

GOS 3.2

Programming Tutorial

版本：0.3.5
作者：中国科学院计算技术研究所
时间：2008-12-30

GOS 3.2 Programming Tutorial

版本：0.3.5

作者：中国科学院计算技术研究所

时间：2008-12-30

中国科学院计算技术研究所

目录

1. 前言	5
2. 使用说明	5
2.1 在 gosShell 中运行	5
2.1.1 拷贝 jar 包	6
2.1.2 启动和登录 gosShell	6
2.1.3 运行	6
2.2 使用 grun 运行	7
2.3 使用 mygrun 运行	8
2.4 操作线程的上下文运行示例程序	8
3. 资源端准备	9
3.1 资源	9
3.2 应用	9
3.3 开发服务	9
3.4 部署服务	10
3.4.1 静态部署	10
3.4.2 动态部署	11
3.5 添加资源	11
3.6 删除资源	12
3.7 权限设置	12
4. 编写 Java 应用	13
4.1 访问资源	13
4.1.1 open	13
4.1.2 Execute	13
4.1.3 Close	14
4.1.4 关于 open, execute 和 close 的一些补充	14
4.1.5 安全特性	14
5. 编写 Webapp 应用	15
5.1.1 访问资源	15
5.1.2 安全特性	15
6. 独立运行的 java 程序	15
6.1 实现自己的 grun	15
6.2 运行时切换操作上下文	16
6.3 设置 gos.home	16
6.4 修改配置文件中的 IP 和端口	17

6.5 所需其他文件.....	17
6.5.1 客户端 wsdd 文件	17
6.5.2 动态链接库.....	17
6.5.3 安全相关配置文件.....	17
6.6 运行.....	17
7. 上下文	18
7.1 设置操作上下文.....	18
7.2 设置和获取线程的操作上下文.....	18
7.3 获取资源端的操作上下文.....	18
7.4 获取网程上下文.....	19
7.5 开发中如何可靠的获取操作上下文?	19
8. 网程的运行	21
8.1 Javaapp 开发.....	21
8.1.1 开发环境配置.....	21
8.1.2 开发接口使用介绍.....	21
8.2 Webapp 开发.....	24
8.2.1 开发环境配置.....	24
8.2.2 开发接口介绍.....	24
8.2.3 开发实例.....	25
9. API 完成系统管理	30
10. 开发 gsh 命令	30
11. 开发资源控制器	31

1. 前言

本文档描述基于 GOS 的应用开发。

本文档主要描述 GOS 应用开发的一般特性，主要针对 java application。对于想要获得安全特性的 webapp 程序的开发，参见 8.2 Webapp 开发。本文档中涉及到 gosShell 的一些操作，更详细的信息请参考《GOS 3.2 GSH User Manual.doc》。如果需要新增资源类型，请关注 11. 开发资源控制器。

为了描述方便，约定下列符号：（可能会混合使用）

`${gos.InstallHome}`：代表 GOS3.2 的安装文件夹。

`${gos.home}`：

如果是以”split”方式安装则代表`${gos.InstallHome}/core`

如果是以”all”方式安装则代表`${gos.InstallHome}/all`

注：涉及安装相关内容可以参见安装手册。

`${gsh.home}`：`${gos.home}/gosShell/binary`

2. 使用说明

本文档自带一个工程 GOSTutorial，其中包含文档中描述的源代码。GOSTutorial 工程放在安装好的 GOS3.2 中的`${gos.home}/docs`目录下，我们现在提供该工程的两种存档文件：GOSTutorial.tar.gz 或者 GOSTutorial.zip。使用此工程之前，需要预先安装 GOS 3.2，或者找到一台安装有 GOS 3.2 的服务器。

GOS中程序如果以普通java进程方式运行，其中没有带用户身份和社区相关信息，因此从AxisPing的服务端也无法获得此信息。对于要做服务端访问控制的那些服务，则会由于权限的问题而运行失败。

解决用户身份和社区信息的问题具体可以有四种做法，这里介绍其中两种，另两种参见 6.1 实现自己的 grun 和 6.2 运行时切换操作上下文。

2.1 在 gosShell 中运行

在 gosShell 中运行的程序自动会以网程方式运行，其中的操作自动获得用户身份和社区信息。

具体做法如下：

2.1.1 拷贝 jar 包

进入 `${gos.home}/docs` 文件夹，解压 GOSTutorial 工程，将工程中已经预先打好的 `lib/gos-tutorial.jar` 包，拷贝到 `${gos.home}/clientLib` 文件夹。如果，用户对该工程代码做过修改，那么 build 之后，请将其 `bin` 目录下的 class 文件（包括包文件夹）打包成 jar 包，然后拷贝到 `${gos.home}/clientLib` 文件夹。

2.1.2 启动和登录 gosShell

进入 `${gos.home}/gosShell/binary` 目录，执行 `./grun.sh`(Linux 机器)或者 `grun.bat` (Windows 机器)。

gosShell 提示 `please input your name:`，此时输入网格用户名。本 tutorial 中为了简单，以网格管理员身份登录。例如：输入 Admin_200@cngriid.org (注意：用户名必须是邮件格式) 回车后，gosShell 提示：`please input password:*`，此时输入该用户密码。

回车后，如果输入正确，gosShell 提示：

```
log in successfully
```

```
welcome Admin_200@cngriid.org to the agora Agora_200
```

```
"grun warfilename" to deploy a web application as a grip
```

```
"grun commandwithargs" to execute an application as a grip
```

```
"goscommand" to execute a gos command
```

```
"commandwithargs" to execute a command or an application in the native system
```

```
"ghelp" to display all the available gos command
```

```
[gos]
```

如果用户名或者密码错误，系统则报异常。

[注：]GOS 安装后会产生两个用户：网格管理员和默认用户，其用户名是在安装前修改 `${gos.InstallHome}/installBase/gosconf.properties` 文件中指定。

。

2.1.3 运行

在提示符 `[gos]` 后键入：

```
grun java -Dgos.home=$GOS_HOME org.gos.core.tutorial.PingClient  
，即可启动此 tutorial。
```

