**Core Engine自动化测试点**

[1 说明 2](#_Toc396141383)

[2 设置KUP 2](#_Toc396141384)

[2.1 原理 2](#_Toc396141385)

[2.2 测试步骤 3](#_Toc396141386)

[2.3 结果检测 3](#_Toc396141387)

[2.4 失败是否继续 3](#_Toc396141388)

[3 添加设备 3](#_Toc396141389)

[3.1 原理 3](#_Toc396141390)

[3.2 测试步骤 3](#_Toc396141391)

[3.3 结果判断 4](#_Toc396141392)

[3.4 失败是否继续 4](#_Toc396141393)

[4 查看视频直播 4](#_Toc396141394)

[4.1 原理 4](#_Toc396141395)

[4.2 测试步骤 4](#_Toc396141396)

[4.3 结果判断 4](#_Toc396141397)

[4.4 失败是否继续 5](#_Toc396141398)

[5 查看设备是否在线 5](#_Toc396141399)

[5.1 原理 5](#_Toc396141400)

[5.2 测试步骤 5](#_Toc396141401)

[5.3 结果判断 5](#_Toc396141402)

[5.4 失败是否继续 5](#_Toc396141403)

[6 查看Session信息的处理 5](#_Toc396141404)

[6.1 原理 5](#_Toc396141405)

[6.2 测试步骤 6](#_Toc396141406)

[6.3 结果判断 6](#_Toc396141407)

[6.4 失败是否继续 6](#_Toc396141408)

[7 存储参数设置 6](#_Toc396141409)

[7.1 原理 6](#_Toc396141410)

[7.2 测试步骤 6](#_Toc396141411)

[7.3 结果判断 7](#_Toc396141412)

[7.4 失败是否继续 7](#_Toc396141413)

[8 视频存储 7](#_Toc396141414)

[8.1 原理 7](#_Toc396141415)

[8.2 测试步骤 7](#_Toc396141416)

[8.3 结果判断 8](#_Toc396141417)

[8.4 失败是否继续 8](#_Toc396141418)

[9 图片存储 9](#_Toc396141419)

[9.1 原理 9](#_Toc396141420)

[9.2 测试步骤 9](#_Toc396141421)

[9.3 结果判断 9](#_Toc396141422)

[9.4 失败是否继续 10](#_Toc396141423)

[10 事件视频存储 10](#_Toc396141424)

[10.1 原理 10](#_Toc396141425)

[10.2 测试步骤 10](#_Toc396141426)

[10.3 结果判断 10](#_Toc396141427)

[10.4 失败是否继续 11](#_Toc396141428)

[11 更新设备 11](#_Toc396141429)

[11.1 原理 11](#_Toc396141430)

[11.2 测试步骤 11](#_Toc396141431)

[11.3 结果判断 11](#_Toc396141432)

[11.4 失败是否继续 11](#_Toc396141433)

[12 删除设备 12](#_Toc396141434)

[12.1 原理 12](#_Toc396141435)

[12.2 测试步骤 12](#_Toc396141436)

[12.3 结果检测 12](#_Toc396141437)

[12.4 失败是否继续 12](#_Toc396141438)

# 说明

1. 该测试当前只针对Node进行
2. 启动之后先调用setCloudServer设置KUP的地址
3. 下文中所有时间的格式均为%d%m%Y%H%M%S ，如31072014080200
4. 下文中提到的配置文件请见configuration.cfg

# 设置KUP

## 原理

调用ConfigControlServer中的setCloudServer方法设置KUP的地址

## 测试步骤

1. 调用ConfigControlServer中的setCloudServer方法设置KUP的地址(KUP的地址写在配置文件)
2. 查看数据库中是否设置成功

## 结果检测

1. setCloudServer返回true
2. 数据库的configurations表中存在name为“kup-arbiter-host”，value为KUP地址的记录
3. 只有上述两个都成功时才认为成功

