Red5

手册

翻译：林溪

2011/9/10

1. 常见问题
2. 常见问题
3. 什么是Red5？

Red5是一个用Java语言编写的开源Flash RTMP 服务器，支持以下特点：  
1 视频、音频流（FLV和MP3）  
2 录制客户端流  
3 共享对象  
4 发布直播流  
5 远程调用

1. Red5主张什么？

Red5 was the "one who did the impossible"

1. 从FMS到Red5有迁移手册吗？

有。[Migration Guide](http://trac.red5.org/wiki/Documentation/UsersReferenceManual/GettingStarted/03-Migration-Guide)

1. 如何创建新程序？

Red5把所有的程序放在根目录下的 “webapps”目录中，根目录下doc文件夹中的templates文件夹中有模板程序，可复制到webapps下创建。创建的程序里应该包含一个 “WEB-INF”目录，WEB-INF目录中包含配置文件的编译好的classes。

1. 有那些配置文件

参见第二章。

1. 邮件列表

[Mailing list](http://trac.red5.org/wiki/MailingLists)

1. Red5用了那些端口

http.port=5080 // tomcat or jetty servlet container   
rtmp.port=1935 // traditional rtmp  
rtmpt.port=8088 // rtmp tunneled over http   
mrtmp.port=9035 // used with an edge/origin setup   
proxy.source\_port=1936 // used to debug  
这些默认的端口在 [RED5\_HOME]\conf\red5.properties 文件里更改。

1. Red5项目里都有谁？

项目经理：  
Chris Allen (mrchrisallen@gmail.com)  
John Grden (johng@acmewebworks.com)  
活跃成员  
Dominick Accattato (daccattato@gmail.com)  
Steven Gong (steven.gong@gmail.com)   
Paul Gregoire (mondain@gmail.com)   
Thijs Triemstra (info@collab.nl)   
Dan Rossi (electroteque@gmail.com)   
Anton Lebedevich ([mabrek@gmail.com](mailto:mabrek@gmail.com))

1. 配置文件
2. conf 目录
3. jetty.xml

这个文件用来配置HTTP Server和Servlet容器，在默认的5080端口上运行所有可用的接口。详情可参考[jetty 官方网站](http://jetty.mortbay.org/jetty6/" \t "_blank)

1. keystore

包含一个简单用于安全连接的私有密钥和认证

1. log4j.properties

配置日志的级别和输出，详情可参考[log4j官方网站](http://logging.apache.org/log4j/docs/" \t "_blank)。

1. realm.properties (Jetty)

为保护区域定义用户名和密码  
格式如下：  
<username>: <password>[,<rolename> ...]

1. tomcat-users.xml (Tomcat)

tomcat用户名和密码，关于tomcat realms可参考[官方网站](http://tomcat.apache.org/tomcat-5.5-doc/catalina/docs/api/org/apache/catalina/realm/package-summary.html)。

1. red5.globals

配置Red5使用的默认上下文配置文件的路径，默认为 /webapps/red5-default.xml。

1. red5.properties

配置主机地址和端口号，Http 和 rtmp的地址和端口。

1. red5.xml

主要配置文件，也是Red5加载的第一个配置文件，加载 red5-common.xml和red5-core.xml以及剩下的构建服务器的配置。可以配置J2EE容器。

使用 Jetty容器  
<bean id="jetty6.server" class="org.red5.server.JettyLoader" init-method="init" autowire="byType" />  
使用Tomcat容器  
<bean id="tomcat.server" class="org.red5.server.TomcatLoader" init-method="init" estroy-method="shutdown">  
 ... cut for brevity ...   
</bean>

1. red5-common.xml

包含对象序列化和反系列化的相关配置，Red5用的网络协议编码（Mina对象序列化编码）和视频编码的配置。所有的子上下文用到的类也在该文件中配置。  
对象缓存也在该文件里配置。目前有四种缓存可以配置：  
1 simple byte-buffers  
<bean id="object.cache" class="org.red5.server.cache.CacheImpl" init-method="init" autowire="byType">  
 <property name="maxEntries"><value>5</value></property>  
</bean>  
2 WhirlyCache  
<bean id="object.cache" class="org.red5.server.cache.WhirlyCacheImpl" init-method="init" autowire="b

