
EMP 通用培训教材

Tutorial

For Version1.2

文档信息

文档版本编号:	1.1	文档版本日期:	2007年08月06日
起草人:	彭楫洲	起草日期:	2007年03月10日

版本记录

版本编号	版本日期	创建/修改	说明
0.8	2007/3/10	彭楫洲	创建提纲和草稿
0.81	2007/3/24	张佳巍、李嘉	创建课程详细流程
0.9	2007/4/2	张佳巍、李嘉	修订
1.1	2007/7/31	程钊希	修订
1.2	2008/1/28	彭楫洲, 程钊希	修订
1.3	2008/3/15	彭楫洲, 程钊希	修订
1.4	2008/7/8	彭楫洲, 程钊希	改版, 增加了基础知识的介绍和练习, 更强调理论学习和动手实践

目 录

1. 培训课程简介.....	5
1.1. 目的.....	5
1.2. 培训目标.....	5
1.3. 参考文档.....	6
2. EMP 基础知识实践.....	7
2.1. 什么是 EMP.....	7
2.1.1. EMP 概述.....	7
2.1.2. EMP 产品的基本构成.....	8
2.2. 课程 1: EMP 组件工厂容器.....	10
2.2.1. IOC 技术原理介绍.....	10
2.2.2. EMP 的 IOC 容器技术实践.....	11
2.2.3. EMP IOC 技术的高级应用.....	16
2.3. 课程 2: EMP 业务逻辑处理容器.....	16
2.3.1. EMP 逻辑处理的构成要素概述.....	16
2.3.2. 数据操作: Data.....	20
2.3.3. 资源结点: Context.....	27
2.3.4. 服务组件: Service.....	30
2.3.5. 格式化处理: Format.....	32
2.3.6. 交易流程: Flow.....	33
2.3.7. 业务逻辑处理相关配置文件.....	36
2.4. EMP web2.0 处理框架.....	38
2.5. EMP 运行环境.....	38
2.5.1. 服务器端运行环境.....	38
2.5.2. 客户端运行环境.....	38
3. EMP IDE 开发实践.....	39
3.1. EMP 开发支持.....	39
3.1.1. IDE 概述.....	39
3.1.2. IDE 的定义模型.....	40
3.1.3. IDE 的基本功能.....	40
3.2. 课程所需资源.....	41
3.3. 课程 3: 第一个 EMP 项目.....	42
3.4. 练习一: 建立 EMP IDE 开发环境.....	42
1.1.1. 练习二: 建立第一个 EMP 项目.....	45
3.5. 课程 4: 第一个开发实践: 客户注册.....	52
3.5.1. 客户注册的功能分析.....	53
3.5.2. 练习一: 实现业务逻辑构件: customerManager.....	53
3.5.3. 练习二: 实现表现逻辑构件.....	68
3.5.4. 练习三: 项目部署运行.....	71
3.5.5. EMP 理论知识介绍.....	72
3.5.6. 练习四: 应用优化.....	78
4. EMP 应用模拟实战.....	86
4.1. 应用项目模拟开发方式.....	86
4.2. 模拟应用项目描述.....	86
4.2.1. 应用项目流程.....	87
4.2.2. 应用项目架构.....	88

