2010 年 03 月 02 日 星期二 上午 10:34

开发你自己的 XMPP IM - [J2EE]

最近没在 Blog 上露脸，为撒类？应师兄的请求，帮他研究一个 XMPP IM 软件的开发。反正最近也没什么大事，每天都想写写代码练练手，就帮忙呗。研究了一通觉得还挺有趣，自己这几天查国内外的资料，发现国内关于这方面间的软件资料太少了，就想在这里写几篇关于此类 IM 软件开发的文章。不过别看东西小，涉及的模块可不少。

所以我基本上分为三篇文章来介绍此类软件的开发：

第一篇是关于 XMPP 协议是啥，IM 是啥以及一个比较有名的开源实现，该开源实现包括三个部分（Spark、Smack 和 Openfire）；

第二篇讲如何开发基于 Spark 的客户端 IM 插件部分；

第三篇讲如何开发基于 Openfire 服务器端的插件部分。

好了，进入正题吧。

什么是 XMPP?

Extensible Messaging and Presence Protocol，简单的来讲，它就是一个发送接收处理消息的协议，但是这个协议发送的消息，既不是二进制的东东也不是字符串，而是 XML。正是因为使用了 XML 作为消息传递的中介，Extensible 才谈的上，不是么？嘿嘿。再详尽的东西，我也就不多介绍了，大家可以去百度百科里查看下。

什么是 IM ？

Instant Messenger，及时通信软件，就是大家使用的 QQ、MSN Messenger 和 Gtalk 等等。其中 Gtalk 就是基于 XMPP 协议的一个实现，其他的则不是。当前 IM 几乎作为每个上网者必然使用的工具，在国外的大型企业中有一些企业级的 IM 应用，但是其商业价值还没完全发挥出来。设想既然 XMPP 协议是一个公开的协议，那么每个企业都可以利用它来开发适合本身企业工作，提高自身生产效率的 IM；甚至，你还可以在网络游戏中集成这种通信软件，不但让你可以边游戏边聊天，也可以开发出适合游戏本身的 IM 应用，比如说一些游戏关键场景提醒功能，团队语音交流等等都可以基于 IM 来实现。说了这么多，就是一个意思，其商业价值远远比你想的高！

Spark Smack 和 Openfire

开源界总是有许多有趣的东东，这三个合起来就是一个完整的 XMPP IM 实现。包括服务器端——Openfire，客户端——Spark，XMPP 传输协议的实现——Smack（记住，XMPP 是一个协议，协议是需要实现的，Smack 起到的就是这样的一个作用）。三者都是基于 Java 语言的实现，因此对于熟悉 Java 的开发者来说不是很难

Spark 提供了客户端一个基本的实现，并提出了一个很好的插件架构，这对于开发者来说不能不说是一个福音。我强烈建议基于插件方式来实现你新增加的功能，而不是去改它的源代码，这样有利于你项目架构，把原始项目的影响降到最低，文章以后的部分也是基于这种插件体系进行开

发的

Openfire 是基于 XMPP 协议的 IM 的服务器端的一个实现，虽然当两个用户连接后，可以通过点对点的方式来发送消息，但是用户还是需要连接到服务器来获取一些连接信息和通信信息的，所以服务器端是必须要实现的。Openfire 也提供了一些基本功能，但真的很基本的！庆幸的是，它也提供插件的扩展，像 Spark 一样，我同样强烈建议使用插件扩展的方式来增加新的功能，而不是修改人家的源代码。

Smack 是一个 XMPP 协议的 Java 实现，提供一套可扩展的 API，不过有些时候，你还是不得不使用自己定制发送的 XML 文件内容的方式来实现自己的功能

下图展示了三者之间的关系：

从图上可以了解到，client 端和 server 端都可以通过插件的方式来进行扩展，smack 是二者传递数据的媒介。

开发你自己的 XMPP IM 续 - Spark 插件开发 - [J2EE]
继续 3 月 18 日介绍基于 XMPP IM 开发的那篇 Blog，今天主要总结一下如何基于 Spark 的插件架构来新增客户端的功能，这里列举出一个获取服务器端群组信息的实际例子，实现后的效果如下图所示：