<property name="maxEntries" value="5" />   
<property name="cacheConfig">   
<bean class="com.whirlycott.cache.CacheConfiguration">   
<property name="name" value="flv.cache" />   
<property name="maxSize" value="5" />   
<!-- This policy removes cached items, biased towards least frequently used (LFU) Items -->   
<property name="policy"><value>com.whirlycott.cache.policy.LFUMaintenancePolicy</value></property>   
<!-- This policy removes cached items, biased towards least recently used (LRU) Items -->   
<!-- property name="policy"><value>com.whirlycott.cache.policy.LRUMaintenancePolicy</value></property>   
<!-- This policy removes cache items in the order in which they were added -->  
Configuration files used by Red5   
<!-- property name="policy"><value>com.whirlycott.cache.policy.FIFOMaintenancePolicy</value></property>   
<!-- A predicate for filtering Collections of Items based on their expiration time -->   
<!-- property name="policy"><value>com.whirlycott.cache.policy.ExpirationTimePredicate</value></property>   
<!-- property name="backend"><value>com.whirlycott.cache.impl.ConcurrentHashMapImpl</value></property>   
<property name="backend"><value>com.whirlycott.cache.impl.FastHashMapImpl</value></property>   
</bean>   
3 Ehcache  
 <bean id="object.cache" class="org.red5.server.cache.EhCacheImpl" init-method="init">   
 <property name="diskStore" value="java.io.tmpdir" />   
 <property name="memoryStoreEvictionPolicy" value="LFU" />   
 <property name="cacheManagerEventListener"><null/></property>   
 <property name="cacheConfigs">   
 <list>   
 <bean class="net.sf.ehcache.config.CacheConfiguration">   
 <property name="name" value="flv.cache" />   
 <property name="maxElementsInMemory" value="5" />   
 <property name="eternal" value="false" />   
 <property name="timeToIdleSeconds" value="0" />   
 <property name="timeToLiveSeconds" value="0" />   
 <property name="overflowToDisk" value="false" />   
 <property name="diskPersistent" value="false" />   
 </bean>   
 </list>   
 </property>   
 </bean>   
4 NoCache  
<bean id="object.cache" class="org.red5.server.cache.NoCacheImpl"/>

可配置远程客户端连接池连接数

1. red5-core.xml

配置网络服务。RTMP和 RTMPT。  
当使用Jetty容器时，red5-rtmpt.xml 可配置RTMPT服务器。  
Jetty：  
<bean id="rtmpt.server" class="org.red5.server.net.rtmpt.RTMPTLoader" init-method="init" autowire="bType"/>  
Tomcat：  
<bean id="rtmpt.server" class="org.red5.server.net.rtmpt.TomcatRTMPTLoader" init-method="init" autowire="bType"> </bean>

1. red5-rtmpt.xml

在RTMPT URLs 和servlets 之间建立映射关系，RTMPT运行在8088端口上。  
详情请参照[Jetty官方网站](http://jetty.mortbay.org/jetty6/)

1. web.xml (Tomcat)

Tomcat用配置文件

1. web-default.xml (Jetty)

Jetty用配置文件

1. Webapp 目录
2. red5-web.xml

Red程序配置文件。  
Java Application  
<bean id="web.handler" class="org.red5.server.webapp.oflaDemo.Application" singleton="true" />

Javascript / Rhino application  
<bean id="web.handler" class="org.red5.server.script.rhino.RhinoScriptFactory">   
 <constructor-arg index="0" value="classpath:applications/main.js"/>   
 <!-- Implemented interfaces -->   
 <constructor-arg index="1">   
 <list>   
 <value>org.red5.server.api.IScopeHandler</value>   
 <value>org.red5.server.adapter.IApplication</value>   
 </list>   
 </constructor-arg>   
 <!-- Extended class -->   
 <constructor-arg index="2">   
 <value>org.red5.server.adapter.ApplicationAdapter</value>   
 </constructor-arg>   
</bean>

