



RCloud

产品用户开发手册

（RCloud 云 Enterprise Service Bus 服务）

北京中软国际信息技术有限公司

2010 年

目 录

第一章 概述.....	2
1.1 面向读者.....	2
1.2 开发准备.....	2
第二章 快速入门.....	2
2.1 消息流定义示例一【统一请假服务】	2
2.1.1 请假服务业务描述.....	2
2.1.2 请假流程服务注册.....	2
2.1.3 请假服务消息流定义流程.....	3
2.1.4 请假服务代码示例.....	7
2.2 消息流定义示例二【天气预报】	8
2.2.1 天气预报业务描述.....	8
2.2.2 天气预报服务注册.....	8
2.2.3 天气预报消息流定义流程.....	8
2.2.4 天气预报代码示例.....	13
第三章 开发指导.....	16
3.1 服务封装设计.....	16
3.1.1 代理服务接口定义.....	16
3.1.2 代理服务调用方式.....	16
3.1.3 代理服务返回值	17
3.2 服务调用.....	17
第四章 开发建议.....	17
4.1 日志系统的使用.....	17

第一章 概述

1.1 面向读者

本手册详细描述了 RCloud 云 EnterpriseServiceBus 服务（以下简称 RCloud 云 ESB 服务）的二次应用开发过程及核心 API 的使用方法，主要面向基于该产品的二次开发人员、实施部署人员。

1.2 开发准备

在开始进行二次应用开发之前，需要按照《RCloud 云 EnterpriseServiceBus 服务产品用户使用手册》说明，在 RCloud 网站中注册用户，并购买 RCloud 云 ESB 服务。

第二章 快速入门

本节以典型的业务应用过程为例，演示基于 RCloud 云 ESB 服务的二次开发步骤与功能实现方式：

2.1 消息流定义示例一【统一请假服务】

2.1.1 请假服务业务描述

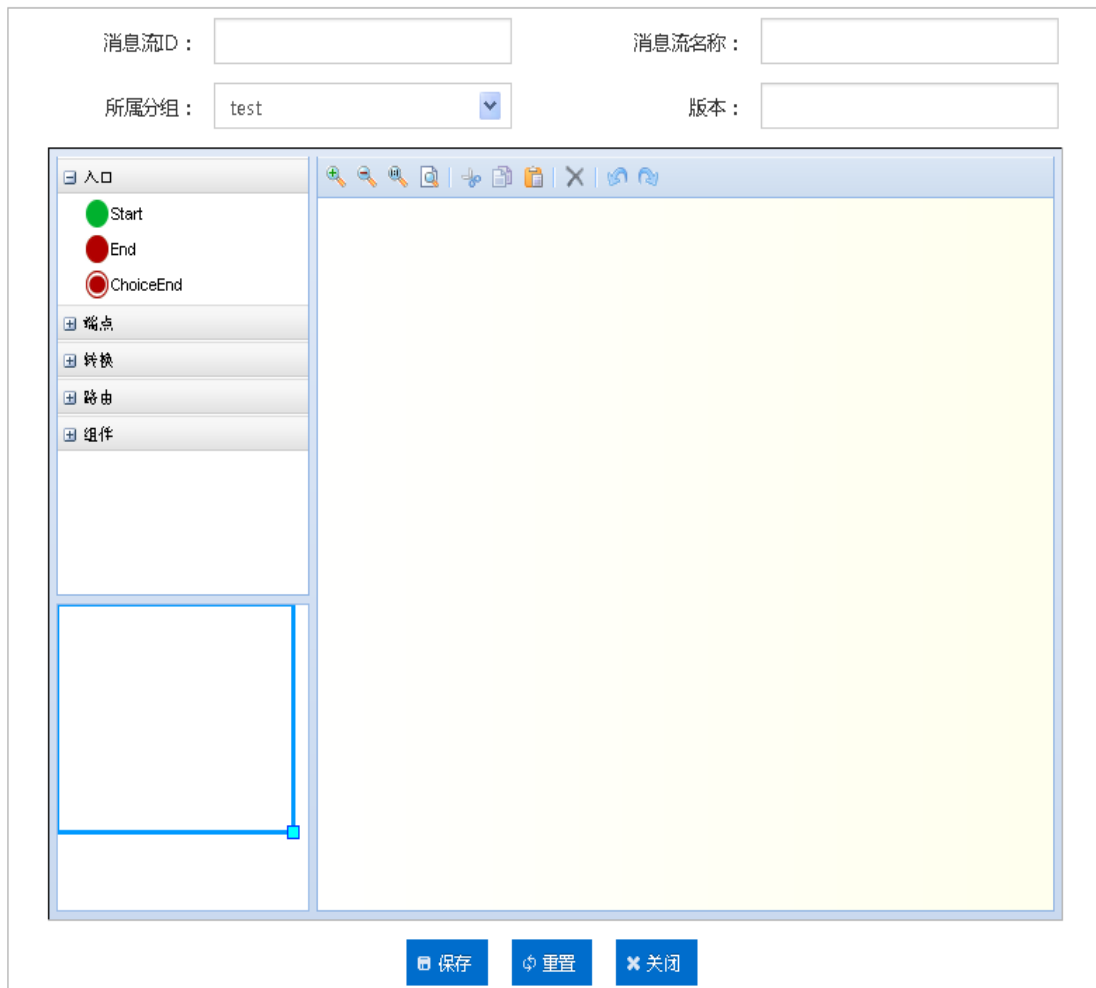
假设公司目前有三个处理请假事件的接口，分别处理小于等于一天的，大于一天小于十天的和大于等于十天的，现要求定义一个消息流提供一个请假处理接口并实现请假流程的自动分发。