Spark 是一个基于 XMPP 协议，用 Java 实现的 IM 客户端。它提供了一些 API，可以采用插件机制进行扩展，上图中，“部门”部分就是使用插件机制扩展出来的新功能。要想实现你的扩展，首先要了解 Spark API 的架构，其中最关键的是要了解它的工厂类，这些工厂类可以获得 Spark 提供的诸如 XMPPConnection、ChatContainer 等实例，从而你可以实现获取服务器的信息，与另外的 Client 通信等功能。最核心的类是 SparkManager，这个类是一系列工厂类的工厂类（呵呵，还真拗口）。它的 getChatManager()、getSessionManager ()、getMainWindow() 、getConnection() 等方法分别可以获得聊天管理器、会话管理器、主窗口、与服务器的连接等等非常有用的实例。基本上可以说 SparkManager 是你与 Spark 打交道的衔接口。其实，每一个 Manager 都使用了单例模式，你也可以不通过 SparkManager 来获取它们，但笔者建议你从单一的入口着手，这样有利于代码的开发和维护。

接下来描述一下插件的开发流程：
1、创建插件配置文件 plugin.xml
2、实现你自己的 Plugin 类的实现（如果你需要实现自己规定格式的 XML 发送、接收和处理，那么你需要在这里注册你的 IQProvider，关于 IQProvider 你可以查询 Smack API，简单的来讲是处理你自定义的 IQ 处理器。）
3、打包你的插件（Spark 有自己的打包机制，我研究了半天才发现其中的玄机，后面介绍）
4、部署你的插件（其实 3、4 两步可以糅合在一起，当然要利用 Ant 啦）

好滴，下面结合一个实际的例子讲述上面的四个步骤
1、plugin.xml

```
<plugin>
    <name>Enterprise IM Client</name>
```

```
    <version>1.0</version>
    <author>Phoenix</author>
    <homePage>http://phoenixtoday.blogbus.com</homePage>
    <email>phoenixtoday@gmail.com</email>
    <description>Enterprise Client Plug-in</description>
    <!-- 关键是这里，这里要定义你的 Plugin 类 -->
    <class>com.im.plugin.IMPlugin</class>
    <!-- 这里定义你使用的 Spark 最低版本 -->
    <minSparkVersion>2.5.0</minSparkVersion>
    <os>Windows</os>
</plugin>
```

这是一个 plugin.xml 文件的内容，插件体系会自动调用你在此文件中定义的 Plugin 类，从而完成你自己扩展的功能。最关键的部分我用红色标识出来了，要声明你的插件扩展类，采用完整的命名空间方式（包括包名），其余的部分结合我的注释，大家应该都能理解，就不做详细的描述了。要注意的是 plugin.xml 文件要放在项目的根目录下，这是严格规定好的。

2、Plugin 类的实现

你的类首先要实现 Spark 提供的 Plugin 接口，然后实现它的一些方法。其中最主要的是实现 initialize() 发放，在这里注册你的的 IQProvider

```
ProviderManager providerManager = ProviderManager.getInstance();
providerManager.addIQProvider("groups", "com:im:group", //1
                new GroupTreeIQProvider());
System.out.println("注册 GroupTree IQ 提供者");
requestGroupTree();
```

上述的代码，就在该类就是我实现的 IMPlugin.initialize() 方法中的一小段，大概的含义是，先获取 ProviderManager（这个貌似不能从 SparkManager 直接获取），然后注册一个 GroupTreeIQProvider（自己创建的）这是一个 IQProvider 的具体实现，它用于解析像下面这样的一个 XML 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<iq type='result' to='domain@server.com' from='phoenixtoday@gmail.com'
id='request_1'>
    <groups xmlns='com:im:group'>
        <group>
            <groupId>1</groupId>
            <name>西安交通大学</name>
            <upGroup>ROOT</upGroup>
            <isLeaf>0</isLeaf>
            <description>xjtu</description>
            <user>
                <userGroupId>1</userGroupId>
```

