第一周实习报告

5080309684 仇卓

目标概述

在同一局域网内的三台 Plug computer 上开发 OSGi 应用，实现在一台 Plug computer 上远程访问另一 Plug computer 上的 Bundle 的功能。

平台介绍

Plug computer

Plug computer 是一种小型嵌入式设备，具有跟传统的个人电脑相同的功能（但一般没有显示卡），与传统个人电脑相比具有低成本、低能耗、占用空间小等优势。Plug computer 插上电源即可自动启动，很适合作为家庭影音服务器、数据备份服务器和用于远程访问一些服务等。

OSGi

OSGi（Open Service Gateway Initiative）一方面它指 OSGi Alliance 组织，另一方面指该组织制定的一个基于 Java 语言的服务（业务）规范——OSGi 服务平台（Service Platform）。OSGi 的标准规定了所有的应用（一个应用被称为一个 Bundle，一个 jar 包）都要在一个实现了 OSGi 接口的 Framework 上运行，每个 Bundle 可能会导入一些来自别的 Bundle 的包，还会导出一些别的 Bundle 需要的包。每个 Bundle 被安装到 Framework 上的时候会自动解析这个 Bundle 需要以及导出的包，Bundle 的升级不需要重启整个 Framework。

一个简单的 Server-Client 例子

Server 中的服务接口：

```
package testserver.service;

public interface HelloService {
    public String hello();
}
```

Server 中服务的实现：

```
package testservet.service.impl;
import testserver.service;

public class HelloServiceImpl implements HelloService {
    public String hello() {
        System.out.println("Hello from HelloServiceImpl");
        return "Hello";
    }
}
```

Server 的 Avtivator:

```
public class HelloServiceActivator implements BundleActivator {
    ServiceRegistration helloServiceRegistration;
    public void start(BundleContext context) throws Exception {
        HelloService helloService = new HelloServiceImpl();
        helloServiceRegistration = context.registerService(
                HelloService.class.getName(), helloService, null);
    }
    public void stop(BundleContext context)throws Exception {
        helloServiceRegistration.unregister();
    }
}
```

Server 的 MANIFEST.MF 需要添加:

```
Export-Package: testserver.service
```

Client 的 Avtivator:

```
Public class Activator implements BundleActivator {
    ServiceReference helloServiceReference;
    public void start(BundleContext context)throws Exception {
        System.out.println("HelloWorld!!");
        helloServiceReference = context.getServiceReference(HelloService.class.getName());
        HelloService helloService=(HelloService)context.getService(helloServiceReference);
        System.out.println(helloService.sayHello());
    }
    public void stop(BundleContext context)throws Exception {
        System.out.println("Goodbye World!!");
        context.ungetService(helloServiceReference);
    }
}
```

Client 的 MAINIFEST.MF 需要添加:

```
Import-Package: com.javaworld.sample.service,
```

实现要点

R-OSGi

任务要求实现多台设备的Bundle间的互相访问，而OSGi标准仅仅规定了单一Framework上的接口标准，并没有提供分布式访问的接口，因此使用 R-OSGi（Remote-OSGi）来实现分布式 OSGi。要使用 R-OSGi 需要先在 Framework 中安装一个 R-OSGi 的 Bundle，server 端通过调用它的注册服务接口来把服务注册在网络上，client 端通过调用它的查询服务接口来获取网络上其他 Bundle 已经注册的服务。client 端只需知道 server 端的 IP 地址即可获取 server 端已经注册的服务，但是真实的应用场景中 Plug computer 可能是接入一个使用 DHCP 的局域网中，很可能每次启动时 IP 地址跟上一次的不同，这样就需要首先得到 server 端的 IP 地址，然后才能获得服务。

SLP

需要查找 server 端的 IP 地址，可以使用 SLP（Service Location Protocol 服务定位协议）。在实验中使用了 SLP 的 java 实现 jSLP，同样是在 Framework 中安装一个 jSLP 的 Bundle。server 端通过 SLP 的注册接口将自己的 IP 地址等信息注册在网络上，client 调用 SLP 的查询接口发送一个多播/广播请求查询到 server 端的信息。client 端得到了 server 端的信息后，通过 server 端的 IP 地址使用 R-OSGi 就可以使用 server 端提供的服务。