2.1.2 请假流程服务注册

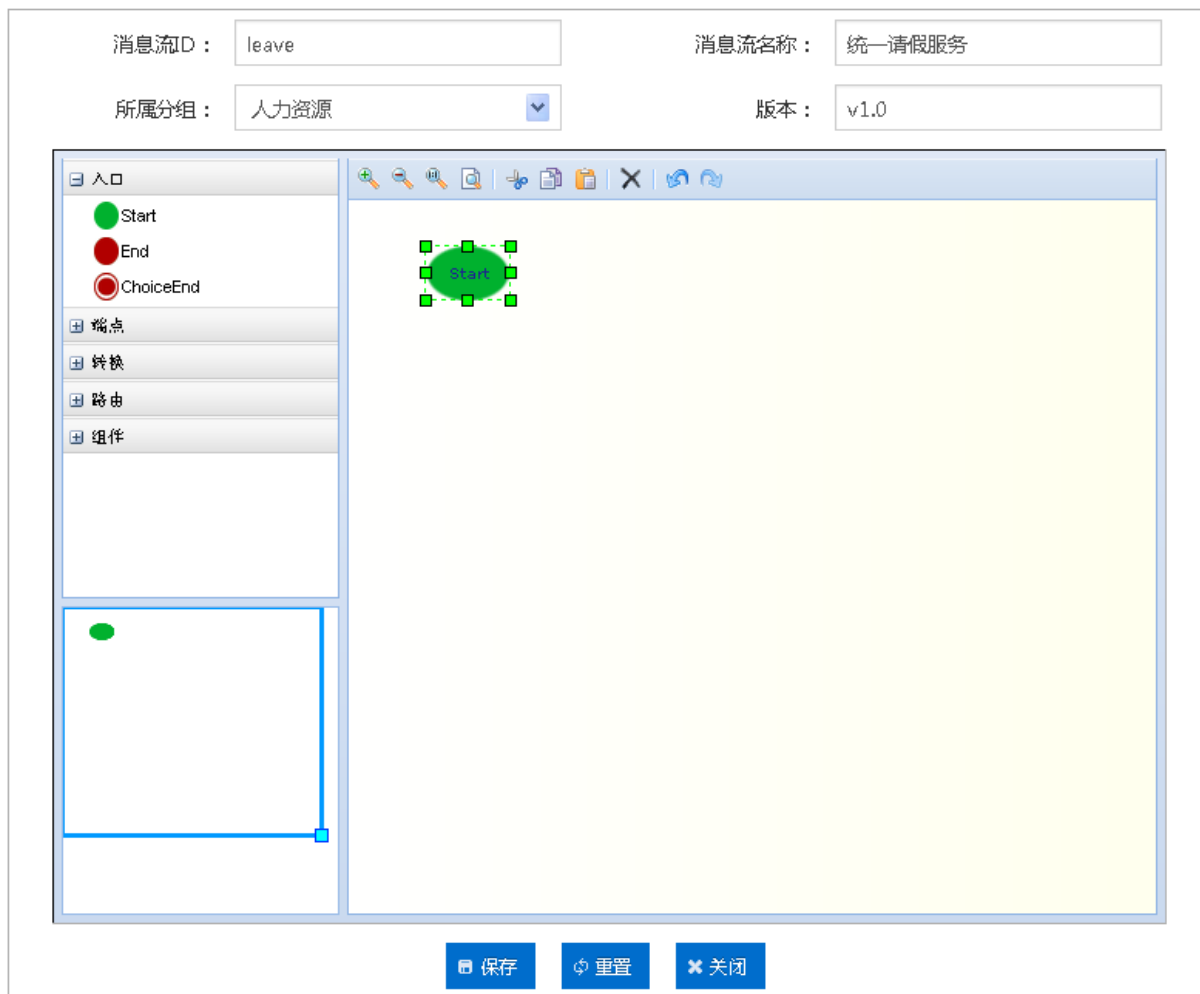
参见用户开发文档，根据服务注册管理注册相应的服务。


2.1.3 请假服务消息流定义流程

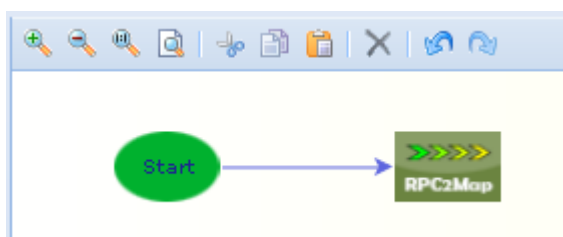
步骤1. 点击左侧树菜单的【消息流管理】，在弹出右侧界面上点击【增加】，弹出一个新的增加消息流页面。如图：