```
            <userName>phoenix_test</userName>
            <role>normal</role>
        </user>
    </group>
    <group>
        <groupId>2</groupId>
        <name>电信学院</name>
        <upGroup>1</upGroup>
        <isLeaf>1</isLeaf>
        <description>xjtu info</description>
    </group>
</groups>
</iq>
```

可以看到，在注册 IQProvider 的时候（代码中标注的 1 部分），需要你提供名称和命名空间，我的 XML 文件中的 iq 下的第一个子节点是<groups> 所以我的名称就写“groups”，命名空间对应于 groups 节点的 xmlns(XML Name Space)所以是“com:im:group”，其实 IQProvider 中最关键的方法是 parseIQ(XmlPullParser parser) 该方法就是解析 XML，完成你的功能，并返回一个相应的 IQ 实例（这里可以把 IQ 看做一个回馈的 Model 类）。说到底实现基于 XMPP 协议的 IM 就是解析 XML 文件，而这正是客户端的 IQProvider 和服务器端的 IQHandler（下一篇文章会涉及到）所做的事情。

3、打包你的插件

现在该有的功能都实现了，那么就是打包了。这最好利用 Ant 来完成，因为每次你都要打包，要部署，如果纯手动的话，那也太不敏捷了，大大影响开发效率。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="IM" default="release" basedir=".">
    <property name="src.dir" value="src" />
    <property name="dest.dir" value="bin" />
    <property name="lib.dir" value="lib" />
    <property name="im.path"
        value="E:/workspace/europa/spark_new/doc/spark/target/build" />
    <target name="clean">
        <!--
            <delete dir="${dest.dir}" />

            <delete dir="${lib.dir}" />
        -->
    </target>
    <target name="init" depends="clean">
        <!--
            <mkdir dir="${dest.dir}" />
```

```
            <mkdir dir="${lib.dir}" />
        -->
    </target>
    <target name="build" depends="init">
        <!--
            <javac srcdir="${src.dir}" destdir="${dest.dir}" />
        -->
    </target>
    <!-- 最重要的是这里，打两次包 -->
    <target name="jar" depends="build">
        <jar jarfile="${lib.dir}/eim.jar" basedir="${dest.dir}" />
        <jar jarfile="${im.path}/plugins/eim.jar">
            <fileset dir=".">
                <include name="lib/*.jar" />
            </fileset>
            <fileset dir=".">
                <include name="plugin.xml" />
            </fileset>
        </jar>
    </target>
    <target name="release" depends="jar">
        <!--
            <exec executable="cmd.exe"
            failonerror="true">
            <arg line="/c e:"/>
            <arg line="/c cd
workspace\europa\spark_new\doc\spark\target\build\bin"/>
            <arg line="/c startup.bat"/>
            </exec>
        -->
    </target>
</project>
```

这是我的这个项目的 build.xml 文件中的内容。因为 Eclipse 其实帮我自动完成了编译的任务，所以我也就省去了这写编译的步骤，最重要的是大家要看到“jar” 部分，Spark 打包的神秘之处也就在此，打两次包首先把你的项目打包到本项目 lib 文件夹下，比如说你的项目目录是 MyPlugin 那么，你就将你的类打包到 MyPlugin/lib 目录下，然后再次的打包，将所有的 lib 文件夹下的内容打包起来，记得这次要包含 plugin.xml。也就是说，最后 Spark 插件体系会读取你的项目下的 lib 文件夹下的内容。这里我也有个疑问，我本来想每次打包后自动执行 bat 文件，启动插件，看看效果，为啥死都调用不了呢，那段代码在最后面，注释掉了，谁能帮我解决，我请他吃饭滴！

4、最后就是发布了

其实我的发布很简单，就是将这个打包好的 jar 文件拷到 Spark 本身的 plugins 目录下，每次启动 Spark 的时候，它会自动调用自定义的插件的。我这里用 Ant 第二次 jar 的时候，就自动拷贝过去了，这里用的是绝对路径，所以你不能直接拷贝就用滴呦（是不是很丑陋呀，这段 Ant 代码）。

基本上客户端的实现原理就是这样的，只是有些地方需要特别注意，还有就是应该利用像 Ant 这样的工具大大简化开发步骤，加快开发效率。还有就是，我建议你在开发自己的插件的时候，多利用 MVC 模式，尤其是在 IQProvider 解析后，生成的部分可以实例化 Model，然后你可以编写自己的 Manager 进行这些 Model 的处理。多写 Log，当然 Log4j 貌似不太起作用，那就 System.out.println() 吧，哈哈！今天就写到这里啦，偶有点累啦。

开发你自己的 XMPP IM 续 - Openfire 插件开发 - [J2EE]
继续上一篇的内容，本篇文章介绍开发 Openfire 的插件

这篇文章拖了很久了，呵呵，真是千呼万唤始出来呀。Openfire 服务器端是支持插件开发的，开发过程可能会涉及到数据库的操作，本篇文章专注于 Openfire 插件的部分，对服务器端涉及到数据库的开发只做简单介绍。