4.2.3.	交易设计.....	88
4.2.4.	数据库表结构.....	89
4.2.5.	主机交易设计.....	90
5.	开发实战课程.....	90
5.1.	课程 5: 建立应用项目.....	90
5.1.1.	课程设计.....	90
5.1.2.	了解个人网银系统基本情况.....	91
5.1.3.	新建有 EMP 支持的 Web 项目.....	91
5.1.4.	定义数据字典.....	91
5.1.5.	业务逻辑处理节点 Context 设定.....	91
5.1.6.	手工复制课程相关文件.....	93
5.2.	课程 6: 签到/签退处理.....	94
5.2.1.	客户签到交易 signOn: 数据库表 (查询).....	94
5.2.2.	签退交易 signOff: endSession 控制器.....	102
5.2.3.	制作主布局页面和菜单树.....	103
5.3.	课程 7: 后台访问处理.....	106
5.3.1.	虚拟主机 Dummy Host 介绍.....	107
5.3.2.	创建报文通讯处理的 action.....	109
5.3.3.	查询客户帐户交易 queryAccount.....	113
5.4.	课程 8: 交易实战练习.....	119
5.4.1.	练习一: 添加客户账户.....	119
5.4.2.	练习二: 转账交易 transfer.....	121
5.4.3.	练习三: 查询交易明细交易 tranDetail.....	127
5.5.	课程 9: 其他 MVC 控制器.....	134
5.5.1.	制作信用卡申请交易.....	134
5.5.2.	显示图表.....	137
5.6.	课程 9: 应用监控.....	140
5.6.1.	将 EMP 应用加入到监控平台.....	140
5.6.2.	启动对应用的监控.....	141
6.	附录.....	141
6.1.	公共定义.....	141
6.1.1.	session 数据.....	141
6.1.2.	公共服务.....	142
6.2.	customerManager.biz.....	142
6.2.1.	biz 配置.....	142
6.2.2.	注册.....	144
6.2.3.	签到.....	144
6.2.4.	签退.....	145
6.2.5.	查询日志.....	146
6.2.6.	申请信用卡.....	146
6.3.	accountManager.biz.....	149
6.3.1.	biz 配置.....	149
6.3.2.	添加关联帐户.....	151
6.3.3.	查询帐户信息.....	152
6.3.4.	转帐.....	153
6.3.5.	查询交易明细.....	154

1. 培训课程简介

1.1. 目的

本文档详细描述了EMP平台通用培训课程，在此基础上，面向客户，EMP平台开发人员，对应用平台开发和使用功能提供的一份培训文档。

本次培训主要是在J2EE核心交易平台（EMP）V2.0的基础上进行的，主要讲述EMPv1.1的基本结构，基于EMP的B/S结构应用系统的实现方法，并以实际的业务开发为例，重点实战讲述具体的交易开发过程。在进行实战开发过程讲述的同时，也配合进行相关的实验，来巩固对开发过程的掌握。

1.2. 培训目标

EMP平台通用培训主要是为了让参加培训的人员学会在EMP平台应用项目中熟练使用EMP开发环境进行业务功能开发实现。参加该培训的学员在完成通用课程后将具备以下的EMP平台开发能力：

- a) 理解EMP设计思想和基础架构
- b) 熟练使用EMP IDE开发工具，包括交易开发、参数配置、交易测试、项目导出及部署等
 - 掌握数据字典定义和维护
 - 掌握数据类型定义和维护
 - 掌握校验公式定义和维护
 - 新增业务组件的IDE扩展操作：action和service
 - 掌握公用数据、渠道数据和会话数据的区别和在开发环境中的维护

- 掌握开发环境中的版本同步操作

c) 熟练使用IDE工具开发业务功能

- 掌握IDE中业务逻辑目录结构及操作方式
- 配置业务流程，准确定义业务跳转处理
- 配置数据格式化处理
- 配置数据库表访问服务：TABLE访问、SQL方式、存储过程访问等
- 配置通信服务：TCP/IP通信服务及其他通信组件

d) 熟练使用IDE工具开发web应用逻辑

- 理解EMP的MVC模型及配置文件内容
- 掌握IDE环境下的web应用逻辑的目录结构和部署方式
- 应用IDE工具配置web应用逻辑
- 掌握SessionController的使用时机和技巧
- 掌握RequestController的使用范围和技巧
- 掌握WizardController的使用范围和技巧
- 掌握endSessionController的使用范围和技巧
- 掌握web应用图表功能配置
- 掌握核心web表现组件：菜单树、EMPTaglib

1.3. 参考文档

文档名称	文件名称	版本号	备注
EMP2.0技术手册	EMP2.0技术手册	1.0.0	

2. EMP 基础知识实践

2.1. 什么是 EMP

EMP项目的实际目标是针对CTP原有版本的一次重大升级。完成本研发项目后，EMP的功能将进一步增强，作为一个具有开发、运行和监控管理能力的完整的J2EE基础应用框架和应用解决方案，促进平台的直接销售；降低开发技术门槛、促进提高项目实施生产率。

2.1.1. EMP 概述

EMP 是一个基于J2EE体系的Web应用的基础框架平台，通过其提供的基础框架、内建的多渠道支持、丰富的基础组件、强大的开发工具及运行维护工具，让用户快速构建具备多渠道支持的企业级应用系统。他是一个集企业级应用系统的开发、运行、管理、监控及维护功能的中间件平台，一方面透过底层的应用服务器技术承载J2EE体系，一方面以组件化的技术实现更加贴近业务以及最终应用。