步骤2. 输入“消息 ID”，“消息流名称”，“所属分组”和“版本”，然后点击左侧树菜单的【入口】=》【Start】，拖曳 Start 节点到右侧编辑区。如图：

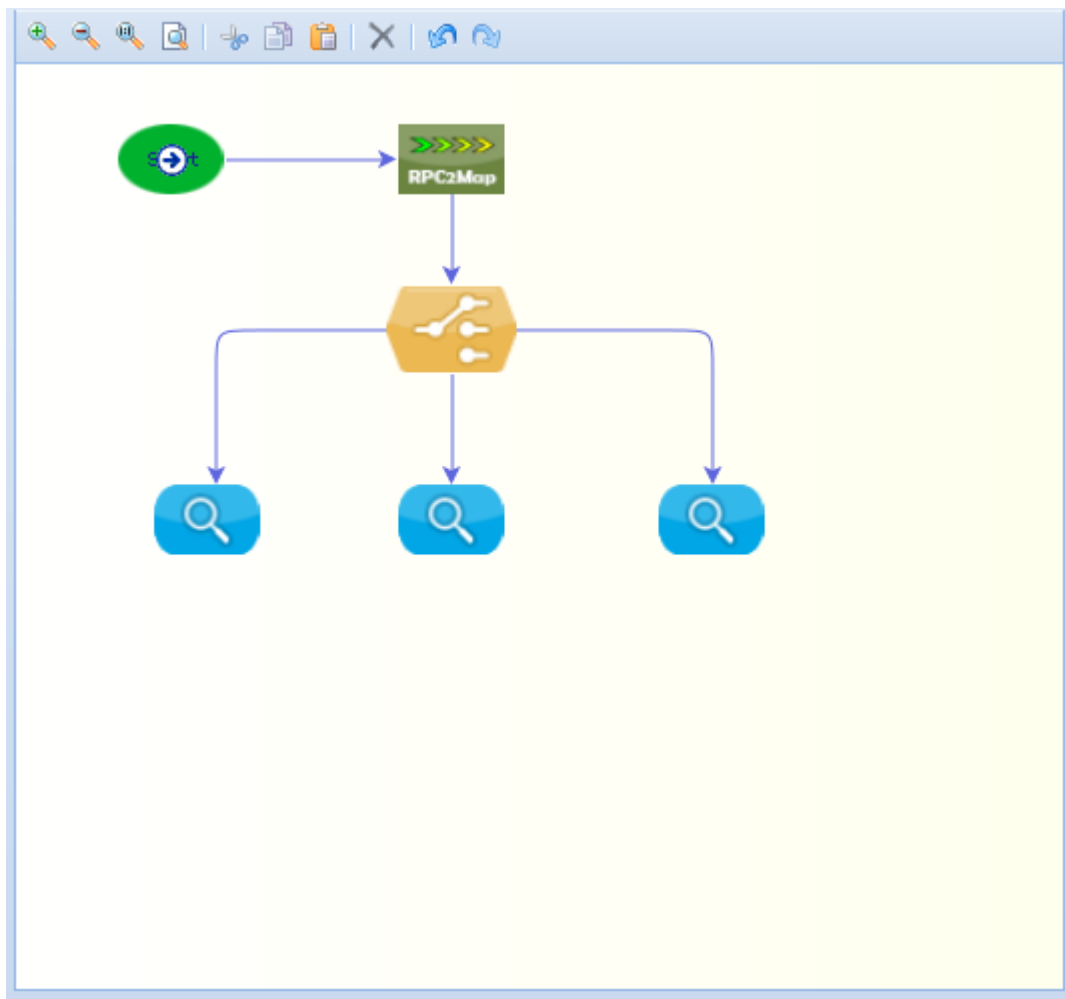


步骤3. 由于入口数据格式是 RPC 格式的, 在这里需要转换为 Map 的格式以方便调用, 所以需要拖曳【转换】=》【RPC2Map】节点到右侧编辑区, 然后把鼠标放到编辑区的【Start】节点上, 此时会显示一个小箭头如 , 点击箭头拖到【RPC2Map】节点上完成【Start】到【RPC2Map】节点的消息流连接。如图:



步骤4. 接下来我们要进行消息选择路由了, 拖曳【Choice】节点到编辑区, 然后分别拖曳三个【Reg】节点, 分别从注册库选择配置(参见 Reg 使用示例)三个不同的请假接口服务(这里默认该服务已在注册库里注册, 注册发布服务见注册库章节), 然后在【Choice】和三个【Reg】节点之间

建立边线。如图：



步骤5. 现在配置【Choice】到三个【Reg】节点之间路由规则表达式。左键双击【Choice】、【Reg】两个节点之间的连线，分别在弹出界面的 expression 属性框中填入对应的如下内容：

`#[message.content['days'] <= 1]`

`#[message.content['days'] > 1 and message.content['days'] <10]`

`#[message.content['days'] >=10]`

如图：



属性

name

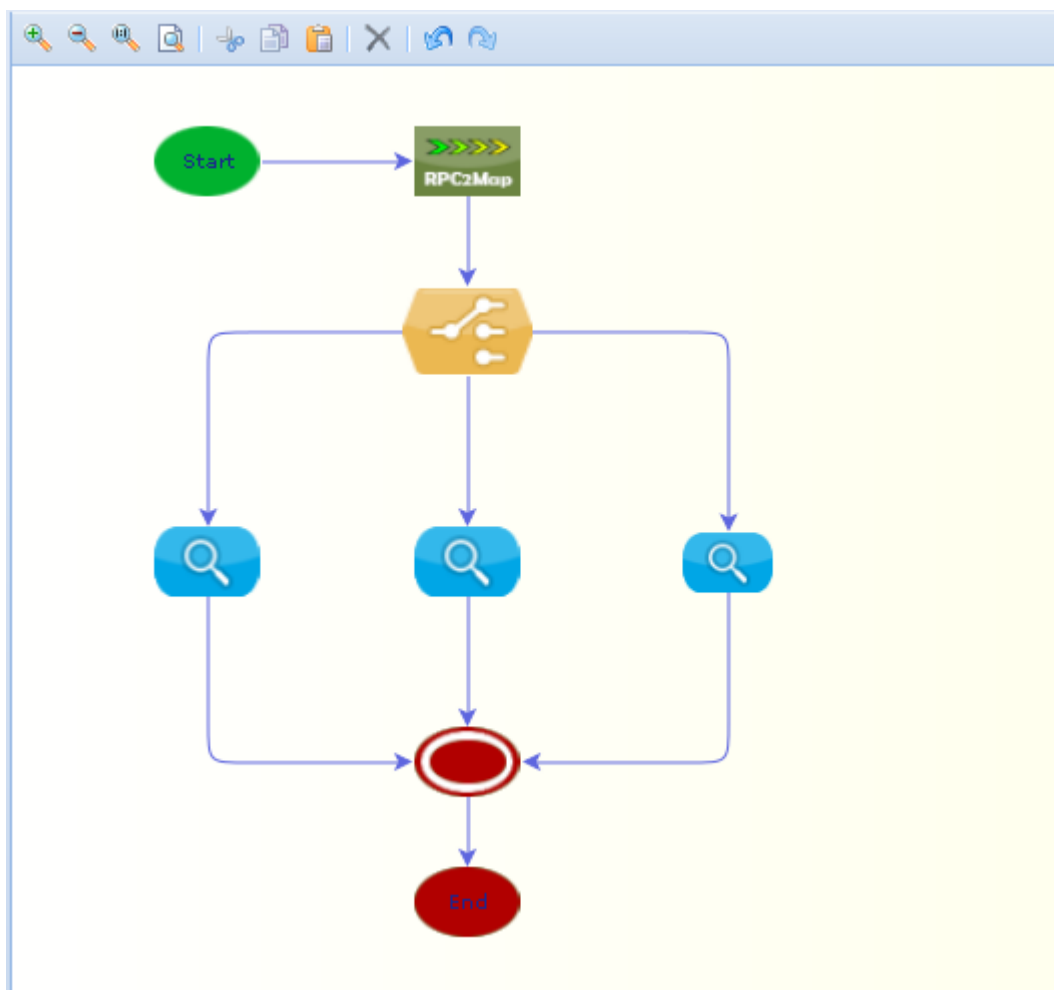
expression

✓ 确定 ✕ 取消

注意：表达式填写要与【Reg】节点配置的接口要对应。

步骤6. 路由规则配置完成之后需要结束路由，对应节点是【ChoiceEnd】。

拖曳【ChoiceEnd】节点，然后在三个【Reg】路由节点到【ChoiceEnd】建立结束路由的连线。到此整个流程结束了，不要忘记在消息流最后拖曳【End】节点来结束消息流。最终流程图见图：