Ruby application  
<bean id="web.handler" class="org.red5.server.script.jruby.JRubyScriptFactory">   
 <constructor-arg index="0" value="classpath:applications/main.rb"/>   
 <constructor-arg index="1">   
 <list>   
 <value>org.red5.server.api.IScopeHandler</value>   
 <value>org.red5.server.adapter.IApplication</value>   
 </list>   
 </constructor-arg>   
</bean>

1. 从FMS迁移

这一章描述了FMS和Red5之间API的差异，旨在帮助从现有的FMS应用迁移到Red5。

服务器端程序的最大功能之一就是监听客户端连接或客户端断开连接，或新实例的创建。

1. IScopeHandler 接口

Scope Handle 在一个scope对象里控制action，并且Scope Handle被分发了所有事件。能细粒性的控制action的can\* 方法的执行，能为on\*方法提供scope里发生的事件的详细报告。这是创建一个应用程序用户必须实现的一个核心的接口，通过接口的一些方法可以得到当前线程的可用的连接。

addChildScope(IBasicScope scope); 添加子scope  
connect(IConnection conn, IScope scope, Object[] params) 连接scope  
disconnect(IConnection conn, IScope scope); 断开scope连接  
join(IClient client, IScope scope); 加入scope  
leave(IClient client, IScope scope); 离开scope  
removeChildScope(IBasicScope scope); 删除子scope  
serviceCall(IConnection conn, IServiceCall call); 调用Service  
start(IScope scope) scope第一次创建时调用  
stop(IScope scope) scope销毁之前调用

1. ApplicationAdapter类

通常作为程序的基类，需要注意的是这个类是单线程执行的。如果想获得最好的表现，可以用MultiThreadApplicationAdapter代替。使用这个类可能会导致使用\*Connec或\*Join方法接受一个客户端要花费比较长的时间，所以推荐使用多线程版本的。

Connect(IConnection conn, IScope scope, Object[] params)  
disconnect(IConnection conn, IScope scope)  
join(IClient client, IScope scope)  
start(IScope scope)  
stop(IScope scope)

appStart roomStart appStop roomStop appConnect roomConnect appJoin roomJoin appDisconnect roomDisconnect appLeave roomLeave

1. MultiThreadedApplicationAdapter类

这个类提供共享对象，流，连接和日程服务的操作方法，ApplicationAdapter是一个application级的scope，如果在程序中要处理流的进程需要实现 IStreamAwareScopeHandler接口和实现处理方法。ApplicationAdapter能为中断流，用户授权等提供有用的事件句柄。在子类的添加的所有方法在客户端可以通过NetConnection.call方法调用，这一点和FMS有所区别，FMS需要传递一个客户端对象到服务器端才能调用。Red5提供更为便利的方法进行远程调用。