EMP平台透过其提供的业务逻辑处理容器技术，将J2EE体系规范、组件技术和可视化开发技术完美结合起来，为基于J2EE平台之上的应用提供了面向组件的应用架构，通过图形化的可视开发工具IDE，使应用系统可以快速高质量的搭建，建成的应用系统具有较强的可管理可维护能力，同时拥有最强的需求变化响应能力，并通过组件积累来持续积累软件知识财富。

随着信息技术的不断发展，Web应用（即Browser/Server，浏览器/服务器）系统日益广泛的应用，企业对Web应用系统各个方面的要求越来越高。Web应用系统中普遍存在的一些问题也越来越受到关注，譬如系统开发质量，稳定性，系统重用性，系统开发效率，系统可维护性、可扩展性，以及新开发系统与其它系统的关系、定位等。

J2EE体系为建立复杂的分布式的企业级应用提供了良好技术支撑环境，其所提供组件化开发技术，如EJB, WebService等技术，允许企业按组件化的方式实现企业应用，提供组件的复用。但是这些组件的开发或复用均是基于代码形式的，如果没有一个良好的系统架构规划和设计，很容易导致应用系统的业务处理逻辑，表现逻辑及数据逻辑交织在一起，随着业务的发展，导致系统的各部分的关联和耦合越来越复杂，给系统的维护管理带来困难，更不用说响应快速的

需求变化了。

EMP平台同样采用组件技术架构，透过参数化控制技术，提供完全可视化的应用组装、运行、维护环境。

2.1.2. EMP 产品的基本构成

EMP作为面向组件的中间件产品，提供了完整的J2EE企业应用从开发到运行、管理、监控的工具或环境支持，同时提供了丰富的基础组件库。产品构成如下图所示：

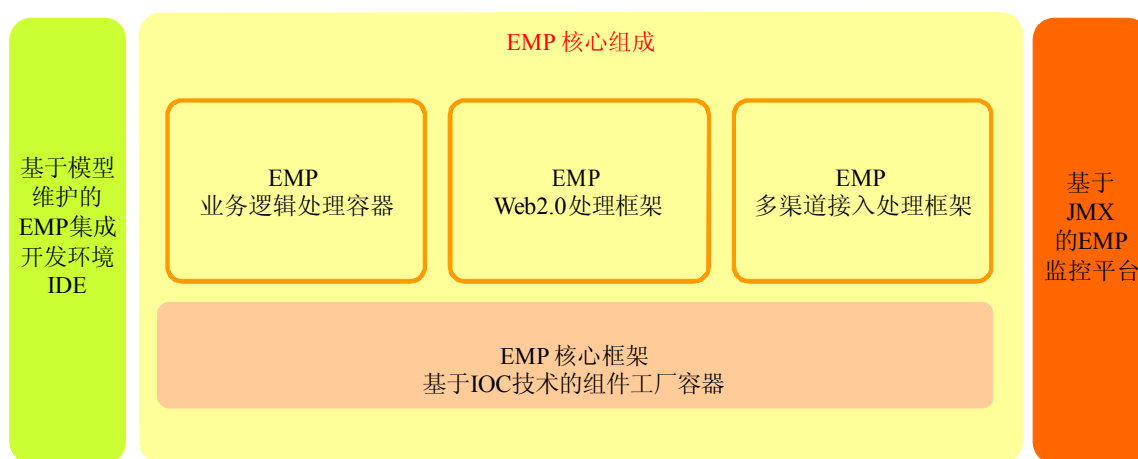


图 1 EMP Web 应用基础平台产品构成图

EMP主要包括如下几大部分内容：

2.1.2.1. EMP 运行平台

EMP 运行平台的作用就是为Web应用系统构造基础的框架结构，使得应用系统的开发不需要关心系统框架以及系统性能上的问题，而更专注于系统业务功能的实现。同时基础平台也为Web应用系统提供组件式开发与运行机制的容器，通过这一容器，具体的业务处理逻辑可以通过基础组件的参数化配置和组合来完成。