## 失败是否继续

否

原因：对于Node而言，如果KUP的地址有问题，后续的测试可能会出问题，所以测试结束

# 添加设备

## 原理

通过thrift客户端发送添加设备的请求到Arbiter服务，调用完之后查看数据库中设备是否添加正确，并判断设备是否正确添加到Device Server中

## 测试步骤

1. 调用CoreServices.thrift的DeviceManagementService服务中的addDevice()方法向Arbiter服务发送添加设备

注意：

1. Arbiter服务器的地址见配置文件中”arbiter-server-host”；DeviceManagementService服务端的端口见” device-management-server-port”
2. addDevice()方法的输入参数见配置文件中” [addDevice]”模块
3. 在判断该设备是否添加到DS中之前让程序暂停30-40s，等待DS进行注册
4. addDevice()方法返回添加设备的id,将此Id写入到配置文件中的[updateDevice]以及[deleteDevice]下的device-id字段，以便更新和删除时调用。

## 结果判断

1. 判断devices表中是否有该device记录
2. 判断ds\_device\_info中是否有该device记录
3. 判断ds\_device\_info中的server\_id 是否是-1 (-1表示尚未添加到DS)，如果是-1，认为测试失败
4. 确认DS中该device是否存在：调用DeviceCommsAPI.thrift中DeviceServerService服务的getDevices方法（DeviceServerService 的host,port 需查数据库中ds\_server\_info 表的host和port字段），判断设备是否存在

## 失败是否继续

否

原因：该过程如果失败，说明设备未正确添加到数据库或者Device Server，接下来的测试将没有意义

# 查看视频直播

## 原理

1. 从Arbiter服务获取某个设备直播视频的URL，判断URL是否正确
2. 从Device Server获取某个设备的帧率，比较该帧率和设备设定帧率之间的误差是否在一定范围内

## 测试步骤

1. 调用CoreServices.thrift 中的StreamControlService 服务的beginStreamSession 方法获得设备对应的URL地址 (StreamControlService 的host 见配置文件的"arbiter-server-host"， port见"stream-control-server-port"；beginStreamSession中startTimestamp 和 endTimeStamp在视频直播时为空（None），type 值为”rtmp/h264”，其返回值为List 列表)
2. 调用DeviceServerService 中的getDeviceInfo获取Device Server中该设备的帧率。

## 结果判断

1. 如果得到的URL非空，就认为是正常的
2. getDeviceInfo的返回值为JSON,其中frame-rate字段表示帧率, 将获得的帧率与额定帧率做比较，误差范围在x%内正常，将测试所得帧率（frame-rate）、额定帧率（rated-frames）、误差范围（percent如0.1）写入到配置文件configuration.cfg中
3. 误差范围的取值（网络状况良好为10%(min-percent)，若网络状况差，可调至30%(max-percent)）

## 失败是否继续

否

原因：如果直播视频的URL为空，说明DS没有正确生成可直播的视频流，应该失败；如果帧率的误差过大，说明肯定是有问题了，因为在局域网网络状况不会如此差

# 查看设备是否在线

## 原理

通过调用Arbiter的DeviceControlService服务的getDeviceStatus()方法，获取设备当前的状态。如果返回online，说明设备在线，测试通过

## 测试步骤

1. 连接DeviceControlService，host------"arbiter-server-host" port------"device-control-port"
2. 调用服务的getDeviceStatus(),参数为deviceId.
3. 判断getDeviceStatus()的返回值

## 结果判断

1. getDeviceStatus()的返回值为”online”,则为True，否则为False，设备未在线。

## 失败是否继续

是

# 查看Session信息的处理

## 原理

举例：

--在1:00:00调用beginStreamSession方法，查看设备的liveview(不用判断结果)，将ttl设置为2分钟(代表两分钟后该Session过期)