addListener(IApplication listener)  
addScheduledJob(int interval, IScheduledJob job)  
addScheduledJobAfterDelay(int interval, IScheduledJob job, int delay)  
addScheduledOnceJob(Date date, IScheduledJob job)  
appConnect(IConnection conn, Object[] params)  
appDisconnect(IConnection conn)  
appJoin(IClient client, IScope app)  
appLeave(IClient client, IScope app)  
appStart(IScope app)  
appStop(IScope app)  
cancelGhostConnectionsCleanup()  
clearSharedObjects(IScope scope, String name)  
connect(IConnection conn, IScope scope, Object[] params)  
createSharedObject(IScope scope, String name, boolean persistent)  
disconnect(IConnection conn, IScope scope)  
FCPublish(String streamName)  
FCUnpublish()  
getBroadcastStream(IScope scope, String name)  
getBroadcastStreamNames(IScope scope)  
getClientTTL()  
getGhostConnsCleanupPeriod()  
getListeners()  
getOnDemandStream(IScope scope, String name)  
getScheduledJobNames()  
getSharedObject(IScope scope, String name)  
getSharedObject(IScope scope, String name, boolean persistent)  
getSharedObjectNames(IScope scope)  
getSharedObjectSecurity()  
getStreamLength(String name)  
getStreamPlaybackSecurity()  
getStreamPublishSecurity()  
getSubscriberStream(IScope scope, String name)  
hasBroadcastStream(IScope scope, String name)  
hasOnDemandStream(IScope scope, String name)  
hasSharedObject(IScope scope, String name)  
join(IClient client, IScope scope)  
killGhostConnections()  
leave(IClient client, IScope scope)  
measureBandwidth()  
measureBandwidth(IConnection conn)  
registerSharedObjectSecurity(ISharedObjectSecurity handler)  
registerStreamPlaybackSecurity(IStreamPlaybackSecurity handler)  
registerStreamPublishSecurity(IStreamPublishSecurity handler)  
rejectClient()  
rejectClient(Object reason)  
removeListener(IApplication listener)  
removeScheduledJob(String name)  
roomConnect(IConnection conn, Object[] params)  
roomDisconnect(IConnection conn)  
roomJoin(IClient client, IScope room)  
roomLeave(IClient client, IScope room)  
roomStart(IScope room)  
roomStop(IScope room)  
scheduleGhostConnectionsCleanup()  
setClientTTL(int clientTTL)  
setGhostConnsCleanupPeriod(int ghostConnsCleanupPeriod)  
start(IScope scope)  
stop(IScope scope)  
streamBroadcastClose(IBroadcastStream stream)  
streamBroadcastStart(IBroadcastStream stream)  
streamPlaylistItemPlay(IPlaylistSubscriberStream stream, IPlayItem item, boolean isLive)  
streamPlaylistItemStop(IPlaylistSubscriberStream stream, IPlayItem item)  
streamPlaylistVODItemPause(IPlaylistSubscriberStream stream, IPlayItem item, int position)  
streamPlaylistVODItemResume(IPlaylistSubscriberStream stream, IPlayItem item, int position)  
streamPlaylistVODItemSeek(IPlaylistSubscriberStream stream, IPlayItem item, int position)  
streamPublishStart(IBroadcastStream stream)  
streamRecordStart(IBroadcastStream stream)  
streamSubscriberClose(ISubscriberStream stream)  
streamSubscriberStart(ISubscriberStream stream)  
unregisterSharedObjectSecurity(ISharedObjectSecurity handler)  
unregisterStreamPlaybackSecurity(IStreamPlaybackSecurity handler)  
unregisterStreamPublishSecurity(IStreamPublishSecurity handler)

1. 连接方法的执行顺序

假设连接到 rtmp://server/app/room1/room2，  
连接建立时，各个Scope的connect 方法被调用。

app (-> appConnect)  
room1 (-> roomConnect)  
room2 (-> roomConnect)  
当连接第一次建立后，各个Scope的join方法被调用  
app (-> appJoin)  
room1 (-> roomJoin)  
room2 (-> roomJoin)  
当同一个客户端第二次建立连接时，只有connect方法被调用。appStart方法只有Red5启动时被调用一次。roomStart方法只有第一个客户端连接到room时被调用。

1. 接受和拒绝客户端

提供了方法来接受和拒绝新客户端的连接。

1. 当前的连接和客户端

Red5提供了两种方法活得当前的连接和客户端。  
Red5接口  
getClient()  
getConnection()  
getConnectionLocal()  
getContext()  
getScope()  
getVersion()  
setConnectionLocal(IConnection connection)  
第一种方法：  
**import** org.red5.server.api.IClient**;**  
**import** org.red5.server.api.IConnection**;**  
**import** org.red5.server.api.IScope**;**  
**import** org.red5.server.api.Red5**;**  
**public** **void** **whoami()** **{**  
 IConnection conn **=** Red5**.**getConnectionLocal**();**  
 IClient client **=** conn**.**getClient**();**   
 IScope scope **=** conn**.**getScope**();****}**

第二种方法  
import org.red5.server.api.IClient;   
import org.red5.server.api.IConnection;   
import org.red5.server.api.IScope;   
public void whoami(IConnection conn) {   
 IClient client = conn.getClient();   
 IScope scope = conn.getScope();   
}

1. handler

为了客户端通过RTMP连接，调用服务器端程序业务逻辑，Red5可以配置handler供客户端调用。  
有两种方法注册handler：

客户端调用：  
nc = new NetConnection();   
nc.connect("rtmp://localhost/myapp");   
nc.call("handler.method", nc, "Hello world!");

在配置文件里配置：  
<bean id="sample.service" class="com.fancycode.red5.HandlerSample" singleton="true" />