示例：

中间服务接口:

```
package testservice;

public interface HelloService {
    public String hello();
}
```

需要导出 HelloService:

```
Export-Package: testservice
```

Server 端：

服务实现:

```
package testserviceimpl;
import testservice.HelloService;

public class HelloServiceImpl implements HelloService {
    public String hello() {
        System.out.println("Hello from HelloServiceImpl");
        return "Hello";
    }
}
```

Server 端 Activator:

```
public class Activator implements BundleActivator {
    ServiceRegistration registration;
    Advertiser advertiser;
    ServiceReference advRef;
    Hashtable properties = new Hashtable();
    String IPAddr = new String();

    public void start(BundleContext context) throws Exception {
        advRef = context.getServiceReference(Advertiser.class.getName());
        properties.put(RemoteOSGiService.R_OSGi_REGISTRATION, Boolean.TRUE);
        registration = context.registerService(HelloService.class.getName(),
                new HelloServiceImpl(), properties);

        getIPAddresses();
        if (IPAddr.equals("")) {
```

```
                System.out.println("Invalid IP address");
                return;
            }

            if (advRef == null) {
                System.out.println("advRef is null!");
            } else   {
                System.out.println("Got reference for Advertiser");
                advertiser = (Advertiser) context.getService(advRef);
                advertiser.register(
                        new ServiceURL("service:osgi:r-osgi://"  + IPAddr + ":9278", 50),
                        null);
                System.out.println("registered: " + "service:osgi:r-osgi://"  + IPAddr + ":9278");
            }

        }

        public void stop(BundleContext context) throws Exception {
            registration.unregister();
            advertiser.deregister(new ServiceURL(
                    "service:osgi:r-osgi://"  + IPAddr + ":9278", 1000));
            System.out.println("unregistered");
        }

        public void getIPAddresses() throws SocketException {    //获取本机 IP 地址
            Enumeration e = NetworkInterface.getNetworkInterfaces();
            while (e.hasMoreElements()) {
                NetworkInterface ni = (NetworkInterface) e.nextElement();
                Enumeration ia = ni.getInetAddresses();
                while (ia.hasMoreElements()) {
                    InetAddress addr = (InetAddress) ia.nextElement();
                    if (addr.isLoopbackAddress())  {
                        continue;
                    }
                    if (addr instanceof Inet4Address) {
                        IPAddr = addr.getHostAddress();
                    }
                }
            }
        }
    }
```

MANIFEST.MF 中需要导入中间服务接口、R-OSGi、jSLP：

```
Import-Package: ch.ethz.iks.r_osgi,
 ch.ethz.iks.slp,
```

```
 org.osgi.framework;version="1.3.0",
 testservice
```

Client 端：

Activator:

```
public class Activator implements BundleActivator {
	Vector ServiceLocation = new Vector();
	RemoteOSGiService remote;
	RemoteServiceReference[] refs;
	ServiceReference sref;
	HelloService helloservice;
	ServiceReference locRef;

	public void start(BundleContext context) throws Exception {
		locRef = context.getServiceReference(Locator.class.getName());
		if (locRef == null) {
			System.out.println("locRef is null!");
		} else {
			System.out.println("Got reference for Locator");
			Locator locator = (Locator) context.getService(locRef);

			ServiceLocationEnumeration slenum = locator.findServices(
					new ServiceType("service:osgi"), null, null);
			System.out.println("RESULT:");
			while (slenum.hasMoreElements()) {
				ServiceURL service = (ServiceURL) slenum.nextElement();
				String url = service.toString();
				String address = url.substring(13);
				System.out.println("address: " + address);
				ServiceLocation.add(address);
			}
		}

		sref = context.getServiceReference(RemoteOSGiService.class.getName());
		if (sref == null) {
			throw new BundleException("OSGi remote service not found");
		}

		remote = (RemoteOSGiService) context.getService(sref);
		for (int i = 0; i < ServiceLocation.size(); i++) {
			refs = remote.getRemoteServiceReferences(
					new URI(ServiceLocation.elementAt(i).toString()),
					HelloService.class.getName(), null);
```

```
                if (refs == null) {
                    System.out.println("Service not found at "
                            + ServiceLocation.elementAt(i).toString());
                    continue;
                }

                helloservice = (HelloService) remote.getRemoteService(refs[0]);
                System.out.println(helloservice.hello());
            }

        }

        public void stop(BundleContext context) throws Exception {
            System.out.println("Client exit");
        }
}
```