-- Arbiter收到消息后会将Session信息存放在stream\_session\_info表中

--在1:00:01分，测试程序再调用keepStreamSessionAlive，ttl为2，将Session延长到4分钟

--在1:00:02分，先查看stream\_session\_info中是否有对应SessionId的记录，如果有就说明keepSession成功。之后，程序调用endStreamSession，删除该Session，再次查看stream\_session\_info，查看Session是否删除成功

## 测试步骤

1. 在某一时刻调用StreamControlService的beginStreamSession方法,参数设定格式与直播测试时类同，将ttl 设置为2分钟，即配置文件中ttl=120 单位”s”;
2. 休眠1min,查看stream\_session\_info表中是否存在id=sessionid的记录,存在则继续执行
3. 调用keepStreamSessionAlive()方法，参数只需设置sessionId, ttl，sessionid需同1)中beginStreamSession的sessionid相同，ttl设置为120s，举例如下:

keepStreamSessionAlive(“sdsjgghjh-sjdh-dgsh-jhsk-325245sds”,120)

1. 休眠1分5秒，查看stream\_session\_info表中是否还存在id=sessionid(1中的sessionid)的记录。
2. 如果上述记录存在，则调用endStreamSession()方法,执行删除操作。
3. 再次查看stream\_session\_info表中是否还存在id=sessionid的记录。
4. 结果判断

## 结果判断

1. 上述测试步骤中第4)步所得结果不为空，则keepStreamSessionAlive执行成功，程序向下执行
2. 第6)步所得结果为空，则endStreamSession执行成功。
3. 上述两次判断均对符合要求，则本测试点执行成功。

## 失败是否继续

是

# 存储参数设置

## 原理

通过调用接口向Arbiter发送设置视频存储长短的消息，Arbiter会把这个消息发送给Recording Server

## 测试步骤

1. 调用CoreService.thrift 中的 ConfigControlService服务的setChunkSize方法，设置”chunk-size”( 将chunk-size写入到配置文件，chunk-size意为每段视频存储的时间长短，单位’分’)
2. ConfigControlService的host，port对应"arbiter-server-host"和"config-control-server-port"

## 结果判断

如果调用接口时返回true，就认为测试成功

## 失败是否继续

是

原因：如果设置视频长短失败，后续的视频存储就会有问题，所以失败

# 设置每路视频的存储空间

## 原理

--设置每一路视频的存储空间的大小，判断接口调用是否成功；如果成功，就开始存储视频，并判断是否按照设置的存储空间的大小进行存储了

--分别测试 3组值，即0, 10M和30M；每组值需要测试以下步骤：

1. 开始存储。
2. 调用ConfigControlService中的setStreamStorageLimit()，，如果接口返回true，就认为设置成功
3. 测试设置的值是否生效：过一段时间后判断是否按照预期的进行存储了。

--即，应该得到一下结果：

--在设置为0时，应该是存储不了任何视频的，即所得视频应该为None

--在10M时，如果存储的视频空间到了10M，会将存储的最早的视频删除掉，进行最新视频的存储

--30M和上面的10M类似

## 步骤

前提 : 已知存储一个两分钟的视频，所用空间大概为6M

设置Arbiter服务中删除线程默认执行时间间隔为1分钟

第一步 设置存储空间为0M

1. 开启视频存储,即更新devices表中的cloud\_recording\_enabled的值为1，记录开始时间begin\_local\_time
2. 调用ConfigControlService中的setStreamStorageLimit()方法，设置存储空间为0M
3. 2)中返回True,等待2分钟，然后停止存储，记录此时的时间end\_local\_time
4. 再等待3分钟，用RecordingServerService的getStreamList()方法，可获得从begin\_local\_time到end\_local\_time这一时间段内存储的视频，如果所得为None，则判定为设置存储空间为0M成功。