Openfire 是一个用 Java 实现的 XMPP 服务器，客户端可以通过 IQ 的方式与其进行通信（其实就是 XML），客户端和服务器之间的通信是依靠底层 Smack 库提供的各种功能来完成的。其实利用插件方式来扩展 Openfire 服务器端主要有两种扩展方式，一种是对服务器控制台页面进行扩展（不是本文的主要内容），其实就是遵循 Openfire 页面的布局方式，进行相应的页面扩展和功能扩展；另一种是对通信功能进行扩展。本文主要针对后者进行具体的描述

本篇文章的结构如下：

1、创建 plugin.xml（这是整个插件最关键的文档）
2、创建服务器插件实例（实现 Plugin 接口的一个类还有一批 IQHandler）
3、打包插件（Openfire 插件也有自己的打包方式）和部署插件

好滴，实刀实枪的来动手做吧

1、创建 plugin.xml

初次开发 Openfire 和 Spark 插件的时候，很容易把二者搞混，千万记得，这里是 Openfire 的 plugin.xml 不是第二篇文章说的那个啦！

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
    <!-- Main plugin class 这里是最重要滴-->
    <class>com.im.server.plugin.GroupTreePlugin</class>

    <!-- Plugin meta-data -->
    <name>GroupTreePlugin</name>
    <description>This is the group plugin.</description>
```

```
    <author>Phoenix</author>

    <version>1.0</version>
    <date>14/03/2008</date>
    <url>http://localhost:9001/openfire/plugins.jsp</url>
    <minServerVersion>3.4.1</minServerVersion>
    <licenseType>gpl</licenseType>

    <!-- Admin console entries -->
    <adminconsole>
        <!-- More on this below -->
    </adminconsole>
</plugin>
```

最重要的那一行我已经标记出来啦，就是你这个插件的初始化和垃圾清理类，例子中是在 com.im.server.plugin 包中的 GroupTreePlugin 类，下文会对这个类进行详细描述。其余的都是描述信息，只要你提供了正确的描述信息，一般都不会出错。建议初次开发者，在写完 plugin.xml 文件后，写一个简单的 Plugin 实例，并打印出一些信息，如果重新启动 Openfire 信息成功显示，恭喜你，你已经迈出一大步了！

2、实现 Plugin 类和 IQHandler

Plugin 类主要起到的作用是初始化和释放资源，在初始化的过程中，最重要的的注册一批 IQHandler，IQHander 的作用有点类似于 Spark 中的 IQProvider，其实就是解析 XML 文件之后，生成一些有用的实例，以供处理。下面分别给出一个 Plugin 类的实例和 IQProvider 的实例

GroupTreePlugin 类

```
/**
* 服务器端插件类
*
* @author Phoenix
*
* Mar 14, 2008 11:03:11 AM
*
* version 0.1
*/
public class GroupTreePlugin implements Plugin
{
    private XMPPServer server;

    /*
     * (non-Javadoc)
     *
     * @see org.jivesoftware.openfire.container.Plugin#destroyPlugin()
```

```
     */
    public void destroyPlugin()
    {

    }

    /*
     * (non-Javadoc)
     *
     * @see
org.jivesoftware.openfire.container.Plugin#initializePlugin(org.jivesoftware.openfi
re.container.PluginManager,
     *      java.io.File)
     */
    public void initializePlugin(PluginManager manager, File pluginDirectory)
    {
        PluginLog.trace("注册群组树 IQ 处理器");
        server = XMPPServer.getInstance();

        server.getIQRouter().addHandler(new GroupTreeIQHander()); //1
        server.getIQRouter().addHandler(new UserInfoIQHandler());
        server.getIQRouter().addHandler(new DelUserIQHandler());
        server.getIQRouter().addHandler(new CreateUserIQHandler());
        server.getIQRouter().addHandler(new AddGroupUserIQHandler());
        server.getIQRouter().addHandler(new SetRoleIQHandler());

    }

}
```

上例所示,在初始化中先找到 IQRouter,然后通过 IQRouter 注册一批 IQHandler,这些 IQHander 会自动监听相应命名空间的 IQ，然后进行处理；由于这个 Plugin 不需要做资源释放的工作，所以在 destroyPlugin() 方法中没有任何内容。具体的 IQHander 类如下

GroupTreeIQHander

```
/**
* 处理客户端发来的 IQ，并回送结果 IQ
*
* @author Phoenix
*
* Mar 14, 2008 4:55:33 PM
*
* version 0.1
*/
```