如果运行成功，可以看到如下结果：

```
add ext service OK. resource is AxisPing  
add ext service OK. resource is vsPingServiceSec  
add ext service OK. resource is vsPingServiceNoSec  
add ext service OK. resource is PingServiceSec
```

```
add ext service OK. resource is PingServiceNoSec
...省略很多行...
ping result : context: #operate-context<org.gos.core.grip.utils.SimpleAgoraHandle@4218cb:org.gos.core.grip.utils.SimpleAgoraHandle@4218cb,user:org.gos.core.grip
.utils.UserAuthStruct@169c6f2,grip-id:http://10.61.0.45/:8080/F9E1F883451F5AD3BE
831A9C18C6C64F7779267C>, called by Admin45@vegasuite.ict.org, in agora Agora45,
origin msg: ping a service at Mon Oct 22 11:20:48 CST 2007
...省略很多行...
removed resource AxisPing
removed resource vsPingServiceSec
removed resource vsPingServiceNoSec
removed resource PingServiceSec
removed resource PingServiceNoSec
```

注意中间含有 ping result 字样的行，其中 context: 等是 AxisPing 服务在服务端获得的 context，其中包含当前用户身份和当前社区等信息。

基于这个 context，服务实现者可以做访问控制。

2.2 使用 grun 运行

进入\${gos.home}/gosShell/binary，执行

```
./grun.sh -u Admin_200@cngid.org -p pwd -c "java -Dgos.home=${gos.home}
org.gos.core.tutorial.PingClient"
```

注意将其中的-u -p 后的参数替换成被测试节点的节点管理员的用户名和密码，-c 中 -Dgos.home 也指向实际的 gos.home。例如：/opt/GOS3。

gosShell 中提供了一个小工具: grun.sh，用于支持以网程模式运行一个程序。Grun 支持以下用法：

- 无参数运行 grun.sh，提示输入用户名和密码，启动 gsh。
- grun.sh -u username，提示输入密码，启动 gsh
- grun.sh -u username -p password，通过用户名和密码登录之后启动 gsh
- grun.sh -pf proxyfile，通过 proxy 登录之后启动 gsh

这些登录方式均可另外带选项 -c command，以网程方式执行指定命令，而非 gsh。如果 command 本身带有选项，需要使用引号括起来，比如

```
./grun.sh -u Admin_200@cngid.org -p pwd -c "java -Dgos.home =/opt/GOS3/ GuserInfo"
```

通过-u -p -c 参数，可以用非交互式的方式,通过网程来运行一个程序。

2.3 使用 mygrun 运行

本 tutorial 为了展示 GOS 中网程的使用，给出了一个特别简单的 mygrun 命令的实现，用于将一个程序以网程方式运行。Mygrun 是 grun 命令的简化版本，其格式与 grun 命令类似: mygrun.bat username password command.

进入 tutorial 工程目录，

- 检查 conf 下的配置文件 namingserver.conf.t，将其中的\${gos_host}和\${gos_port}使之指向正确的网格节点的 core 容器的 ip 和端口，并将此文件改名为 namingserver.conf;
- 检查 mygrun.bat 中的 JAVA_HOME，并利用 Windows 的 cmd 控制台进入 mygrun.bat 文件目录中执行：

```
mygrun.bat      Admin_200@cngrid.orgpwd      "java      -Dgos.home      =.
org.gos.core.tutorial.PingClient"
```

注：如果在 Linux 下，建议将 mygrun.sh 拷贝到某个 GOS 安装节点的 gosShell/binary 目录，并修改 grun.sh 的 GOS_HOME 的定义，使之指向真正的 GOS 的安装目录。

如果执行成功，可以看到 2.1 节所述输出信息。

2.4 操作线程的上下文运行示例程序

为了调试方便，也可以在 IDE 中直接运行程序。当然，需要指定虚拟机参数

- -Dgos.home=.
- -Djava.library.path=.

或者也可以像 2.3 节所述，利用 mygrun 来运行：

```
Windows:  mygrun.bat  Admin\_200@cngrid.org  pwd  "java  -Dgos.home  =.
org.gos.core.tutorial.ThreadOperateContextDemo"
```

Linux 下类似利用 mygrun.sh 运行。

运行成功会有类似的输出：

```
start running...
get defaultUserName User_200@cngrid.org
login success!
OperateContext: user is User_200@cngrid.org, agora is Agora_200
logout OK!
```

3. 资源端准备

3.1 资源

GOS 中最重要的概念是资源。资源在 GOS 系统中是指可以被统一接入和管理的实体，并且这些实体将作为客体被其他网格应用所访问。在 GOS 系统中，服务、消息、软件、数据库，或更细粒度的实体都可以是资源。资源在 GOS 系统中是一个结构体分散的存储在不同的节点上，并且可以由创建者设置其访问控制权限。

GOS 中资源相关的所有接口都在 `org.gos.core.resource.client.ResourceClient` 中。

3.2 应用

GOS 中使用资源的程序称为应用，包括本地程序和 `webapp` 两种形态。

应用开发时只需要关注所访问的资源，应用如果以网程方式运行，则会在整个生命周期中拥有网程上下文 (`GripContext`)，其中包含当前用户身份和当前社区信息。当应用访问资源时，GOS 会根据 `GripContext` 构造一个适用于本次资源访问的操作上下文 `OperateContext`，并传递到资源端。资源端可以取出此 `context`，并基于此做访问控制。GOS 安全的服务需要在资源端配置相应的 `ACHandler`，此 `handler` 会检查此 `context` 并做访问控制。如果是 `axis` 服务，没有 `ACHandler`，则可以取出 `context` 在程序逻辑中做访问控制。具体内容如下文所述，也可以参照 `GOSTutorial` 工程中 `AxisPing` 服务的实现及配置。

3.3 开发服务

GOS 安装后会自带一个 `AxisPing` 服务，其服务实现代码见 `org.gos.tutorial.services`。其关键代码如下：

```
public String ping(String msg) {
    OperateContext context = null;
    MessageContext mc = MessageContext.getCurrentContext();
    if (mc == null) {
        String errMsg = "Can't get MessageContext, it maybe not run
in axis.";
        System.err.println(errMsg);
    } else {
        try {
            context = ForAxis.getOperateContext(mc);
        } catch (SOAPException e) {
            e.printStackTrace();
        }
    }
}
```

```
}
String prefix = "context: " + context;

if (context != null) {
    if (context.getUser() != null) {
        prefix += ", called by " +
            context.getUser().getUserHandle().getName();
    }
    if (context.getAgora() != null) {
        prefix += ", in agora " + context.getAgora().getName();
    }
}
msg = prefix + ", origin msg: " + msg;

return msg;
}
```