在程序中注册：  
public boolean appStart(IScope app) {   
 if (!super.appStart(scope))   
 return false;   
 Object handler = new com.fancycode.red5.HandlerSample();   
 app.registerServiceHandler("sample", handler);   
 return true;   
}  
需要注意的是，只有Application scope才能使用 sample Handler，子scope里无法访问，如果想在Room里使用，必须在scope的roomStart里注册。

1. 调用客户端方法

在red5程序端调用客户端方法，必须获得当前的连接对象  
import org.red5.server.api.IConnection;   
import org.red5.server.api.Red5;   
import org.red5.server.api.service.IServiceCapableConnection;   
IConnection conn = Red5.getConnectionLocal();

如果连接对象实现了IServiceCapableConnection接口，就可以调用客户端的方法。  
if (conn instanceof IServiceCapableConnection) {   
 IServiceCapableConnection sc = (IServiceCapableConnection) conn;   
 sc.invoke("the\_method", new Object[]{"One", 1});   
}

IServiceCapableConnection接口  
invoke(IServiceCall call)  
invoke(IServiceCall call, int channel)  
invoke(String method)  
invoke(String method, IPendingServiceCallback callback)  
invoke(String method, Object[] params)  
invoke(String method, Object[] params, IPendingServiceCallback callback)  
notify(IServiceCall call)  
notify(IServiceCall call, int channel)  
notify(String method)  
notify(String method, Object[] params)

如果需要回调的结果，需要提供一个实现了IPendingServiceCallback接口的类。  
import org.red5.server.api.service.IPendingService;   
import org.red5.server.api.service.IPendingServiceCallback;   
class MyCallback implements IPendingServiceCallback {   
 public void resultReceived(IPendingServiceCall call) {   
 // Do something with "call.getResult()"   
 }   
}  
调用代码如下：  
if (conn instanceof IServiceCapableConnection) {  
 IServiceCapableConnection sc = (IServiceCapableConnection) conn;  
 sc.invoke("the\_method", new Object[]{"One", 1}, new MyCallback());  
}

1. 共享对象
   1. ISharedObjectService接口

clearSharedObjects(IScope scope, String name)  
createSharedObject(IScope scope, String name, boolean persistent)  
getSharedObject(IScope scope, String name)  
getSharedObject(IScope scope, String name, boolean persistent)  
getSharedObjectNames(IScope scope)  
hasSharedObject(IScope scope, String name)

使用ShareObject时，需要注意的是ShareObject所在的域。  
可以在roomStart 方法中创建一个新的ShareObject对象。  
import org.red5.server.adapter.ApplicationAdapter;   
import org.red5.server.api.IScope;   
import org.red5.server.api.so.ISharedObject;   
public class SampleApplication extends ApplicationAdapter {  
 public boolean roomStart(IScope room) {   
 if (!super.roomStart(room))   
 return false;   
 createSharedObject(room, "sampleSO", true);   
 ISharedObject so = getSharedObject(room, "sampleSO");   
 // Now you could do something with the shared object...   
 return true;   
 }   
}  
每当第一个用户连接到room时，sampleSO 共享对象将被调用。  
如果要在全局范围内使用使用共享对象，可以在appStart中创建。

* 1. 服务器端监听

要像FMS的onSync事件那样监听共享对象中的更新事件，需要实现ISharedObjectListener接口。  
ISharedObjectListener接口  
onSharedObjectClear(ISharedObjectBase so)  
onSharedObjectConnect(ISharedObjectBase so)  
onSharedObjectDelete(ISharedObjectBase so, String key)  
onSharedObjectDisconnect(ISharedObjectBase so)  
onSharedObjectSend(ISharedObjectBase so, String method, List params)  
onSharedObjectUpdate(ISharedObjectBase so, IAttributeStore values)  
onSharedObjectUpdate(ISharedObjectBase so, Map<String,Object> values)  
onSharedObjectUpdate(ISharedObjectBase so, String key, Object value)  
  