注: 停止存储后，等待，是因为删除线程进行判断，删除，需要时间

第二步 设置存储空间为10M

1. 开启存储，记录开始时间begin\_local\_time
2. 调用setStreamStorageLimit()方法，设置存储空间为10M，结果为True，则继续
3. 让视频存储2分钟后，记录此时的时间为end\_local\_time
4. 调用getStreamList()，可获得begin\_local\_time到end\_local\_time所存储的第一个视频的开始时间first\_video\_begin
5. 再等待3分钟后，记录此时的时间为end\_local\_time
6. 再次执行4)的步骤，获得的时间为second\_video\_begin
7. 比较first\_video\_begin与second\_video\_begin，即可得出所设置的空间为10M是否起作用

第三步 设置存储空间为30M

1. 在第二步的基础上，执行setStreamStorageLimit()，参数设为30M
2. 首先，调用getStreamList()获得begin\_local\_time(第二步中的)到当前时间，存储空间中的第一条记录的开始时间first\_video\_begin
3. 让视频存储2分30秒的时间，调用getStreamList()获得begin\_local\_time(第二步中的)到当前时间存储空间中第一条记录的开始时间为second\_video\_begin
4. 比较first\_video\_begin与second\_video\_begin的值，即可判断出设置空间大小为30M是否成功

## 结果判断

1. 第二步中，若二者值不等则设置存储空间为10M成功

说明: 第二步中视频总计存储5分多钟，预计可存储2个视频，大小预期为12M，若设置空间10M成功，则必定要删除开始时存储的视频，所以如果两次getStreamList所得结果不相同，则说明删除成功，设置存储空间成功。

1. 第三步中，若二者相等则设置存储空间为30M成功

说明: 本步骤中，接着让程序存储2分30秒，如果存储空间还是10M，则会删除第一个视频，那么first\_video\_begin与second\_video\_begin的值应不相等，如果二者相等，则没有删除第一个视频，即设置是成功的

## 失败是否继续

是

注: 未避免由于本步骤影响“视频存储”的存储过程，需设置为不影响视频存储的大小的值，

“视频存储“预计需要的存储空间为26M以上，所以最终将存储空间大小设置为30M

# 视频存储

## 原理

1. 在某个时间段内按照如下方式进行视频存储测试：开启视频存储🡪过段时间后停止存储🡪过段时间再次开启🡪再次停止
2. 之后通过接口从Recording Server查询是否在正确的时间段存储了视频

注意：

1. 开启和停止视频存储的方法：通过向Arbiter发送更新设备的消息，修改cloud\_recording\_enabled的值（1开启存储，0为停止）

## 测试步骤

1. 开启视频存储🡪过段时间后停止存储🡪过段时间再次开启🡪再次停止🡪查看存储的视频是否正确
2. 在某个时间段内执行上述存储策略，记录策略开启时间和结束时间（这策略中的时间节点必须是”chunk-size”的整数倍，最好为整分整秒）,计算出这个时间段内理论上存储视频的个数。将策略开启时间和结束时间记录在配置文件中，以便后期使用。策略开启时间和记录时间均需记录两种（Local-Time And UTC-Time，二者相差8小时）

Local-Time:

开启时间 ------- liveview-begin-time-local

结束时间 ------- liveview-end-time-local

UTC-Time:

开启时间 ------ liveview-begin-time

结束时间 ------ liveview-end-time

注：“chunk-size”是2分钟，则策略时间节点需为：0,2, 4,6,8,10…

## 结果判断

1. url检测

调用StreamControlService 的beginStreamSession 获得url，URL 非空即为正确（与测试视频直播不同的是，参数中startTimestamp 和 endTimeStamp 此时应为 UTC-Time的 liveview-begin-time 与 liveview-end-time，beginStreamSession的参数请查阅java对应的thrift文件获得，其type为rtmp/h264）

1. 帧率检测

调用RecordingCommsAPI.thrift 中 RecordingServerService服务的getStreamList方法，可获得每一段视频的信息，如：帧率，视频长度。

getStreamList 输入举例：

getStreamList( (int)(deviceid,0,streamInfo);

streamInfo 格式为JSON ,其内参数举例:

{"storage-type":"video-recording","stream-type":"http/h264","begin":beginTime,"end":endTime},其中的begibTime和endTime 分别对应Local-Time 的 “liveview-begin-time-local” 和 “liveview-end-time-local”

getStreamList输出举例：

getStreanList 的输出格式为JSON : {"from":"07112013000000","to":"07112013001500", "dur":"900", "fn":"26370", "fps":"29.3", "size":"50000000"}。其中”dur”表示视频长度，单位秒,”fn”表示总帧数，”fps”表示帧率，”size”表示视频大小，单位字节

1. 通过以下方式判断测试结果

--“dur”与”chunk-size”的误差在x%范围之内。

--“fps”与直播中的测试所得的帧率误差在x%范围之内。

--“fn”和”size” 非空

注：误差范围写入配置文件，字段名用 “rates”,值初定为”0.1”

## 失败是否继续

继续

原因：视频存储失败不会影响后续的测试

# 图片存储

## 原理

1. 执行存储策略: 某个时间段内执行开启存储---停止---再次开启---停止这一过程。
2. 策略执行过程中，每存一张图片都会返回一个url地址，因此，在策略执行结束后，可通过调用StreamControlService的方法判断获得的url是否符合要求，可通过调用RecordingServerService的方法判断存储的图片是否符合标准。如果二者都OK，则图片存储功能测试成功。

注意: 图片存储控制:设置devices 表中snapshot\_recording\_enabled为’1’ 或 ’0’表示开启/关闭;表中的 snapshot\_recording\_interval字段表示存储时间，单位 ’s’，默认为5S .

## 测试步骤

1. 图片存储策略：

假设图片每隔五秒存储一次。存储策略为：开启---停止---开启---停止。

1. 在某段时间内执行上述存储策略，与视频存储相同，记录策略开始时间和结束时间，写在配置文件中。开始/结束时间同样需要两种，且为5秒的整数倍：

Local-Time: photo-begin-time-local & photo-end-time-local

UTC-Time: photo-begin-time-utc & photo-end-time-utc

再根据开始时间和结束时间，以及snapshot\_recording\_interval的值，计算出理论上能够存储多少张图片。

## 结果判断

1. 测试URL

调用StreamControlService 的beginStreamSession 获得urlList。与视频存储不同的是，照片存储中beginStreamSession的输入参数中type值为http/jpeg。而且在判断URL的时候，视频存储只要进行非空判断即可，而图片存储时，每张图片都会有一个URL地址，所以，需要判断所获得的urlList 的大小，是否等于理论上得到的值。相等则正确。

参数中的两个时间分别对应UTC-Time的两个之间；

1. 检测照片信息

调用RecordingServerService服务的getStreamList可获得照片信息。getStreamList的参数设置与视频测试相同。与视频测试不同的是，照片存储的type值为http/jpeg。

时间设定分别 对应Local-Time 的两个时间。

getStreamList 输入举例:

{"storage-type":"image-recording","stream-type":"http/jpeg","begin":"07112013000000","end":"07112013101000"}

getStreamList输出举例:

{“time”:” 07112013101000”,”width”:”640”,”height”:”480”,”size”:”50000”}

width: 宽, height: 高, size:大小 。

判断条件: 宽，高，大小，均不为0，则为正确。

## 失败是否继续

继续

原因：图片存储失败不会影响后续的测试

# 事件视频存储

## 原理

1. 调用DeviceDataReceiverService的sendEventData向Arbiter发送事件消息，Arbiter收到消息后，会向RS服务发送存储视频的消息，从而开始事件视频存储。
2. 结束后调用RecordingServerService的getStreamList可获得存储的视频信息，判断是否符合条件。