注意：这段代码中利用 `ForAxis.getOperateContext(mc)` 获得调用者身份以及所在的社区的信息。这是 GOS 对于开发安全的服务的一种支持。服务可以通过这些信息自己完成特定于本服务的访问控制。

3.4 部署服务

3.4.1 静态部署

在 GOS 安装后，已经部署了 5 种安全形式的 Ping 服务，分别叫做 `vsPingServiceSec`，`vsPingServiceNoSec`，`PingServiceSec`，`PingServiceNoSec`，`AxisPing`。这五种服务主要区别在于 Handler 的配置不同。其中 `vsPingServiceSec`，`vsPingServiceNoSec` 是 GOS 2.1 推荐的部署方式，`PingServiceSec`，`PingServiceNoSec` 是 GOS2 起支持的服务部署方式，`AxisPing` 是把 Ping 服务部署成一个纯 Axis 的服务。

可以直接使用已部署的服务，这步可以省略。

下面以 `vsPingServiceNoSec` 为例，讲解以上服务的部署过程。

如果需要重新部署，部署过程如下：

- 把编译好的源码打包，命名为 `gos-ping-service.jar`，拷贝到 `${gos.home}/jakarta-tomcat-5.0.28/webapps/axis/WEB-INF/lib/` 下，覆盖原来的同名的 jar。
- 修改 `${gos.home}/jakarta-tomcat-5.0.28/webapps/axis/WEB-INF/` 下的 `server-config.wsdd`，在其中加入以下部分。

```

<service name="vsPingServiceNoSec" provider="java:RPC">
  <requestFlow>
    <handler
type=" java:org.ict.gos.core.security.handler.GetAttachmentsHan
dler"/>
    <handler
type=" java:org.ict.gos.core.security.handler.VerifyCertsHandle
r"/>
  </requestFlow>
  <responseFlow>
    <handler
type=" java:org.ict.gos.core.security.handler.AddHandler"/>
  </responseFlow>
  <parameter name="allowedMethods" value="*" />
  <parameter name="className"
value="org.ict.gos.tutorial.services.Ping" />
</service>

```

其中 `GetAttachmentsHandler` 用于取出 SOAP 附件中的 `OperateContext`。
`VerifyCertsHandler` 用于验证 `OperateContext` 中携带的用户证书。
`AddHandler` 用于将 `OperateContext` 添加到 SOAP 消息的附件中。

- 重启 GOS

进入 GOS 安装目录 `{gos.InstallHome}`，执行 `gos.sh restart` 重启

3.4.2 动态部署

略。

3.5 添加资源

使用资源之前，必须将资源添加到 GOS 系统中。

添加资源采用 `ResourceClient` 的 `add` 接口。其定义如下：

```

public ResourceHandle add(String rControllerType, Object[] resInfo,
    String agoraID, String ownerID) throws GosException;

```

GOS 可以接入多种资源，包括服务和消息等，不同类型的资源由相应的资源控制器 `RController` 来识别和处理。加入资源时必须指明其资源控制器，由参数 `rControllerType` 给出。GOS 中已经实现了一系列的资源控制器。对服务的资源控制器是 `org.gos.core.rc.axis.AxisClient`。

每种资源控制器在添加对应资源时需要提供资源相关的一些参数，这些参数由 `resInfo` 给出。

添加资源时可以指定以用户身份 `ownerID` 添加到社区 `agoraID`。当 `ownerID` 为 `null` 时会以当前用户身份，当 `agoraID` 为 `null` 时会添加到当前社区。

本 tutorial 中添加资源的代码可以见 `org.gos.core.tutorial.PingClient.addResource` 方法。其关键代码如下：

```
// add a resource by current user in current agora.  
ResourceHandle rh = resourceClient.add(extSrvRControllerType,  
    new Object[] { resourceUrl, resourceName }, null, null);
```

资源添加后可以获得一个 `ResourceHandle`，其中包含此资源相关的各种信息。其中非常重要是其 `guid`，通过 `ResourceHandle.getGuid()` 可以得到。`Guid` 是此资源在全网唯一的标识。一般由系统自动产生。

3.6 删除资源

当所有人都不再需要资源时，可以将其删除。

删除资源通过 `org.gos.core.resource.client.ResourceClient` 的 `remove` 接口。其定义如下：

```
public void remove(String resourceID) throws GosException
```

其中 `resourceID` 指待删除资源的 `guid`。

删除资源的示例代码见 tutorial 中的 `org.gos.core.tutorial.PingClient.removeResource` 方法。其关键代码是：

```
resourceClient.remove(this.resourceID);
```

3.7 权限设置

添加资源到某个社区的动作必须由此社区管理员完成，默认当前用户（社区管理员）成为资源的属主，当然如果添加资源时指明了资源的 `owner`，则由此 `owner` 作为资源的属主。

资源的属主可以设置其权限，由资源的 `acl` 表示，可以通过 `ResourceHandle.getAcl` 和 `setAcl` 访问资源的 `acl`。ACL 是 `ugorwx` 的形式，与 UNIX 文件的权限类似。比如 `acl=rwxr-xr--`，表明属主拥有读写和执行权限，资源所在的组的用户可以读和执行，社区内的其他用户只能读。拥有读权限可以通过 `ResourceClient.readAttribute` 操作读资源的元信息，拥有写权限可以通过 `ResourceClient.writeAttribute` 操作修改修改资源的元信息。拥有执行权限可以通过 `ResourceClient.execute` 调用资源的各种操作。

资源加入到网格中时只在一个社区，最多只能被此社区的用户访问。如果想资源被其他社区访问，则需要资源的属主将资源的权限代理给其他社区。这个通过 `export` 操作完成。`Export` 可以将权限代理给一个或者多个社区，也可以用 `*` 表示全部社区。资源 `export` 后，相应社区可以通过 `link` 操作完成。`Export` 和 `link` 的 `gosShell` 中的命令分别为 `GexportRes` 和