import org.red5.server.api.so.ISharedObject;   
import org.red5.server.api.so.ISharedObjectListener;   
public class SampleSharedObjectListener  
Migration Guide implements ISharedObjectListener {   
 public void onSharedObjectUpdate(ISharedObject so, String key, Object value) {   
 // The attribute &lt;key&gt; of the shared object &lt;so&gt;   
 // was changed to &lt;value&gt;.   
 }  
 public void onSharedObjectDelete(ISharedObject so, String key) {   
 // The attribute &lt;key&gt; of the shared object &lt;so&gt; was deleted.   
 }   
 public void onSharedObjectSend(ISharedObject so, String method, List params) {   
 // The handler &lt;method&gt; of the shared object &lt;so&gt; was called   
 // with the parameters &lt;params&gt;.   
 }   
 // Other methods as described in the interface...   
}

注册监听器：  
ISharedObject so = getSharedObject(scope, "sampleSO");   
so.addSharedObjectListener(new SampleSharedObjectListener())

* 1. 服务器端操作共享对象

在服务器端改变ShareObject  
ISharedObject so = getSharedObject(scope, "sampleSO");   
so.setAttribute("fullname", "Sample user");

如果需要对共享对象进行一系列的操作，而只分发一个更新事件，可以使用beginUpdate和endUpdate方法  
ISharedObject so = getSharedObject(scope, "sampleSO");   
so.beginUpdate();   
so.setAttribute("One", "1");   
so.setAttribute("Two", "2");   
so.removeAttribute("Three");   
so.endUpdate();

* 1. Handler

可以像注册application handler一样，注册共享对象的handler，供客户端调用。  
在Flash客户端可以通过 remote\_so.send(<handler>, <args>) 调用注册的handler。  
package com.fancycode.red5;   
class MySharedObjectHandler {   
 public void myMethod(String arg1) {   
 // Now do something   
 }   
}   
...   
ISharedObject so = getSharedObject(scope, "sampleSO");   
so.registerServiceHandler(new MySharedObjectHandler());

handler也可以指定一个名字注册：  
ISharedObject so = getSharedObject(scope, "sampleSO");   
so.registerServiceHandler("one.two", new MySharedObjectHandler());  
方法可以通过one.two.myMethod调用。

另外一种方法注册事件handler是在red5-web.xml中配置，与配置application handler类似，需要注意的是bean的id必须以共享对象名开始，以soservice结束，MySharedObjectHandler在red5-web.xml中配置如下：  
<bean id="sampleSO.one.two.soservice" class="com.fancycode.red5.MySharedObjectHandler" singleton="true" />

1. 持久化

持久化能保持对象的属性，当服务器重启后，FMS在服务器端使用本地的共享对象。  
red5允许任意的对象持久化，需要做的就是实现IPersistable接口。

对象的序列化和反序列化：  
import java.io.IOException;   
import org.red5.io.object.Input;   
import org.red5.io.object.Output;   
import org.red5.server.api.persistence.IPersistable;   
class MyPersistentObject implements IPersistable {   
 // Attribute that will be made persistent   
 private String data = "My persistent value";   
 void serialize(Output output) throws IOException {   
 // Save the objects's data.   
 output.writeString(data);   
 }   
 void deserialize(Input input) throws IOException {  
 // Load the object's data.   
 data = input.readString();   
 }   
 // Other methods as described in the interface...   
}

保存和读取对象  
import org.red5.server.adapter.ApplicationAdapter;   
import org.red5.server.api.IScope;   
import org.red5.server.api.Red5;   
import org.red5.server.api.persistence.IPersistenceStore;   
class MyApplication extends ApplicationAdapter {   
 private void saveObject(MyPersistentObject object) {   
 // Get current scope.   
 IScope scope = Red5.getConnectionLocal().getScope();   
 // Save object in current scope.  
 scope.getStore().save(object);  
 }   
 private void loadObject(MyPersistentObject object) {   
 // Get current scope.   
 IScope scope = Red5.getConnectionLocal().getScope();   
 // Load object from current scope.   
 scope.getStore().load(object);   
 }   
}