步骤7. 点击【保存】按钮保存整个消息流。

2.1.4 请假服务代码示例

```

<!-------服务端代码：1 天以下----->
Public      class      LeaveHandlerExpress      implements
SimpleProviderHandler {
    @Override
    public Object handle(ServiceData arg0) {
        return arg0.getParams().get("name") + "：您的请假一天以
下，将报送快速请假流程";
    }
}

<!-------服务端代码：10 天以下--->
public      class      LeaveHandlerComplex      implements
SimpleProviderHandler{
    @Override
    public Object handle(ServiceData arg0) {
        return arg0.getParams().get("name") + "：您的请假一天以
上，走常规审批程序";
    }
}

<!-------服务端代码：10 天以上----->
public      class      LeaveHandlerBlock      implements
SimpleProviderHandler {
    @Override
    public Object handle(ServiceData arg0) {
        throw new ProviderException("E-DENIED", "自动请假系统不
受理十天以上的请假请求，请直接向上级领导申请");
    }
}

<!--      -----      监      听      注      册
----->

public void contextInitialized(ServletContextEvent arg0) {
    ProviderHolder.registry("leaveExpress",      new
LeaveHandlerExpress());
    ProviderHolder.registry("leaveComplex",      new
LeaveHandlerComplex());
    ProviderHolder.registry("leaveBlock",      new
LeaveHandlerBlock());
}

<!--      -----web.xml      配      置
----->

<servlet>

```



```
<servlet-name>provider</servlet-name>
<servlet-class>com.icss.soa.esb.sdk.provider.EsbServiceSe
rvlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>provider</servlet-name>
<url-pattern>/esb-services/*</url-pattern>
</servlet-mapping>
<listener>
<listener-class>com.csi.hello.util.RegistryListener</list
ener-class>
</listener>
```