GimportRes。

4. 编写 Java 应用

编写应用时，和 GOS 接口的部分只需要关注于如何访问资源。

4.1 访问资源

访问资源关键是使用 `ResourceClient.open`, `execute` 和 `close` 接口。示例代码见 `org.gos.core.tutorial.PingClient.executeResource` 接口，其关键内容如下：

```
OperateSession os = resourceClient.open(this.resourceID, null);
String pingStr = "ping a service at " + new Date();
RuntimeHandle rh = resourceClient.execute(os, "ping",
    new Object[] { pingStr }, true);
System.out.println("ping result : " + rh.getResult());
resourceClient.close(os);
```

4.1.1 open

访问任何一个资源前，都需要通过 `open` 动作绑定到所需访问的资源。

`Open` 接口如下：

接口描述：`OperateSession open(String resourceID, String option)`

输入参数：

`resourceID` 指代需要操作的资源

`Option` 给出 `open` 时所需的参数，具体如何取值由具体的资源决定

输出：

创建的 `OperateSession` 对象

抛出异常：

权限不够

资源不存在

4.1.2 Execute

`Open` 成功后，可以执行 `execute` 接口，执行对资源的操作，其接口定义如下：

接口描述：`RuntimeHandle Execute(String operateSessionID, String operationName, Object[] parameter, Boolean isSync)`

输入参数:

operateSessionID open 得到的 OperateSession 对象的 id

operationName 操作名

parameter 参数列表

isSync 是否同步方式

输出:

返回运行实体的指代物 (是一个 RuntimeHandle 对象, 对于 Web service, 可能代表了一个服务线程), 通过 RuntimeHandle 的 getResult()接口可以取得调用结果。同步时得到 RuntimeHandle 可立刻取得调用结果, 异步调用时由 RuntimeHandle 的 getResult()接口可能导致等待 (wait-by-necessary)。

抛出异常:

权限不够

资源不存在

4.1.3 Close

访问完成之后, 需要释放资源, 其接口定义如下:

接口描述: Void close(String operateSessionID)

输入参数:

operateSessionID open 得到的 OperateSession 对象的 id

输出:

无

抛出异常:

权限不够

资源不存在

4.1.4 关于 open, execute 和 close 的一些补充

GOS 中一般仅仅在 open 时检查用户权限, 在 open 和 close 之间可以执行多次 execute 执行此资源的多个操作, 这些操作都使用同样的权限。即使资源的真实的权限已经修改了, 其不会影响已经 open 的资源的操作。

4.1.5 安全特性

应用中可以使用 GripClient. getCurrentGripContext 获得一个上下文, 其中有本应用程序的用户身份和当前社区等信息。

但只有当应用以网程方式启动时, 才能取得 GripContext。

有两种标准的方法将程序以网程方式运行：

- 在 gosShell 中运行
- 利用网程的 API 启动程序

5. 编写 Webapp 应用

5.1.1 访问资源

Webapp 中如果需要访问资源，与 Java 应用完全一致，仍然使用 ResourceClient.open, execute 和 close 接口。

5.1.2 安全特性

Webapp 中可以通过 GripClient. getCurrentGripContext 获得一个上下文。但要保证得到的 context 才不是 null，必须做了以下两件事情：

- 部署 webapp 时设置 Web 应用 Filter
- 有一个登录程序，使用提供的 SessionClient 类中的 setSessionContext () 接口，将请求 Web Application 的 HttpServletRequest 请求对象、及调用者的身份信息传递给 setSessionContext ()

具体用法请参考 8.2 Webapp 开发。

6. 独立运行的 java 程序

很多时候，程序员可能希望脱离 gosShell 运行。这时一般推荐的做法是利用网程的 API 将程序以网程方式启动。这里我们实现一个 mygrun，用来将一个程序以网程方式运行。

6.1 实现自己的 grun

本 tutorial 中，实现了一个最简单的 mygrun，是 gosShell 提供的 grun 命令的简化版本。通过本例子，展示如何将一个程序以网程方式运行。

其最核心的代码如下：

```
public static void execApp(GripClient gclient, UserAuthStruct auth,
    AgoraHandle agora, String[] cmdargs) throws Exception {
```

```
GripHandle childgrip = gclient.create(auth, agora, cmdargs);
int pid = childgrip.getPid();

String systemType = System.getProperty("os.name");
if (systemType.equals("Linux")) {
    LinuxUtil.waitChild(pid);
} else if (systemType.indexOf("Windows") > -1) {
    WinUtil.waitChild(pid);
}
}
```

execApp 中，关键是创建网程。

gclient.create 即是网程的 create 接口，用某个用户身份（由 UserAuthStruct auth 定义），在某个社区（由 agora 定义），将某个程序（由 cmdargs）定义，以网程方式启动，并运行。

启动网程后，程序本身可以等待此网程运行结束，即函数中的 *waitChild* 动作。

6.2 运行时切换操作上下文

GOS 中每个程序运行时都会附着一个网程，其中提供了用户身份和当前社区等网程上下文信息（GripContext），这些信息在整个程序运行期间保持不变。

本质上 GOS 中的每一个系统调用都需要 OperateContext，称为操作上下文。如果程序以网程方式运行，则 GOS 会根据网程自动构造出访问资源的操作上下文，其用户与社区信息来自网程的设置。

操作上下文与网程上下文的关键的区别在于：操作上下文只对具体的这次调用生效，而网程在整个程序运行期间不变。操作上下文还包含对某个资源的访问控制 token。

ResourceClient 的 add, open, remove 接口都提供了一个带参数 OperateContext 的接口。通过显式的传入此 OperateContext，可以在运行时切换此次操作的操作上下文。

6.3 设置 gos.home

打开附带的工程 GOSTutorial，设置 java 虚拟机参数 -Dgos.home=., 其中 "." 指向一个目录，此虚拟机参数 gos.home 的目的是帮助找到相关配置文件。GOS 中约定，在 gos.home 所指的文件夹中必须有一个 conf 文件夹，其中至少需要有 namingserver.conf，其内容分别指向所要访问的 GOS 节点的相关服务路径。GOS 可以安装多个节点并互联，但应用程序必须要找到一个节点作为起始节点，通过此节点可以自由的访问全网格各个节点的资源。