EMP平台的核心框架是基于IOC技术实现的组件工厂容器，EMP组件工厂容器通过外部XML文件配置提供Java对象的依赖注入实现，提供灵活的，扩展的外部化配置开发能力。EMP平台的所有实现均以EMP组件工厂容器为基础提供对象生成支持。

EMP平台提供了三个核心框架，分别为业务逻辑处理容器、Web2.0处理框架和多渠道接入处理框架。业务逻辑处理容器和多渠道接入处理框架均是专门针对金融行业业务模型而设计的架构，通过配置化手段快速实施引用应用系统。这也是EMP平台有别于其他通用平台的主要特点——它为行业应用提供了专业化的支持，而通用平台是没有这方面的支撑的。

EMP平台做一个轻量级的J2EE应用整合平台，它所生成的应用是标准的J2EE企业应用，可以很灵活地在各种web服务器和应用服务器上部署和应用。

EMP平台是基于JDK1.5版本进行研发的，同时也提供了针对JDK1.4版本的兼容性。它可以部署在支持JDK1.4以上的web服务器和应用服务器，如TOMCAT5.0及以上版本，JBoss3.0及以上版本，websphere5.0及以上版本，bea webLogic7.0及以上版本。

在J2EE应用中，EMP平台作为一个基础框架和应用组件的提供者提供web应用的缺省框架和可运行的初始化项目版本，EMP以jar包形式和配置文件形式提供平台支持。在EMP平台完成开发后，通过生成标准的war/ear程序包，然后在各个应用服务器上部署即可。

2.1.2.2. EMP 开发平台 IDE

EMP开发平台IDE是基于Eclipse的EMP应用开发工具，它提供EMP应用系统整个生命周期内的维护与开发。主要包括如下功能：

小组开发的支持，它与Eclipse结合配合其CVS版本管理工具实现应用系统开发过程中的小组协调功能，让参与的小组成员分工协作完成系统的开发。

组件的维护，提供EMP平台所有组件库的维护，如每个组件的具体参数设定，使用方法等，同时提供用户扩展组件的添加与维护。

基于组件的业务处理逻辑开发，提供可视化的开发工具来组装这些基础组件实现具体的业务功能处理。

基础组件使用的向导功能，提供向导完成具体的基础组件的使用，如提供数据库访问的向导，实现数据库表到平台数据的自动映射。

文档的维护与生成，通过维护应用模型，能够生成相关的设计文档。

2.1.2.3. EMP 监控平台

EMP监控平台是EMP为应用系统的运行维护而提供的一个系统的监控和管理平台，透过此平台完成对EMP应用系统的监察与控制，包括基础平台的监控和应用系统的监控。实现系统性能的监视，如系统内存的使用情况，数据库连接池的使用情况，以及基础组件的资源使用情况。

基础组件的监视与控制，提供基本的接口，可以监视到基础组件的基本情况，同时可以向基础组件发送指令更改参数。

业务处理逻辑的监控，也就是可以通过监控平台跟踪业务处理逻辑的异常情况。

2.2. 课程 1：EMP 组件工厂容器

2.2.1. IOC 技术原理介绍

自从Spring出现以来，依赖注入技术就成为J2EE应用的一种潮流。IOC带来的是一种设计方法和代码质量的优化。

通过IOC可以将Java对象很方便地在外部进行配置，并在运行时自动产生。通过IOC容器来统一管理Java对象的创建和获取，将Java编程变得更加清晰和灵活。采用IOC使基于接口的设计和编程得到最大化的应用，可以将应用对象之间的编程性耦合度转化为外部化配置文件方式的关系性描述，降低对象之间的耦合度。

看下面一段XML片断对Java对象的描述：

```
<List id="testList">
    <Item id="item1" value="value1"/>
<Item id="item2" value="value2"/>
...
</List>
```

上面描述的是一个List对象，它其中包含了两个变量对象，分别是Item对象（item1和item2）。如果采用传统的java编程的方式，我们需要进行如下类似的代码操作：

```
List list=new List();
list.setId("testList");
Item item1=new Item("item1");
Item1.setValue("value1");
list.add(item1);
Item item2=new Item("item2","value2");
list.add(tiem2);
```