## 测试步骤

1. 调用DeviceCommsAPI.thrift中的DeviceDataReceiverService的sendEventData方法向Arbiter发送事件，当 Arbiter收到消息后会向RS发送存储事件视频的消息
2. sendEventData的输入举例：

long serverId, long deviceId, String eventType, long eventTime, String description, String stringData, ByteBuffer binaryData

1. 其中serverId,deviceId,eventTime,description和binaryData都可为null
2. 将eventType设置为”CAPTURE\_EVENT\_VIDEO”，stringData是个json字符串，里面设置eventId（随机字符串，比如95754ca2-4037-4747-ab38），deviceId（添加的设备的ID）和channelId（为0就行）,所以:

stringData = {"eventId":str(uuId),"deviceId":devId,"channelId":"0"}

sendEventData(None,None,'CAPTURE\_EVENT\_VIDEO',None,None, stringData,None).

1. DeviceDataReceiverService 的 Host --------- "arbiter-server-host"

Port ----------- "data-receiver-port"

1. 结果: sendEventData 的返回值为 True/False ,So, True 则正确

注: eventId 用 uuid.uuid1() 可获得随机字符串。eventId需写入配置文件，以便在结果判断中调用。

## 结果判断

1. 调用RecordingServerService服务的getStreamList，获取返回视频的帧率和时长。
2. 参数设定为:

{"storage-type":"event-recording","stream-type":"http/h264","event-id":"5279d99be4b0a381f22beda1"}

1. event-id 从配置文件中读取，每次调用需同7.2测试步骤中一致。
2. 方法返回（JSON）举例:

{"dur":"10", "fn":"263", "fps":"29.3", "size":"500000"}

1. 结果分析:所有值不为’0’ , "dur"和规定的秒数10s的误差范围x , x取自配置文件中的’rates’ .
2. 符合条件则，判断此次事件视频存储正确。重复执行(1)(2)，测试数据6次。若准确数据在5次以上，则判断为结果正确，事件视频存储功能可以正常运行。

## 失败是否继续

继续。

事件测试循环六次，若在循环过程，某一次循环失败，仍然继续，记录正确的事件数。

在六次循环结束后，安装判定规则，若结果失败，仍然继续测试其他的测试点，因为本次测试不影响其他测试点的进行。

# 更新设备

## 原理

调用CoreServices.thrift中DeviceManagementService服务的updateDevice方法更新设备。

## 测试步骤

1. 更新设备-->分别查看数据库信息和DS上的设备更新情况。
2. 调用CoreServices.thrift中DeviceManagementService服务的updateDevice方法更新设备。设备的参数都写在配置文件中，更新设备时，先修改配置文件中需修改的参数的值，后执行updateDevice方法。
3. 设备更新成功后，重复执行2-4-5-6 步骤，测试更新是否有误。

注: 更新设备的Id 由配置文件的[updateDevice] 的device-id获得。

## 结果判断

1. updateDevice 的返回值是Boolean类型，为True则成功执行。

## 失败是否继续

如果updateDevice返回False，则更新失败，就不再重复执行2-4-5-6步骤。但是对于其他的测试点而言已无影响，故测试可继续向下进行。

# 删除设备

## 原理

调用DeviceManagementService服务的deleteDevice方法,用于删除数据表中的添加的设备信息以及设备的注册信息等,在删除操作结束后，执行查询操作，查看是否成功清除数据表中的数据，若返回为空，则数据删除成功，否则，不成功。

## 测试步骤

1. 调用CoreServices.thrift中DeviceManagementService服务的deleteDevice()方法删除设备
2. 查询表中是否还存在要被删除的数据: 如:ds\_device\_info

注: 删除数据时的deviceId 由配置文件中的[deleteDevice]下的device-id 获得。

## 结果检测

1. deleteDevice()方法返回为True
2. 查询表中数据所得为None

符合上述条件，则删除操作可成功进行。

## 失败是否继续

删除操作测试是所有测试进行之后才开始进行的测试，故其测试结果不影响其他，成功与否都继续进行，本次系统测试也将由此结束。