```
public class GroupTreeIQHander extends IQHandler
{

    private static final String MODULE_NAME = "group tree handler";

    private static final String NAME_SPACE = "com:im:group";

    private IQHandlerInfo info;

    public GroupTreeIQHander()
    {
        super(MODULE_NAME);
        info = new IQHandlerInfo("gruops", NAME_SPACE);
    }

    /*
     * (non-Javadoc)
     *
     * @see org.jivesoftware.openfire.handler.IQHandler#getInfo()
     */
    @Override
    public IQHandlerInfo getInfo()
    {
        return info;
    }

    /*
     * (non-Javadoc)
     *
     *@see org.jivesoftware.openfire.handler.IQHandler#handleIQ(org.xmpp.packet.IQ)
     */
    @Override
    public IQ handleIQ(IQ packet) throws UnauthorizedException
    {
        IQ reply = IQ.createResultIQ(packet);
        Element groups = packet.getChildElement();//1

        if (!IQ.Type.get.equals(packet.getType()))
        {
            System.out.println("非法的请求类型");
            reply.setChildElement(groups.createCopy());
            reply.setError(PacketError.Condition.bad_request);
            return reply;
        }
```

```
        String userName =
StringUtils.substringBefore(packet.getFrom().toString(),"@");

        GroupManager.getInstance().initElement(groups,userName);

        reply.setChildElement(groups.createCopy());//2

        System.out.println("返回的最终 XML" + reply.toXML());

        return reply;
    }

}
```

可以看到主要有两个方法，一个是 getInfo() 这个方法的目的是提供要解析的命名空间，在本例中，这个 IQHandler 对每个命名空间为"com:im:group" 的实例进行处理；还有一个最重要的方法：handleIQ() 该方法对包含指定命名空间的 XML 进行解析，然后返回一个解析好的 IQ。其实我认为，这个 IQHandler 和 IQ 的关系就是 Controller 和 Model 的关系（如果你了解 MVC 的话，那么你一定知道我再说什么），只不过这里并没有指定什么 View，你完全可以把 IQ 当成 Model 类进行理解。在这里，我用了 GroupManager 进行了 XML 的处理，因为我返回的 IQ 内容中要从数据库读取所有群组信息，所以转交给 GroupManager 进行处理，你完全可以在这个方法中进行具体的 XML 处理，在这里，解析和创建新的 XML 主要用到的是 JDOM（如果你对 Java 解析 XML 有所了解，那真的太好了！）。程序//1 处主要是获取创建返回的 IQ，并获取原来 IQ 的子元素（用于创建我们返回的 IQ）；程序//2 处很关键，如果你不调用 createCopy 方法，程序会出错（程序会死锁还是什么，忘记咧，不好以西）。

这就是程序的主体部分，我在这里有一个建议，能不用 Openfire 原始的程序函数，就不要用它们。我的提取数据库方式都是自己写的 Bean，这样有利于你自己对程序的掌控，其实更有利于快速开发（这世道不是啥都讲究敏捷么，哇哈哈）

3、打包插件

打包依然遵循二次打包的原则（如果你不了解啥叫要二次打包，请看上一篇）
这是我的 ant 文件，由于 Eclipse 帮我做了 build 等很多工作，实际我的 ant 工作就是在打包，并放入插件目录下的 plugin 文件夹下

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="IM" default="release" basedir=".">

    <property name="openfire.path"
        value="E:/workspace/europa/openfire_src/target/openfire" />
    <property name="classes.dir" value="classes" />
    <property name="lib.dir" value="lib" />

    <target name="jar">
```

```
        <jar jarfile="${lib.dir}/grouptreeplugin.jar" basedir="${classes.dir}" >
            <fileset dir=".">
                <include name="*.jar"/>
            </fileset>
        </jar>
        <jar jarfile="${openfire.path}/plugins/groupTreePlugin.jar">
            <fileset dir=".">
                <include name="lib/*.jar" />
                <include name="plugin.xml" />
                <include name="logo_small.gif" />
                <include name="logo_large.gif" />
                <include name="readme.html" />
                <include name="changelog.html" />
                <include name="build.xml" />
            </fileset>
        </jar>

    </target>

    <target name="release" depends="jar">
    </target>

</project>
```

好了，至此 XMPP+Spark+Openfire 的插件开发三部曲彻底结束了，希望你们对这个开发流程有了系统的了解。