其中Item与List均为Java类的实现。这种硬编码带来的问题是代码之间的耦合度太高，当需求发生改变时，可能带来的是大量的代码级变更。如上，如果item对象进行了扩展，增加了属性type，我们就不得不首先对Item对象进行修改，同时需对应用系统中所有涉及到Item对象实例化的代码进行修改，如上面的代码就需要增加:Item1.setType("1")这样的代码。

通过IOC，我们将对象之间的这种耦合关系放在代码外的配置文件进行维护，修改我们的配置文件：

```
<List id="testList">
    <Item id="item1" value="value1" type="1"/>
    <Item id="item2" value="value2"/>
    ...
</List>
```

在IOC中，xml标签会对应于一个JavaBean类，所有的Java对象都通过一个独立的IOC容器来进行创建和管理，IOC容器通过分析XML文件在运行时动态创建应用系统所需要的对象并进行对象之间的组装（如将Item对象放置到List对象中），将对象的创建和管理从编码中解放出来，我们在应用中需要的Java对象都直接从IOC容器中获得，而不需要去关心该对象是如何创建出来的。

EMP采用的IOC技术也遵循以上的设计原则，与Spring一样在IOC基础上均是同一原理，只是EMP平台在IOC容器上拥有自身的特性，它采用了更直接和更方便地描述Java对象之间的依赖关系和嵌套关系。

2.2.2. EMP 的 IOC 容器技术实践

EMP提供了通用的IOC工厂容器，通过配置规范的XML文件可以完成Java对象的创建和管理，

以及管理复杂Java对象之间的关系。从这个意义上说，EMP与Spring非常相似，也是一个通用的对象容器应用的技术平台，只不过在此基础上，集成了更多的行业应用经验和技術而成为专业平台。

我们通过动手使用EMP提供的IOC工厂来创建我们需要的Java对象。通过EMP的IOC容器来管理自己的对象非常的简单，EMP提供了通用的工厂对象ComponentFactory和解析器对象GeneralComponentParser来帮助用户完成复杂的XML文件解析和对象实例化操作。

我们通过XML文件来描述在金融交易所所需要使用到的数据对象。考察一支交易所需要的数据对象，可以采用一个集合对象来保存一支交易所需要用到的交易数据，这些交易数据具有简单的名称/数据值的属性。

2.2.2.1. 练习一：描述我们的交易数据对象

描述简单的一个交易数据：id=customerId, value=“0001001”。以上描述的是一个客户编号的简单数据对象，我们使用一个类来封装它：DataField，它有两个属性，一个是id，一个是value。Id表明了该数据的具体含义，value表明了该数据的当前值。

一支交易的数据由许多简单数据对象构成，我们使用一个类来描述一支交易的对象：DataKColl类，该类有两个属性：一个属性是Id，代表该交易数据集合的命名；一个属性是datas，是一个Map类型，用来存放所有的简单数据对象（DataField）。在eclipse中建立一个Java项目来完成代码编写工作：

（注：将培训教材提供的emp.jar包放置到该项目的构建路径上buildPath）

DataField类：

```
public class DataField {
    String id;
    String value;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getValue() {
        return value;
    }
    public void setValue(String value) {
```

```
        this.value = value;
    }
}
```