6.4 修改配置文件中的 IP 和端口

修改 `namingserver.conf` 的 `localSiteUrl`，修改 IP 和端口使之指向一个真正的 GOS 节点。

6.5 所需其他文件

6.5.1 客户端 `wsdd` 文件

程序运行需要在当前目录放置客户端的 `wsdd` 文件，即 `gos-client-config*.wsdd`，共有 5 个。

6.5.2 动态链接库

GOS 运行需要一些动态链接库，具体包括：

`libc4linux.so/cmd4linux.dll`

`libNativeProcessing4Java.so/NativeProcessing4Java.dll`

`libNativeProcessing4Web.so/NativeProcessing4Web.dll`

为了让这些动态链接库生效，需要设置虚拟机参数：`-Djava.library.path=`具体的 `so` 或者 `dll` 的具体路径。

6.5.3 安全相关配置文件

需要在当前目录下准备 `cog.properties`，以及一个 CA 文件夹。

可以考虑本 tutorial 中自带的这些文件。

6.6 运行

进入 tutorial 工程，执行：

```
mygrun Admin_200@cngid.orgpwd "java -Dgos.home=. org.gos.core.tutorial.PingClient"
```

如果执行成功，可以看到如下信息：

...

```
ping result : context: #operate-context<org.gos.core.agora.client.AgoraHandle@a5
```

```
824a:org.gos.core.agora.client.AgoraHandle@a5824a,user:org.gos.core.user.client.
```

```
UserAuthStruct@d020d,grip-id:http://10.61.0.159:8080/E99E20F691D35CC42DC895F7302
```

```
215D9DBFD09AF, token:[B@1a488>, called by Admin218@vegasuite.ict.org, in agora A
```

gora218, origin msg: ping a service at Fri Dec 21 00:55:12 CST 2007

...

7. 上下文

7.1 设置操作上下文

ResourceClient 的 add, open 和 remove 都支持传入一个 OperateContext, 这样可以改变某次资源访问时提供的上下文。

7.2 设置和获取线程的操作上下文

如果想对同一个线程的某一组操作采用同样的上下文, GOS 提供了一个 GripClient.setThreadOperateContext 方法, 用于为当前线程设置一个操作上下文, 之后此线程的所有资源访问默认都用此上下文。

示例程序见 Tutorial 中的 ThreadOperateContextDemo。

采用线程的操作上下文可以使一个程序在一组操作中无需每次 open 时都设置 OperateContext。

采用线程的操作上下文的另一个好处是支持调试与分析。如果程序在 gosShell 中运行, 或者程序用 grun.sh 运行起来, 则程序是以 grun.sh 的子进程方式运行的, 很多调试工具, 或者性能分析工具都无法调试或者分析子进程, 不方便开发与分析。GOS 推荐的方式是先 login 在 setThreadOperateContext, 使程序具有上下文可以调试运行和分析。在程序调试或者分析完成后再去掉 login 和 setThreadOperateContext 的步骤, 真正放到 gsh 中运行。

可以用 GripClient.getCurrentThreadOperateContext 获取当前线程的 OperateContext。

7.3 获取资源端的操作上下文

客户端访问资源端时提供 OperateContext, GOS 系统在访问资源时会将 OperateContext 以某种方式安全地传到资源端。这样资源端可以根据此 OperateContext 获得本次访问的上下文, 从而自己完成所需的自定义的操作, 比如记录全局用户身份, 自己做访问控制。

不同类型的资源获得 OperateContext 的方式不同, 具体由处理此资源的 RController 决定。

对于服务，用 `org.gos.core.rc.axis.ForAxis` 类的 `getOperateContext` 系列接口完成。
对于消息，用 `org.gos.core.rc.activemq.ForJms` 类的 `getOperateContext` 系列接口完成。

7.4 获取网程上下文

如果程序是以网程方式运行，则我们可以通过 `GripClient.getCurrentGripContext()` 获得当前网程的 `GripContext`，即网程上下文。通过网程上下文可以获得用户身份和当前社区等信息。GOS 会根据 `GripContext` 自动构造出 `OperateContext`，用于具体的对资源的某次访问。

GOS 中支持 `javaapp` 和 `webapp` 两种类型的网程，都使用同样的 API 获得网程上下文。

7.5 开发中如何可靠的获取操作上下文？

默认情况下，GOS 系统会自动处理操作上下文的设置和获取问题，应用程序无需关心。

如果开发时需要获取操作上下文做一些自定义的操作，建议使用 GOS 提供的辅助类 `org.gos.core.util.GOSUtil` 的 `getOperateContext` 方法。这个类会首先从 `ThreadOperateContext` 获取，如果不成功，则尝试获取资源端的 `OperateContext`，最后尝试获取网程上下文并构造 `OperateContext`，这样可以保证开发的库可以被客户端程序、资源端程序使用。

在 `ThreadOperateContextDemo` 中，我们实现了一个 `getOperateContext`，其原理类似 `org.gos.core.util.GOSUtil` 的 `getOperateContext`，其代码示例如下：

```
public static OperateContext getOperateContext() {
    OperateContext context = null;

    // First try to get it from Current thread. 2007.12.28.
    // thread level.
    context = GripClient.getCurrentThreadOperateContext();
    if (context != null) {
        return context;
    }

    // then try to get it from service.
    try {
        // it may be in axis?
        MessageContext mc = MessageContext.getCurrentContext();
        if (mc == null) {
            // only log when debug mode.
        } else {
            context = ForAxis.getOperateContext(mc);
        }
    }
}
```

```
        if (context != null) {
            UserHandle hd = context.getUser().getUserHandle();
            System.out.println("OperateContext in service: user = "
                + hd.getGuid() + ", userName = " + hd.getName()
                + ", agora = " + context.getAgora().getGuid()
                + ", agoraName = " +
context.getAgora().getName()
                + ", gripID " + context.getGripId());

            return context;
        }
    } catch (Exception e) {
        // don't throw it now! just return a null
    }

    // then try to get it from grip context.
    // session level or grip level.
    try {
        GripContext gripContext =
GripClient.getCurrentGripContext();
        if (gripContext != null) {
            context = new OperateContext();
            context.setAgora(gripContext.getSimpleAgoraHandle());
            context.setUser(gripContext.getUserAuthStruct());
            context.setGripId(GripClient.getCurrentGripID());

            UserHandle hd = context.getUser().getUserHandle();
            System.out.println("OperateContext in grip: user = "
                + hd.getGuid() + ", userName = " + hd.getName()
                + ", agora = " + context.getAgora().getGuid()
                + ", agoraName = " + context.getAgora().getName()
                + ", gripID " + context.getGripId());

            return context;
        }
    } catch (Exception e) {
        // don't throw it now! just return a null
    }

    return null;
}
```