2.2 消息流定义示例二【天气预报】

2.2.1 天气预报业务描述

实现根据城市名称获取该城市的天气预报，并将结果以 Map 的形式返回。

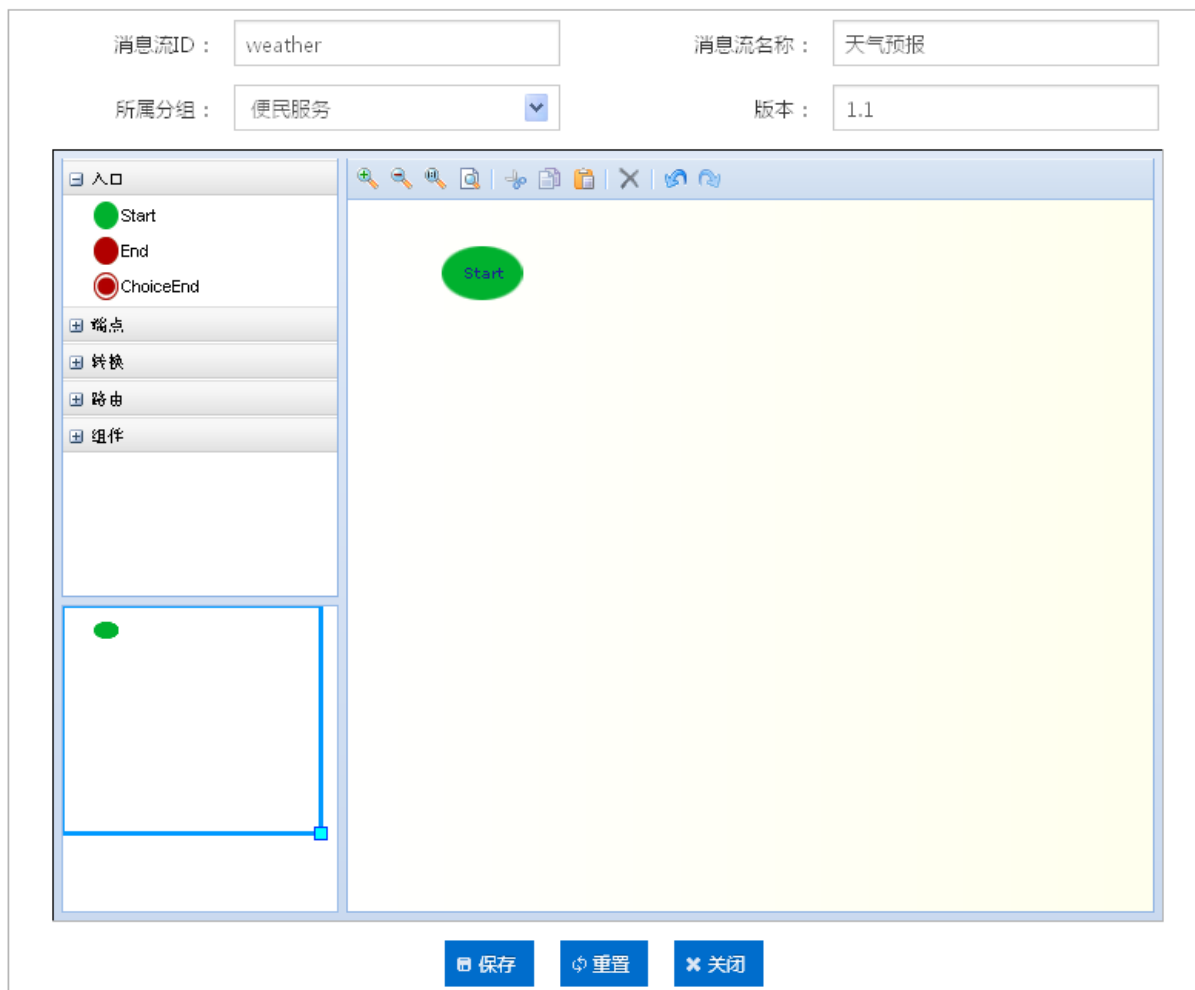
2.2.2 天气预报服务注册

参见用户开发文档，根据服务注册管理注册相应的服务。

2.2.3 天气预报消息流定义流程

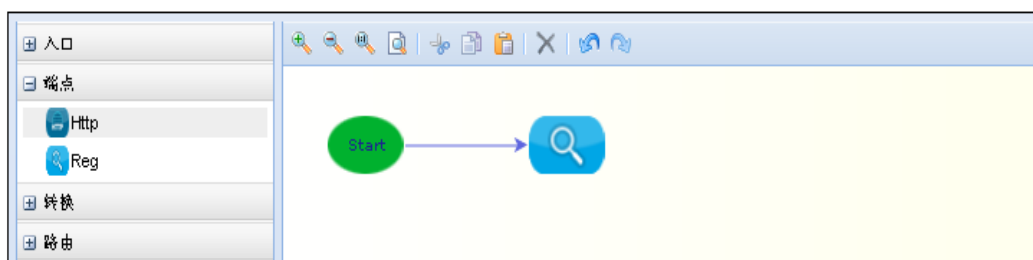
步骤1. 填写基本信息，建立消息开始节点。

点击左侧树菜单的【消息流管理】，在弹出右侧界面上点击【增加】，弹出一个新的增加消息流页面。如图，然后输入“消息 ID”，“消息流名称”，“所属分组”和“版本”，然后点击左侧树菜单的【入口】⇒【Start】，拖曳 Start 节点到右侧编辑区，如图。



步骤2. 根据城市名获取城市代码。

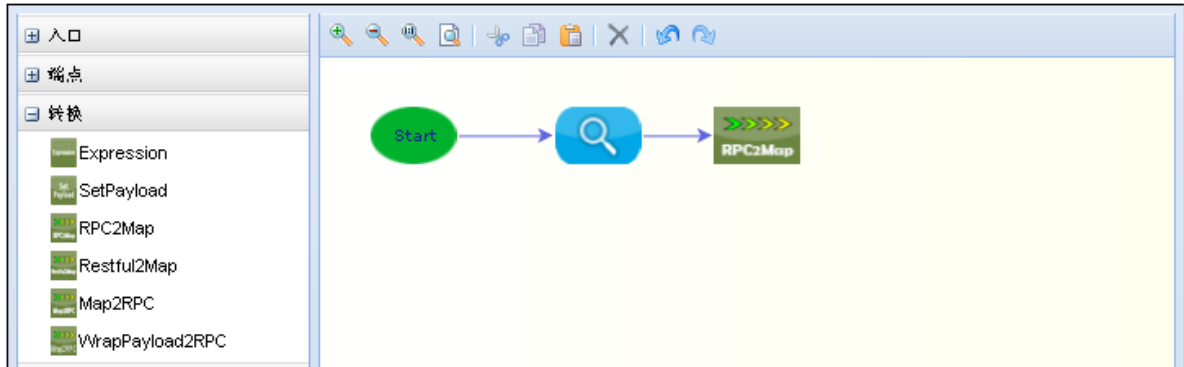
从编辑区左侧【端点】节点分类下拖曳一个【Reg】节点到编辑区，从注册库选择“根据城市名称获取城市代码”的服务(注册库默认已经配好)，同时建立【Start】节点到【Reg】节点之间的连线。见图