DataKColl类:

```
package com.yucheng.emp.ioctest;

import java.util.HashMap;
import java.util.Map;

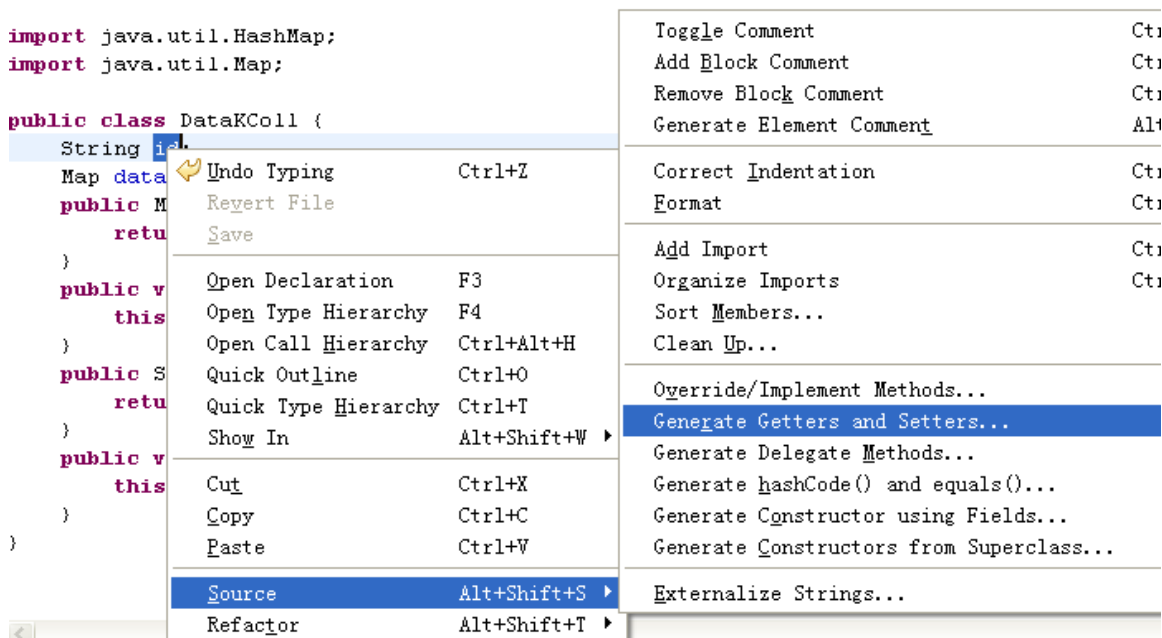
public class DataKColl {
    String id;
    Map datas=new HashMap();

    public Map getDatas() {
        return datas;
    }
    public void setDatas(Map datas) {
        this.datas = datas;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }

    /**
     * 将DataField加入到集合中
     * @param field
     */
    public void addField(DataField field){
        datas.put(field.id, field);
    }

    /**
     * 从集合中获取指定数据对象
     * @param id
     * @return
     */
    public DataField getField(String id){
        return (DataField) datas.get(id);
    }
}
```

提示，你可以采用Eclipse的快捷菜单来帮助快速生成标准的属性操作方法，如下图：



2.2.2.2. 练习二：配置我们的 XML 文件

我们通过XML文件来描述一个账户信息对象，它拥有以下数据对象：账户号、账户名、账户余额、账户类型等。通过XML文件可以如下描述：

```
<?xml version="1.0" encoding="UTF-8" ?>

<data.xml>
  <classMap>
    <map id="kColl" class="com.yucheng.emp.ioctest.DataKColl"/>
    <map id="field" class="com.yucheng.emp.ioctest.DataField"/>
  </classMap>
  <kColl id="accountInfo">
    <field id="accountNo" value="1234567890123456789"/>
    <field id="accountName" value="pengjizhou"/>
    <field id="balance" value="2000"/>
    <field id="accountType" value="1"/>
  </kColl>
</data.xml>
```

将该文件存到一个指定目录下，如C: \。

以上的XML文件是EMP平台中对IOC对象容器的标准格式。

ClassMap中定义的是在该XML文件中所使用到的所有Tag(标签)的对应类映射定义。如kColl标签在运行时，系统会将它解析为一个DataKColl类对象，并将它的id属性自动赋予该类对象。

```
<kColl id="accountInfo">
  <field id="accountNo" value="1234567890123456789"/>
  <field id="accountName" value="pengjizhou"/>
  <field id="balance" value="2000"/>
  <field id="accountType" value="1"/>
</kColl>
```

kColl中包含若干个field, 描述的是一种对象之间的关系, 在运行时, 根据标签名称,kColl对象首先查找setField方法, 如果没有则查找addField方法来构建两个对象之间的关系。

2.2.2.3. 练习三：通过 EMP IOC 容器来管理对象

通过EMP IOC容器来管理对象是十分简单的, EMP作为一个容器可以非常方便地帮助用户来扩展和设计自己的应用对象并统一进行管理。参考下面的类实现:

```
package com.yucheng.emp.ioctest;

import com.ecc.emp.component.factory.ComponentFactory;
import com.ecc.emp.component.xml.GeneralComponentParser;

public class DataIocTest {
    static public void main(String[] arg){
        ComponentFactory factory=new ComponentFactory();
        GeneralComponentParser parser=new GeneralComponentParser();
        factory.setComponentParser(parser);
        factory.initializeComponentFactory("datas", "c:\\data.xml");
        try{
            DataKColl
kColl=(DataKColl) factory.getComponent("accountInfo");
            System.out.println(kColl.getField("accountNo").getValue());
            System.out.println(kColl.getField("accountName").getValue());
            System.out.println(kColl.getField("balance").getValue());
            System.out.println(kColl.getField("accountType").getValue());
        }catch(Exception e){
            System.out.println(e.toString());
        }
    }
}
```