8. 网程的运行

8.1 Javaapp 开发

8.1.1 开发环境配置

将`{gos.home}/clinetLib` 目录下的 jar 包配置到 java 工程的 CLASSPATH 中；确认`{gos.home}/jakarta-tomcat-5.0.28/bin` 和 `{gos.home}/gosShell/binary` 目录下存在 `NativeProcessing4Java.so` 、 `NativeProcessing4Web.so` 和 `NativeProcessing4Java.dll` 、 `NativeProcessing4Web.dll`；确认`{gos.home}/jakarta-tomcat-5.0.28/webapps/axis/WEB-INF/lib` 下存在 `gos-grip-common.jar` 包。

8.1.2 开发接口使用介绍

提供给用户开发 javaapp 型网程的接口有以下八个：全由 `GripClient` 类提供：

返回类型	接口名称	简要说明
<code>GripHandle</code>	<code>create</code>	创建网程，启动网格应用
<code>String</code>	<code>getCurrentGripID</code>	获得当前网程 <code>gripId</code>
<code>GripContext</code>	<code>getCurrentGripContext</code>	获得当前网程 <code>Context</code>
<code>OperateContext</code>	<code>getCurrentThreadOperateContext</code>	从 <code>ThreadLocal</code> 中获得当前线程 <code>OperateContext</code>
	<code>setCurrentGripContext</code>	设置当前网程 <code>Context</code>
	<code>setThreadOperateContext</code>	将当前线程 <code>OperateContext</code> 设置到 <code>ThreadLocal</code> 中
<code>HashMap</code>	<code>gripStatus</code>	获得当前节点所有网程信息
	<code>kill</code>	杀掉一个特定的网程，同时释放网程资源空间中占有的资源。

- `create` 使用介绍

`create`函数的声明为：

```
public GripHandle create(String [] parameters) throws Exception
public GripHandle create(UserAuthStruct userAuthStruct, AgoraHandle
    simpleAgoraHandle, String[] parameters) throws Exception
public GripHandle create(UserAuthStruct userAuthStruct, AgoraHandle
    simpleAgoraHandle, Map property,String[] parameters)
    throws Exception
```

将执行 java 程序的整个命令字符串传递给 `create` 接口，则创建一个网程去执行指定的 java 程序（对应为本地的一个进程）。Property 为网程属性，类似于环境变量，该值可由用户自定义传入，若用户未制定 `property`，则创建的网程属性继承父网程的属性。创建网程 `grip` 的同时，将在服务端记录创建该网程的用户信息，包括 `UserAuthStruct`，

SimpleAuthAgoraStruct, Proxy 等身份信息, 所有这些信息以 GripContext 对象的形式保存在服务端 GripContainer 中。例:

```
String parameters = " java -Dgos.home =/xxx/xxx/gos3-xxx -classpath
                    /xxx/xxx:/xxx/xxx test.junit.xxxx";
String[] str = parameters.split(" ");
try {
//方法一：用户传入网程属性
GripClient gripClient = new GripClient();
    Map testMap = new HashMap();
        testMap.put("test", "setproperty");;
        javaappHander =
            gripClient.create(gc.getUserAuthStruct(), gc
                .getSimpleAgoraHandle(), testMap, str);
//方法二：直接继承父网程属性
gripClient.create(userAuthStruct, simpleAgoraHandle, str)
}
catch (Exception e) {
    e.printStackTrace();
}
```

- **getCurrentGripID** 使用介绍

getCurrentGripID 函数声明: **static** String getCurrentGripID() **throws Exception**
在某个网程内调用该接口, 获得当前网程的 gripId。例:

```
String gripID = GripClient.getCurrentGripID();
```

- **getCurrentGripContext** 使用介绍

getCurrentGripContext 函数声明: **static** GripContext getCurrentGripContext()
throws Exception

在某个网程内调用该接口, 获得当前网程的 Context; 例:

```
GripContext context = GripClient.getCurrentGripContext();
```

- **getCurrentThreadOperateContext** 使用介绍

getCurrentThreadOperateContext 函数声明:

OperateContext getCurrentThreadOperateContext()

从 ThreadLocal 中获得与当前线程级网程相关的 OperateContext。

```
OperateContext context = GripClient.getCurrentThreadOperateContext ();
```

- **setCurrentGripContext** 使用介绍

setCurrentGripContext 函数声明: **void** setCurrentGripContext

(**GripContext** gripContext) **throws Exception**

在某个网程内调用该接口, 可设置当前网程 Context; 例:

```
GripClient gc = new GripClient();
gc.setCurrentGripContext(currentGripContext);
```

- **setThreadOperateContext** 使用介绍
setThreadOperateContext 函数声明: **void** setThreadOperateContext(OperateContext gc)
设置当前线程级网程的 OperateContext, 将 OperateContext 保存到 ThreadLocal 中。

```
GripClient gc = new GripClient();
gc.setThreadOperateContext(threadOperateContext);
```

- **gripStatus** 使用介绍
gripStatus 函数声明:
HashMap gripStatus() **throws Exception**
HashMap gripStatus(**String** UserOriginalID, **String** instructionType) **throws Exception**
HashMap gripStatus(**String** UserOriginalID, **String** instructionType, **String** siteIP, **String** sitePort) **throws Exception**
获得当前节点上所有网程信息; 例:

```
HashMap map = new HashMap();
try {
    GripClient gripClient = new GripClient();
    map = gripClient.gripStatus();
} catch (Exception e) {
    e.printStackTrace();
}
```

