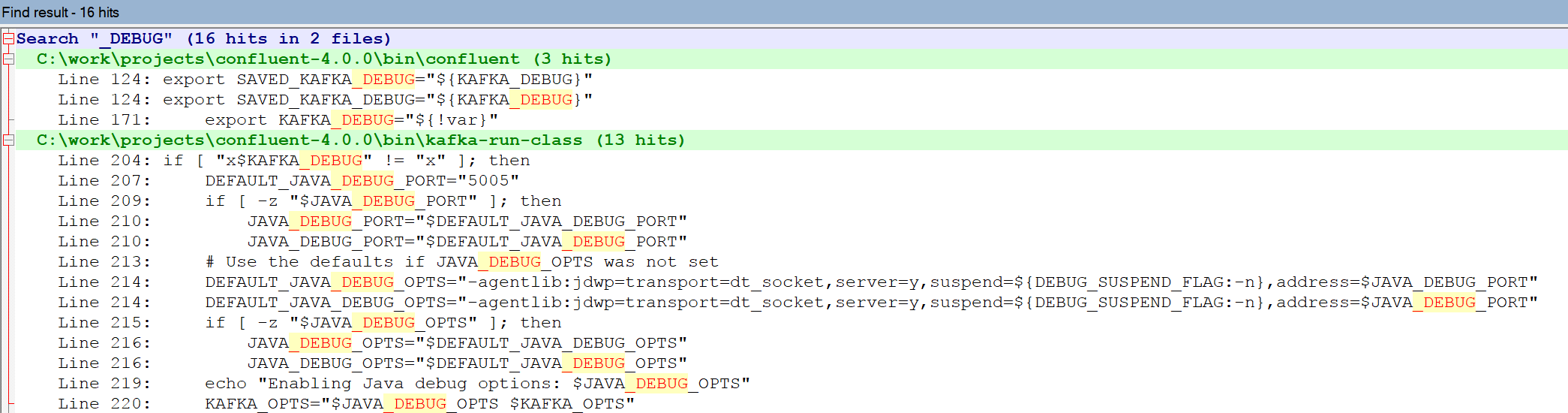
### \*\_JMX\_OPTS环境变量

设置JMX监控相关的JVM选项，与监听端口JMX\_PORT一起设置。可以通过jconsole等监控工具查看。

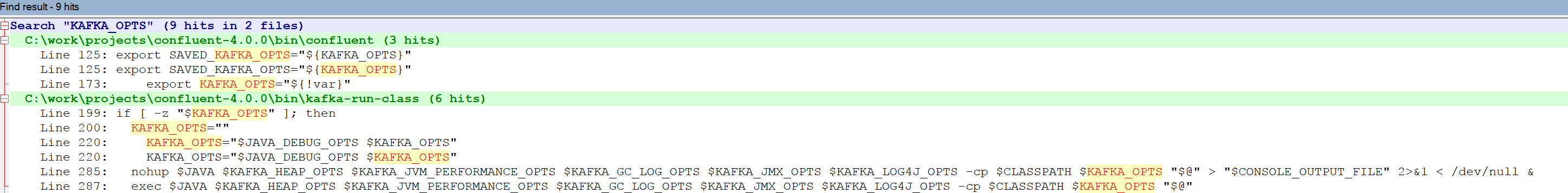


### \*\_DEBUG环境变量

设置JVM进程调试的选项，与调试端口 JAVA\_DEBUG\_PORT 一起设置。



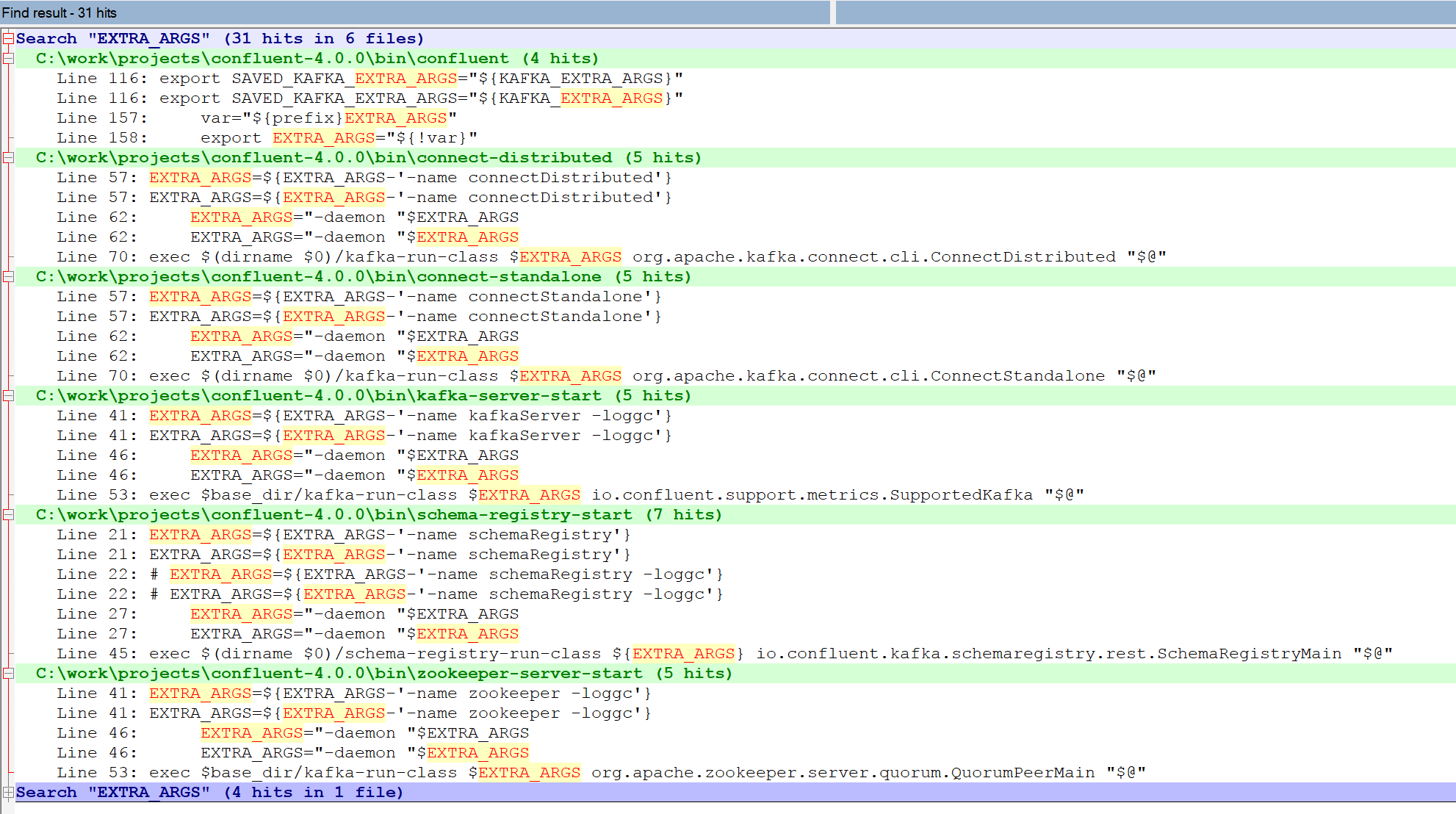
### \*\_OPTS环境变量



### EXTRA\_ARGS环境变量

最终在 kafka-run-class 中使用，用于运行特定 java 类时指定的额外参数，比如是否作为后台服务进程的方式运行等。

在各个脚本中的应用：



# 配置

目录： ./etc

# Confluent框架