步骤3. 处理城市名获取城市代码结果。

为方便后续结果处理，这里先把获取到的城市代码结果转换成 Map。

拖曳【转换】=》【RPC2Map】节点到编辑区，同时建立【Reg】到【RPC2Map】的连线（见图）。

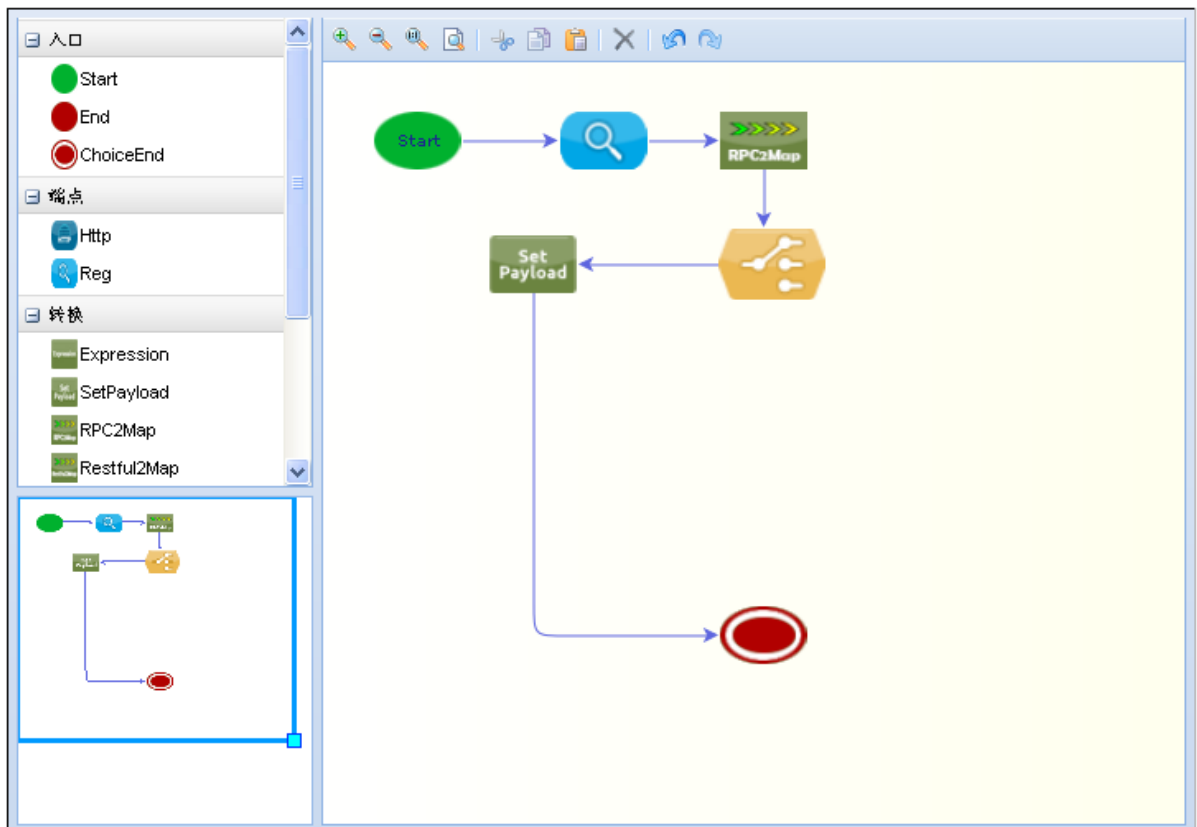


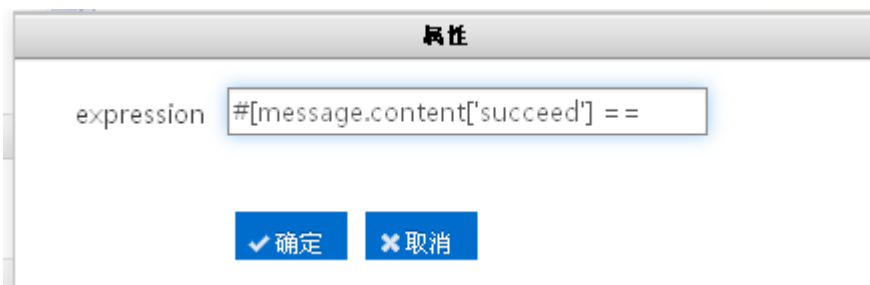
步骤4. 然后我们对结果进行判断选择，如果结果为空，结束流程。

依次拖曳【Choice】、【SetPayload】和【ChoiceEnd】节点到编辑区，并依次分别建立连线（见错误!未找到引用源。），同时填写该分支的路由选择表达式：

```
#[message.content['succeed'] == false]
```

如图

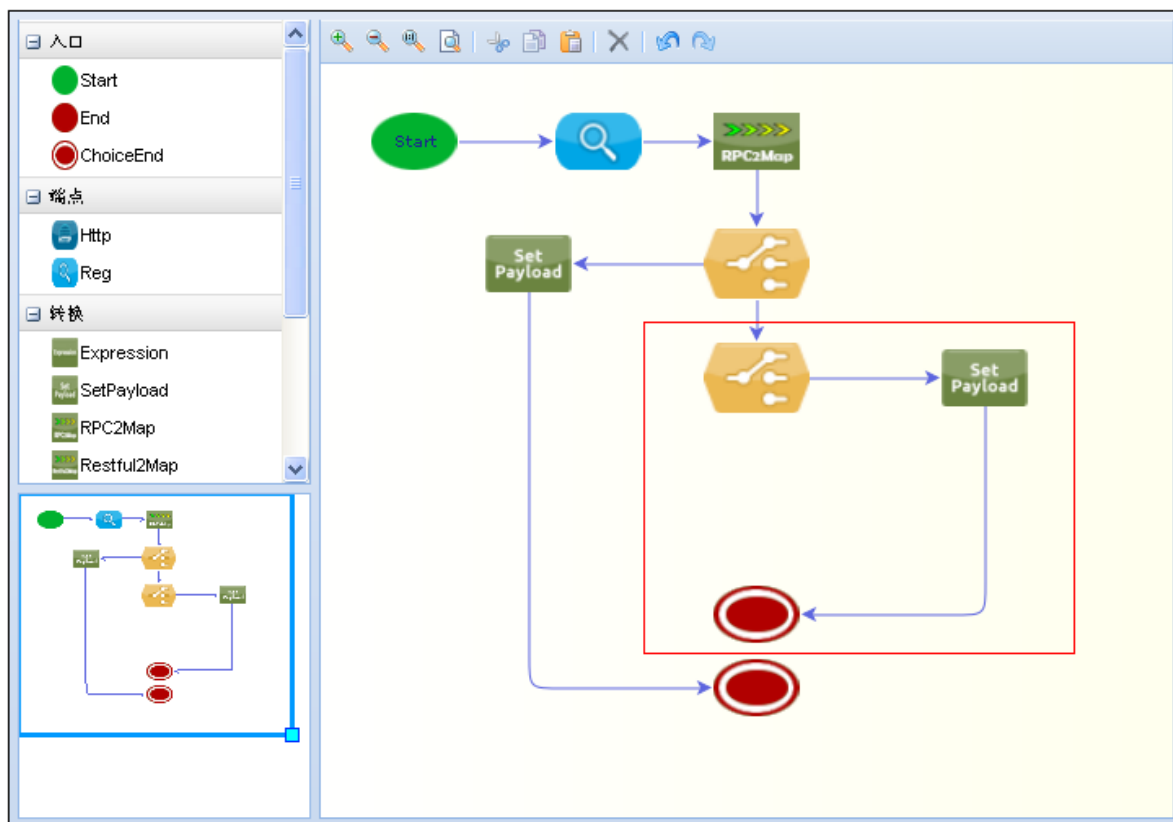




步骤5. 如果城市代码不为空，从天气预报官网获取天气信息。

由于涉及到从第三方获取天气信息，同样会遇到结果为空的情况，所以在这里需要再次对消息进行路由选择。拖曳【Choice】、【SetPayload】和对应的【ChoiceEnd】节点到编辑区，并依次建立连线（如图），同时填写【Choice】到【SetPayload】分支的路由选择表达式：