- **kill** 使用介绍
kill函数声明: **void** kill(**String** gripID, **String** usrName, **String** usrPassword) **throws Exception**

调用该接口, 可杀掉指定 gripId 对应的网程; 例

```
try {
    String gripId = "10.61.0.61:8080/xxxxxxxxx";
    GripClient gripClient = new GripClient();
    gripClient.kill(gripId, "root", "root");
} catch (Exception e) {
    e.printStackTrace();
}
```

8.2 Webapp 开发

8.2.1 开发环境配置

除了需要做与 javaapp 相同的配置外，还需要确认在 jakarta-tomcat-5.0.28/common/lib 下存在 gos-grip-loader.jar 包。

8.2.2 开发接口介绍

SessionClient 介绍

SessionClient 类只包含一个接口 setSessionContext，该接口的完整声明为：

```
static void setSessionContext ( HttpServletRequest request,  
                                UserAuthStruct userAuthStruct,  
                                AgoraHandle agoraHandle);
```

该接口是提供给用户在编写 jsp 页面时调用的，主要功能是将登录获得的用户身份和社区信息存储在请求对象 request 中的 session 变量内；例由下节给出。

SessionFilter 介绍

SessionFilter 继承自 Fileter，主要重写了 Fileter 中的 doFilter () 函数。

Filter 提供对 Servlet 容器请求和相应对象进行检查和修改，它本身并不产生请求和相应对象，而是提供过滤作用。实现不同 Web 应用的 Filter 需要实现 javax.servlet.Filter 接口。在本系统中提供了 SessionFilter 类来对 Web 请求和相应对象进行预处理，主要功能是把对 Servlet 的请求对象 ServletRequest 保存在 ThreadLocal 中。

设置 Web Application 的 Filter 还必须修改 Web Application 的配置文件 web.xml，如下所示为在 web.xml 中添加的内容：

```
<filter>  
  <filter-name>session-filter</filter-name>  
  <filter-class>org.gos.core.grip.session.SessionFilter</filter-class>  
</filter>  
<filter-mapping>  
  <filter-name>session-filter</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>
```

ThreadlocalUtil 介绍

ThreadlocalUtil 提供 ThreadLocal 对象，该对象是一个与该线程绑定的线程变量，因此 webapp 可将某个用户的信息保存在与相应该用户的 Thread 的 ThreadLocal 变量中，因此在 Web 容器中，不同的请求对应不同的线程，而不同的线程中又保存着发起请求的不同用户的身份。

创建 webapp 型网程介绍

与创建 Java Application 型网程类似，创建 Web Application 型网程同样调用 GripClient 提供的 Create () 接口，但传递给 Create () 接口的参数必须是一个 Web Application 包，如打包 Web 应用后生成的 war 包；例：

```
try {
    GripClient.create(userAuthStruct,
        simpleAgoraHandle, new String[] { "./test.war" });
}
catch (Exception e) {
    e.printStackTrace();
}
```

创建 Web Application 时，同样将在服务端记录该 Web Application 的用户信息，包括 UserAuthStruct, SimpleAuthAgoraStruct, Proxy 等身份信息，所有这些信息以 GripContext 对象的形式保存在服务端 GripContainer 中。

与 Java Application 不同的是，保存用户身份信息的 GripContext 在 Web Application 中具有不同的意义：其中，iniUserAuthStruct 和 iniSimpleAgoraHandle 保存着创建该 Web Application 的原始用户的身份信息，而 UserAuthStruct 和 simpleAgoraHandle 保存着调用或登录该 Web Application 的各个不同用户的身份信息。

获取 webappcontext 介绍

同样调用 GripClient 中提供的 `getCurrentGripContext()` 接口来获取当前访问该 Web Application 的用户的 Context。但不同的是，在返回 Context 前，先取得保存在 ThreadLocal 中的 request，然后将保存在 request 的 session 对象中的访问 Web Application 的不同用户的身份信息存入 GripContext 的 UserAuthStruct 和 AuthAgoraStruct 两个字段，然后再返回。

8.2.3 开发实例

下面用一个简单的页面登录测试实例，介绍如何开发利用网程 (grip) 功能的 web 应用：该 Web 应用包含 3 个网页和一个必须的提供网程功能的 (grip) jar 包。

Jsp 网页内容：

- 用户登录页面 index.html

系统登录	
用户名	<input type="text" value="bearfly"/>
社区名	<input type="text" value="xiong"/>
密码	<input type="password"/>
<input type="button" value="登录"/> <input type="button" value="取消"/>	

代码如下：


```
        <input type="reset" value="取消">
    </div></td></tr>
</center>
</form>

</tbody>
</table>
</center>
</body>
</html>
```

- 验证并保存用户身份信息页面 login.jsp
login.jsp 代码如下：

```
<%@page language="java" contentType="text/html; charset=GB2312"
import="org.gos.core.grip.session.SessionClient"
import="org.gos.core.agora.client.*"
import="org.gos.core.user.client.*"
%>
<%
    if(!(request.getParameter("username")).trim().equals(""))
    && !(request.getParameter("agoraname")).trim().equals(""))
    {
        String userName = request.getParameter("username").trim();
        String agoraName = request.getParameter("agoraname").trim();

        byte[] proxy = new byte[10];
        proxy[0] = 1;
        UserHandle uh = new UserHandle();
        uh.setInitUserID("userID");
        uh.setOwnerDN("userDN");
        uh.setName(userName);
        uh.setInitAgoraID("ownerAgoraID");
        UserAuthStruct userAuthStruct = new UserAuthStruct(uh, proxy);

        AgoraHandle agoraHandle = new AgoraHandle();
        agoraHandle.setInitAgoraID("agoraID");
        agoraHandle.setName(agoraName);
        agoraHandle.setOwnerDN("agoraDN");
        agoraHandle.setInitUserID("ownerID");

        SessionClient.setSessionContext(request,userAuthStruct,agoraHandle);
        response.sendRedirect("main.jsp");
    }else{
        out.println("请输入用户名和社区名！");
    }
%>
```