如果程序里不用自定义对象，数据必须保存复用，可以通过IAttributeStore 接口添加到IScope里，在scope里只要以不IPersistable.TRANSIENT\_PREFIX开头的属性都会被保存。  
在后端存贮对象也是可以配置的，持久化可以配置在内存或文件系统中。  
当使用文件系统持久化时，对象文件将被创建，路径为 "webapps/<app>/ persistence/<type>/<path>/<name>.red5"  
例如：. 名为"theSO"的共享对象，连接为" rtmp://server/myApp/room1"，创建的文件的路径为： webapps/myApp/persistence/ SharedObject/room1/theSO.red5

IPersistable接口  
deserialize(Input input)  
getLastModified()  
getName()  
getPath()  
getStore()  
getType()  
isPersistent()  
serialize(Output output)  
setName(String name)  
setPath(String path)  
setPersistent(boolean persistent)  
setStore(IPersistenceStore store)

1. 周期性事件

如果程序要执行有规律的任务，在FMS里可以使用 setInterval 方法周期性的住行。  
Red5提供日程服务类ISchedulingService接口，ApplicationAdapter已经实现了该接口，可以注册一个实现了IScheduledJob接口的对象，并在指定的时间里执行对象的方法。

import org.red5.server.api.IScope;   
import org.red5.server.api.IScheduledJob;   
import org.red5.server.api.ISchedulingService;   
import org.red5.server.adapter.ApplicationAdapter;  
class MyJob implements IScheduledJob {  
 public void execute(ISchedulingService service) {  
 // Do something   
 }   
}   
public class SampleApplication extends ApplicationAdapter {   
 public boolean roomStart(IScope room) {   
 if (!super.roomStart(room))   
 return false;   
 // Schedule invokation of job every 10 seconds.   
 String id = addScheduledJob(10000, new MyJob());   
 room.setAttribute("MyJobId", id);  
 return true;   
 }   
}

停止执行  
public void roomStop(IScope room) {   
 String id = (String) room.getAttribute("MyJobId");  
 removeScheduledJob(id);   
 super.roomStop(room);   
}

1. 远程调用

Red5提供了远程Service，远程调用允许非rtmp客户端调用，也允许Red5调用其他服务器上的方法。  
远程service的注册和application handler的注册相同。  
要让Red5允许远程Service，必须在web.xml里添加以下servlet：  
<servlet>   
 <servlet-name>gateway</servlet-name>   
 <servlet-class>   
 org.red5.server.net.servlet.AMFGatewayServlet   
 </servlet-class>   
</servlet>

<servlet-mapping>   
 <servlet-name>gateway</servlet-name>   
 <url-pattern>/gateway/\*</url-pattern>   
</servlet-mapping>  
全局域中调用的路径为  
<http://localhost:5080/myApp/gateway>  
调用子域中的service，路径为：  
<http://localhost:5080/myApp/gateway/room1/room2>

1. 远程客户端

RemotingClient类  
addHeader(String name, boolean required, Object value)  
invokeMethod(String method, Object[] params)  
invokeMethod(String method, Object[] methodParams, IRemotingCallback callback)  
processHeaders(org.apache.mina.common.ByteBuffer in)  
removeHeader(String name)  
resetCredentials()  
setCredentials(String userid, String password)  
setThreadPool(ThreadPool threadPool)

远程客户端调用实例：  
import org.red5.server.net.remoting.RemotingClient;   
String url = "http://server/path/to/service";   
RemotingClient client = new RemotingClient(url);   
Object[] args = new Object[]{"Hello world!"};   
Object result = client.invokeMethod("service.remotingMethod", args);  
默认将设置30秒超时，可以给构造函数传递第二个参数决定最大超时毫秒数。

远程的数据头 AppendToGatewayUrl, ReplaceGatewayUrl 和 RequestPersistentHeader将被Red5的远程客户端自动处理。

如果方法调用很费时，为了不阻塞线程，Red5允许异步的调用，回调对象需要实现IRemotingCallback接口，异步调用实例：  
import org.red5.server.net.remoting.RemotingClient;   
import org.red5.server.net.remoting.IRemotingCallback;   
public class CallbackHandler implements IRemotingCallback {   
 void errorReceived(RemotingClient client, String method, Object[] params, Throwable error) {   
 // An error occurred while performing the remoting call.   
 }   
 void resultReceived(RemotingClient client, String method, Object[] params, Object result) {   
 // The result was received from the server.   
 }   
}