`#[message.content['result']] == 'null'`

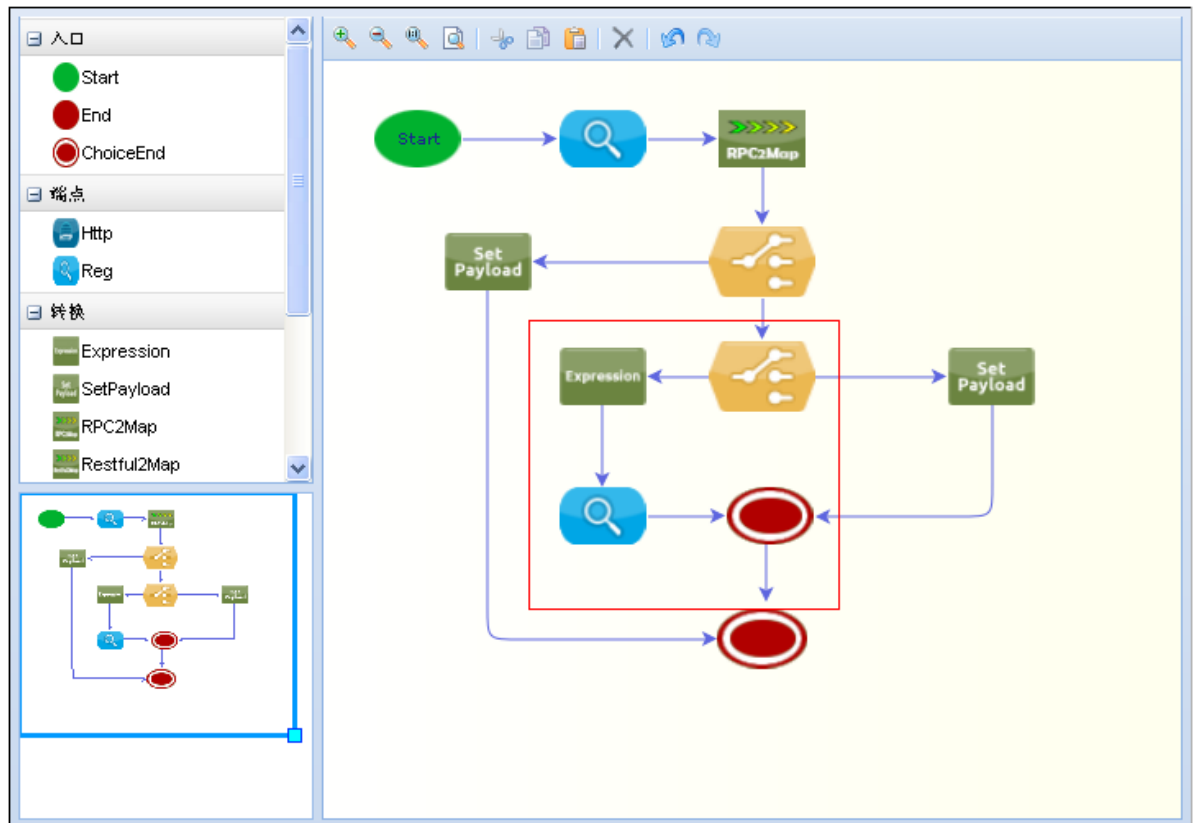


步骤6. 从天气官网获取天气信息。

拖【转换】=》【Expression】和【Reg】到右侧编辑区。并在【Expression】填写表达式：

```
#[message.content['cityCode'] = message.content['result']]
```