- 获取并显示用户身份信息页面 main.jsp
以下将获得的 gripContext 显示在页面上：

Show JSP!

```
getSession username = bearfly
```

```
getSession agoraname = xiong
```

main.jsp 代码如下:

```
<% @page language="java"      import="org.gos.core.grip.client.GripClient"
                               import="org.gos.core.agora.client.*"
                               import="org.gos.core.user.client.*"
                               import="org.gos.core.grip.utils.*"
%>
<html>
  <head>
    <title>Show Page</title>
  </head>
  <body bgcolor="#CC99DD">
    <% !String strHello="Show JSP!";%>
    <h1><%=strHello%></h1>
<%
  GripContext webContext = GripClient.getCurrentGripContext();
  if( webContext != null)
  {
    if((webContext.getUserAuthStruct()!=null) && (webContext.getSimpleAgoraHandle() != null))
    {
      out.println("getSession username = "+
webContext.getUserAuthStruct().getUserHandle().getName();
%>
<p>
<%
      out.println("getSession agoraname = "+
webContext.getSimpleAgoraHandle().getName();
    }
    else{
      out.println("UserAuthStruct Or SimpleAgoraHandle is NULL!");
    }
  }else{
    out.println("GetWebContext is NULL!");
  }
}
```

jar 包内容:

Web 应用若用到网程 (grip) 功能, 必须在其中包含网程相关 jar 包, 名为 **gos-grip-common.jar**, 将其放置于 WebappName/WEB-INF/lib/目录下 (注: 网程功能的实现还需要一些相关类的支持, 因此该目录下除 **gos-grip-common.jar** 外还应放置一些其他必须的 jar 包) 在该 jar 包中直接涉及到的有关 web 应用的类有:

- 基本数据结构类

如: `UserAuthStruct`, `SimpleAgoraHandle` 和 `GripContext`。其中 `UserAuthStruct` 保存用户信息, `AgoraHandle` 保存社区信息, `GripContext` 是 `UserAuthStruct` 和 `AgoraHandle` 的容器。

- 存取用户社区信息

如: `SessionClient`, `SessionFilter`, `ThreadlocalUtil` 和 `GripClient` 类。在 `SessionClient` 中提供接口 `setSessionContext()`, 实现将不同用户的身份信息保存在此次 session 中, 不同的用户具有不同的 session。而 `GripClient` 提供的接口 `getCurrentGripContext()` 则是取出保存在 session 中的用户信息。

- 实现 `ThreadLocal` 功能类

如 `SessionFilter` 和 `ThreadlocalUtil` 两个类, `SessionFilter` 实现了 `javax.servlet.Filter` 接口, 并将其设置为该 Web Application 的 Filter, 其功能是将访问此次 Web Application 的 `ServletRequest` 请求对象保存在 `ThreadLocal` 中; `ThreadlocalUtil` 实现对保存在 `ThreadLocal` 中数据的存取, 在其中创建了一个静态的 `ThreadLocal` 对象。

9. API 完成系统管理

在 GOSTutorial 工程中, `org.gos.core.tutorial.SimpleManager` 类简单的介绍了一下如何利用 GOS 的 API 来进行系统管理。这个类设计的管理场景是首先由节点管理员登陆 GOS 的默认社区, 然后在默认社区中添加一个用户, 再添加一个社区, 并将新添加的这个用户设为该社区的属主。接着由这个新用户登录到他自己的社区, link 一个默认社区的用户到自己的社区。最后, 做一些“清理工作”。包括 unlink 用户, 删除用户, 删除社区等操作。

主要通过 `AgoraClient`、`UserClient`、`SystemCfgClient` 等的相关 API 来实现。具体内容可以参见该类的源码。

10. 开发 gsh 命令

目前开发 gsh 的命令如下几个约定:

- 1) 首先为命令起一个名字, 目前 GOS 命令的命令名字规范是以 g 开头的小写字母组合, 并且尽可能不与节点本地系统命令重名
- 2) 命令实现需要提供 -h 选项, 提供命令的帮助说明, 主要指命令功能和格式
- 3) 如果有异常需要中止命令执行的时候, 退出并返回 -1

- 4) 编译后的 jar 放在 \$GOS_HOME/clientLib/ 目录下
- 5) 在 \${gsh.home}/conf/command.alias 中追加如下两行内容:

```
#sec commandType
alias commandname='grun execwithargs'
```

如果提供了一组命令归属于同一个类别,只需要在同一个 #sec commandType 后面定义一组 alias 即可。

遵守这些规定只不过允许 ghelph 命令输出的内容中包含该命令的相关信息,而且在 gsh 环境中可以直接输入命令名字执行网格命令,因此普通的 GOS 应用不需要遵守这些规定。

举例来说,如果要开发一个命令查看当前网格上有哪些社区的命令,用 java 语言实现,具体代码参见附带的工程 GOSTutorial 中的 org.gos.core.tutorial.GagoraInfoCMD 类的实现。首先为命令起一个名字,这里起名为 glsa,然后使用 commons-cli.jar 中提供的各种类解析命令行参数,其中提供一个 -h 选项,输出帮助信息,比如:

```
usage: glsa [-h][-i guid][-n name][-o user][-l]
to list agora info in the grid
-h    print help for the command
-i    specify the guid of the agora
-l    give a detailed information
-n    specify the name of the agora
-o    specify the owner name of the agora
```

在命令的实现中会调用 GOS 提供的接口实现相应的功能,当有异常发现需要中止程序执行的时候,需要使用 System.exit(-1)退出程序。

命令代码开发完毕后,将编译好的 java 类文件打成一个 jar 包,放到 \$GOS_HOME/clientLib/目录下,这样在启动 gsh 之后就可以通过命令

```
grun java -Dgos.home=$GOS_HOME org.gos.core.tutorial.GagoraInfoCMD
```

执行命令。说明命令实现没有问题。然后在 {GSH_HOME}/conf/command.alias 中追加如下两行内容:

```
#sec agora
alias glsa='grun java -Dgos.home=$GOS_HOME org.gos.core.tutorial.GagoraInfoCMD'
```

这样定义之后,重新启动 gsh,即可在 gsh 执行环境中通过输入 glsa 执行该命令。用户也可以在 gsh 中通过 ghelph 命令查看到新增的命令名字。

11. 开发资源控制器

略。