MANIFEST.MF 中需要导入中间服务接口、R-OSGi、jSLP：

```
Import-Package: ch.ethz.iks.r_osgi,
 ch.ethz.iks.slp,
 org.osgi.framework;version="1.3.0",
 testservice
```

开发环境和工具

开发环境

实验的开发环境是在 Linux 系统中使用 Eclipse 进行开发。需要注意的是，因为要用到 R-OSGi 和 jSLP，需要分别从他们的官方网站上下载实现的 jar 文件，放入 PC 的 /usr/lib/eclipse/plugins 目录中，必要时还需要将权限改为 644（文件拥有者可读写，其他用户只读）。需要的 jar 文件放入 plugins 目录后，重启 Eclipse，新建 Plug-in project，这时在 MANIFEST.MF 的 dependency 标签中就可以导入 ch.ethz.iks.r_osgi（R-OSGi）和 ch.ethz.iks.slp（jSLP）两个包了。要将开发的 Bundle 安装到 Plug computer 上时，首先需要导出 Bundle，在 project 名字上右击，点击 export，选择路径导出为 jar 文件。

部署工具

Plug computer 需要一个 OSGi 的 framework，这个由 prosyst 公司的 OSGi SDK 提供。给 Plug computer 插上网线以便远程管理，使用 ssh 登录到 Plug computer，通过 u 盘将 prosyst 公司提供的 osgi_kit 放入 Plug computer，运行 osgi_kit/osgi-hgw/bin/vms/j9/server，这时 framework 启动，同时会在本地开一个端口可以使用 web 登录到这台 Plug computer 进行管理。假设 Plug computer 的 IP 是 192.168.0.219，在同一局域网内的 pc 上，用浏览器访问 192.168.0.219/system/console，用户名密码默认都是 admin，这样就进入了 web 管理界面。现在我们可以安装 pc 上的 Bundle 到 Plug computer，很简单，只需点击 web 管理界面上的 install/update 即可，我们不需关心如何传输文件。安装后即可在终端或 web 界面管理 Bundle。

注意事项和问题

因为需要远程访问服务，所以至少要 2 个 Bundle，一个是 server 端提供服务，一个是 client 端使用服务，而 client 需要使用 server 的服务就需要导入 server 导出的包。在一台机器上运行时，server 到处服务，client 使用服务，现在 server 和 client 在两台机器上，client 无法直接导入 server 的服务，因为 server 是在另一台机器上导出服务的。这时就需要一个中间接口，server 和 client 端的机器上都要安装这个中间接口的 Bundle，server 端导入这个接口并将其实现，然后将实现的服务注册到网络上，client 端导入这个接口，从网络上获得 server 的服务，通过中间接口就可以使用 server 提供的服务。

jSLP 默认使用 427 端口进行通信，然而在 Linux 平台上必须是超级用户（root）才能使用 427 端口进行通信。在 PC 上测试时，需要使用 root 身份运行 Eclipse，在 Plug computer 中，也需要使用 root 身份启动 framework。

在刚开始使用 jSLP 时，在 PC 上会报错：bad argument for IP_MULTICAST_IF: address not bound to any interface。通过在网上查询得知，我在 Elicpse 中新建 project 时运行环境选择的是 OSGi/Minimum-1.0，这个运行环境应该是不支持 IPv6 的通信，解决方法是禁用操作系统的 IPv6 功能。

后来的测试中，出现了一个奇怪的问题， Plug computer 中可以发现 PC 中在 SLP 注册的服务，而 PC 中却找不到 Plug computer 上注册的服务。而 Plug computer 之间则可以发现对方的服务。这个问题目前还没有找到原因。

参考资料

http://www.osgi.org/ OSGi Alliance
http://r-osgi.sourceforge.net/ R-OSGi
http://jslp.sourceforge.net/ jSLP
http://www.prosyst.com/ Prosyst