【Reg】为“中央气象台提供的天气预报 api”（默认已经在注册库里配置），最后消息节点之间建立连线，此时流程图如图。



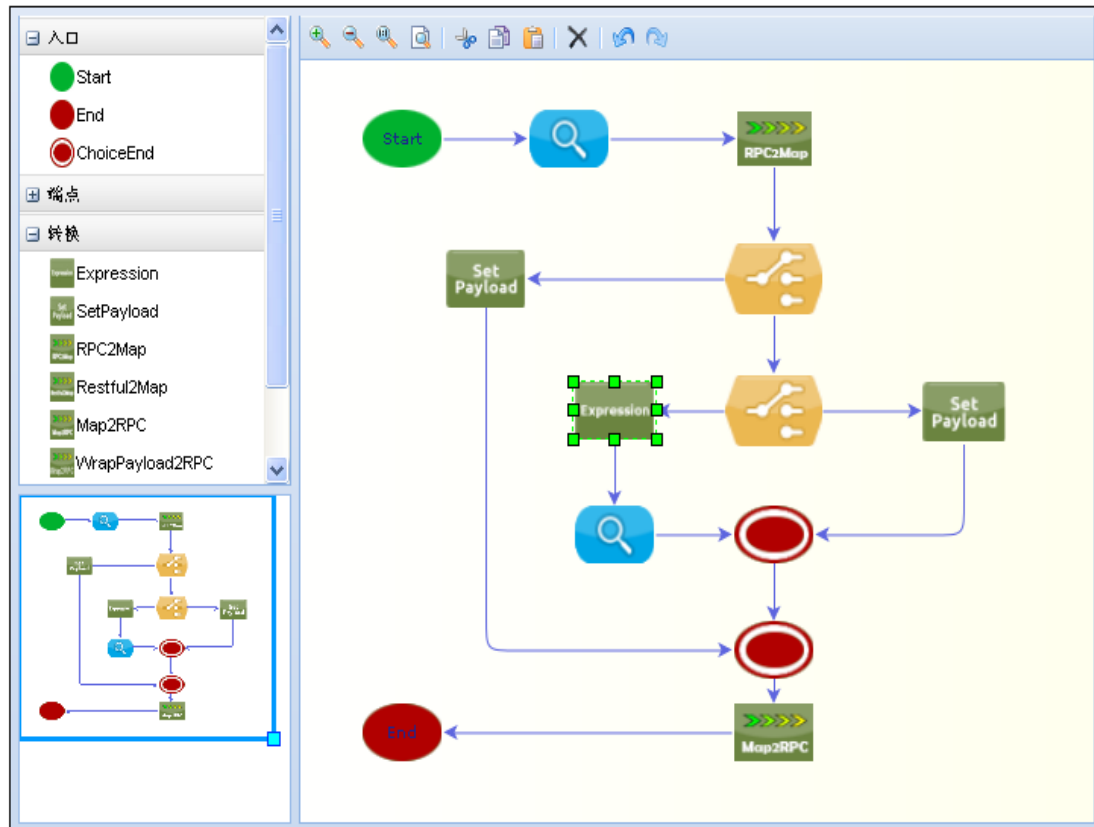
步骤7. 将结果转换为 RPC 格式。

拖【转换】⇒【Map2RPC】到右侧编辑区，并连接消息节点。

步骤8. 结束并保存整个消息流。

拖【入口】⇒【End】到右侧编辑区并连线完成消息流。点击【保存】

按钮保存消息流。如图



2.2.4 天气预报代码示例

```
<!--获取 code 代码-->
package com.csi.city.service.impl;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.HashMap;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.InitializingBean;
import com.csi.city.service.CityCodeService;
public class CityCodeServiceImpl implements
CityCodeService, InitializingBean {
    private final static Logger log = LoggerFactory
        .getLogger(CityCodeServiceImpl.class);

    private HashMap<string, string=""> cities = new
HashMap<string, string="">();
```

```

        @Override
        public void afterPropertiesSet() throws Exception {
            InputStream is = getClass()
                .getResourceAsStream("/weathercity.code.
txt");
            try {
                BufferedReader br = new BufferedReader(new
InputStreamReader(is,
                    "utf-8"));
                String line;
                while ((line = br.readLine()) != null) {
                    int idx = line.indexOf(",");
                    if (idx <= 0) {
                        continue;
                    }
                    cities.put(line.substring(0,
idx).trim(),
                        line.substring(idx + 1).trim());
                }
            } finally {
                try {
                    if (is != null) {
                        is.close();
                    }
                } catch (IOException e) {
                    log.error("Error closing inputstream",
e);
                }
            }
        }

        public String getCityCode(String cityName) {
            if ("北京".equals(cityName)) {
                return "101010100";
            }
            return cities.get(cityName);
        }
    }

    <!--注册代码----- -->
    public class CityCodeServiceHandler implements
SimpleProviderHandler {

```

```

        private final CityCodeService cityCodeService =
(CityCodeService) BeanFactory
                .getBean("cityCodeService");
        @Override
        public Object handle(ServiceData data) {
            return
String.valueOf(cityCodeService.getCityCode((String) data
                .getParams().get("cityName")));
        }
    }

    <!--监听注册----->
    public void contextInitialized(ServletContextEvent arg0) {
        ProviderHolder.registry("cityCode", new
CityCodeServiceHandler());
    }

    <!--web.xml 配置----- -->
    <servlet>
        <servlet-name>provider</servlet-name>
        <servlet-class>com.icss.soa.esb.sdk.provider.EsbServices
ervlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>provider</servlet-name>
        <url-pattern>/esb-services/*</url-pattern>
    </servlet-mapping>
    <listener>
        <listener-class>com.csi.hello.util.RegistryListener</lis
tener-class>
    </listener>
</string></string>

    <!--客户端代码---->
    HashMap<String, Object> params = new HashMap<String, Object>();
    InfoUtil.setUserId("test");
    params.put("cityName", cityName);
    RPCResult result = RPCUtil.invoke(
        new URL(ESBUtil.getInvokeUrl(Constants.R1ESB_URL,
"weather")),
        params);
    if (result.isSuccess()) {
        print(result.getResult());
    } else {
        print("查询失败: " + result.getErrorCode() + "\n"

```



```
+ result.getErrorDetail());  
}
```

第三章 开发指导

在开发手册中,主要讲解 RCloud 云 ESB 服务设计工具支持的几个重要的特性,有关 RCloud 云 ESB 服务设计工具使用的详细说明,请参考《RCloud EnterpriseServiceBus 服务产品用户使用手册》中的 RCloud 云 ESB 服务设计工具部分。

代理服务的功能体现了服务中介的作用,它是为了实现服务消费者与服务提供者关注点的隔离,通过提供虚拟服务与服务的动态、静态调用降低服务消费者与服务提供者的耦合。通过封装可重用的业务服务定义,代理服务屏蔽了对后台服务的直接访问,使得后台业务服务实现的切换对服务消费者完全透明;实现了根据业务规则实现动态的服务路由;在服务交互双方之间提供协议与消息转换;代理服务消息流也可以使用带分支的流程,但不支持多条分支同时满足条件输出消息流,因此分支迁移线上需设置条件,使得只能有唯一方向的消息流输出。

3.1 服务封装设计

使用 RCloud 云 ESB 服务的代理服务,需要设计代理服务消息流,为调用请求者定义服务接口,并配置所代理的 Web 服务的参数。

3.1.1 代理服务接口定义

用户在 RCloud 云 ESB 服务设计工具中可以定义代理服务的接口,包括服务名称,操作名称,命名空间和服务地址。另外,在映射设计中还可以指定操作的请求参数与返回值。

3.1.2 代理服务调用方式

调用代理服务时,请求的 SOAP 消息的格式符合代理服务中定义的消息格式。

3.1.3 代理服务返回值

通过 SDK 开发，返回对象是 RPCResult 对象，该对象说明请查看 API 说明。

3.2 服务调用

消息流程中的【服务调用】节点，可以同步调用用户配置的外部 Web 服务，如果被调用的服务有响应消息，那么消息流引擎会根据节点的响应处理设置自动将响应消息的内容插入或替换到 XPATH 指定的源消息位置，消息流向下流转时，会使用更新后的消息内容。

Web 服务调用节点的配置说明请参考《RCloud 云 EnterpriseServiceBus 服务产品用户使用手册》，以下说明 WEB 服务调用节点的功能。

第四章 开发建议

4.1 日志系统的使用

RCloud 云 ESB 服务提供了独立、友好的日志系统。基于 RCloud 云 ESB 服务进行二次开发时，可以使用 RCloud 云 ESB 服务自身的日志系统。

RCloud 云 ESB 服务提供日志查看服务，可以在管理控制台进行实时的日志查看。采用 RCloud 云 ESB 服务日志系统，便于在远程清晰的监控程序的运行状态。同时，可以和 RCloud 云 ESB 服务使用公共的日志配置文件，统一对日志级别与显示格式进行控制。