String url = "http://server/path/to/service";   
RemotingClient client = new RemotingClient(url);   
Object[] args = new Object[]{"Hello world!"};   
IRemotingCallback callback = new CallbackHandler();   
client.invokeMethod("service.remotingMethod", args, callback);

1. RED5库
2. Spring脚本支持

spring-support.jar

1. Groovy

asm-2.2.2.jar  
antlr-2.7.6.jar  
groovy-1.0.jar

1. Beanshell

bsh-2.0b5.jar  
cglib-nodep-2.1\_3.jar

1. Ruby

jruby.jar  
cglib-nodep-2.1\_3.jar

1. Jython / Python

jython.jar

1. Java 5 Libraries

Java5 需要以下包：  
jsr-223-1.0-pr.jar  
jsr173\_1.0\_api.jar

1. Javascript / Rhino

js.jar  
xbean.jar - needed for E4X

1. 编译RED5
2. 构建编译环境

构建编译环境

Ant

编译red5，需要Ant 1.7或以上版本，ant下载地址：

<http://archive.apache.org/dist/ant/binaries/>，需要把ant添加到环境变量中

Java

需要java1.5，1.6或以上版本，才能编译运行

Java5下载地址：<http://java.sun.com/j2se/1.5.0/download.html>

Java6下载地址：<http://java.sun.com/j2se/1.6.0/download.html>

Java需要设置 java\_home

Red5 Source

SVN checkout

<http://red5.googlecode.com/svn/java/server/trunk/>

Red5 Demos

<http://red5.googlecode.com/svn/java/example/trunk/>

Red5 Flash Demo

<http://red5.googlecode.com/svn/flash/trunk/>

执行ant编译

Ant dist

1. 在eclipse中如何编译
2. Eclipse插件

需要以下eclipse插件才能编译red5

IvyIDE，下载地址：<http://ant.apache.org/ivy/ivyde/download.cgi>.

Spring IDE: 下载地址：<http://springide.org/project/wiki>

Subclipse SVN Plugin： 下载地址：<http://subclipse.tigris.org/>

1. 导入eclipse工程

有两种方式导入eclipse工程，从本地文件导入或者是从svn导入

1. Ant，IVY

ant ivyclear

1. 发行RED5
2. RED5系统要求

|  |  |
| --- | --- |
| Supported operating systems | Windows 2000 Server  Windows 2003 Server, Standard Edition  Linux Variants  Mac OSX 10.4 and above |
| Minimum Hardware Requirements  (Development / Budget / Low Traffic) | X86-compatible CPU (Pentium 4, 3.2 GHz or better, Intel Duo Core 2, PentiumD)  1 GB Available Memory  100MB or 1GB Ethernet card  200 MB of available disk space (SATA II) |
| Recommended Hardware Requirements  (High Traffic Production) | Dual-core / Quad Core (Intel XEON 2Ghz and above, Opteron 2Ghz and above)  2 - 4 GB Available memory or above  1GB Ethernet Card with big pipe network  200 MB of available disk space (10K RPM and above SATA II \ SCSI RAID 1-5)  Network Storage Cluster Solution for mass content storage (ie Isolon, Dell MD1000) |
| Software Requirements | Java JRE 1.5 or 1.6  Service Scripts (Java Service Wrapper, FireDaemon Pro) |

1. Red5核心技术
2. 创建新应用程序
3. 将Red5部署到tomcat中

标准的Red5发布版本包含一个标注的Java应用和一个嵌入的J2EE容器（jetty或Tomcat），能做为一个系统服务运行，而War版本的red5也可以在j2ee容器里运行

部署

Tomcat war包部署器会周期性的扫描webapp目录的war包，当发现有war没有被部署时，会根据war名字解压到一个相对应目录。

Red5服务器被打包到一个叫Root的war里，在大多数J2EE应用程序里，root这个名字有着特殊的含义，通常是默认上下文或者是网站根上下文，网站根上下文可以响应路径中没有参数的请求，例如：一个<http://www.example.com/myapp>的应用的网站跟上下文为<http://www.example.com/>，上下文描述文件放在META-INF