EMP提供了两个类来完成对象的创建和管理:

com.ecc.emp.component.factory.ComponentFactory

提供工厂容器对象, 通过ComponentFactory factory=new ComponentFactory();直接得到一个工厂容器。

com.ecc.emp.component.xml.GeneralComponentParser

提供一个解析器对象，通过它来处理XML文件，该过程对用户是透明的，无须关注。只需要将该对象赋予工厂对象即可。如下面的代码：

```
GeneralComponentParser parser=new GeneralComponentParser();
```

```
factory.setComponentParser(parser);
```

最后调用：`factory.initializeComponentFactory("datas", "c:\\data.xml")`；来完成XML文件到Java对象的转换，其中datas为工厂名称，可自定义；`c:\\data.xml`为需要解析的XML文件的完全路径和文件名称，在这里采用手工输入文件名称，在实际应用中应进行相应优化处理。

然后我们就可以通过‘id’名来获取我们定义的对象了：

```
factory.getComponent("accountInfo")。
```

2.2.3. EMP IOC 技术的高级应用

以上的练习只是通过简单的对象管理来描述IOC容器的基本实现。通过该练习应可以初步掌握EMP平台在应用扩展上的实现原理。

EMP平台上的所有组件和流程处理均是以IOC容器为基础实现的，所有的组件和流程描述均是采用XML文件来进行描述和参数配置的。在强大的IOC容器支持下可以非常灵活的得到支持，在后续章节中将看到IOC对EMP平台的强大支持能力。

EMP IOC容器拥有更强大的功能，还有继承关系的处理、特殊属性（List、Map）的注入等高级功能，这些内容可参考《EMP2.0技术手册》相关章节。

2.3. 课程 2：EMP 业务逻辑处理容器

2.3.1. EMP 逻辑处理的构成要素概述

EMP逻辑处理的基本封装是业务逻辑构件（BizLogic），业务逻辑构件是将一个业务相关的流程进行整合后产生的一个独立的可描述完整业务功能的业务组件的封装对象。业务逻辑构件

提供了对外访问的输入/输出标准接口,可以开放为一个web service对象提供给外部应用访问。

EMPBizLogic的例子如下:

```
<EMPBusinessLogic id="firsttest" operationContext="firsttestSrvCtx">
  <operation id="testFlow" name="testFlow">
    <input>
      <field id="month"/>
    </input>
    <output>
      <iColl id="balanceInfos">
        <kColl>
          <field id="timeBalance"/>
          <field id="outCome"/>
          <field id="month"/>
          <field id="inCome"/>
          <field id="currentBalance"/>
        </kColl>
      </iColl>
    </output>
    <flow>
      <action id="JDBCSQLException0" implClass="com.ecc.emp.jdbc.sql.JDBCSQLException"
        trxType="TRX_REQUIRED">
        <transition conditon="$retValue='0'" dest="JDBCSQLException1"/>
        <transition conditon="$retValue='2'" dest="JDBCSQLException2"/>
      </action>
      <action id="JDBCSQLException1" implClass="com.ecc.emp.jdbc.sql.JDBCSQLException"
        trxType="TRX_REQUIRED"/>
      <action id="JDBCSQLException2"
        implClass="com.ecc.emp.jdbc.sql.JDBCSQLException"
        trxType="TRX_REQUIRED"/>
    </flow>
  </operation>
  <operation>
    ....
  </operation>
</EMPBusinessLogic>
<context>
```

```
... <!--共享的资源定义-->
</context>
<kColl id="xxx">
...<!--共享的数据定义-->
</kColl>
```

从上面的例子可以看到，一个bizLogic.xml文件中包含了多个operation流程处理，包含了共享的资源和数据定义：context，包含了统一的对外发布的输入/输出接口，整个构成了一个bizlogic的描述。

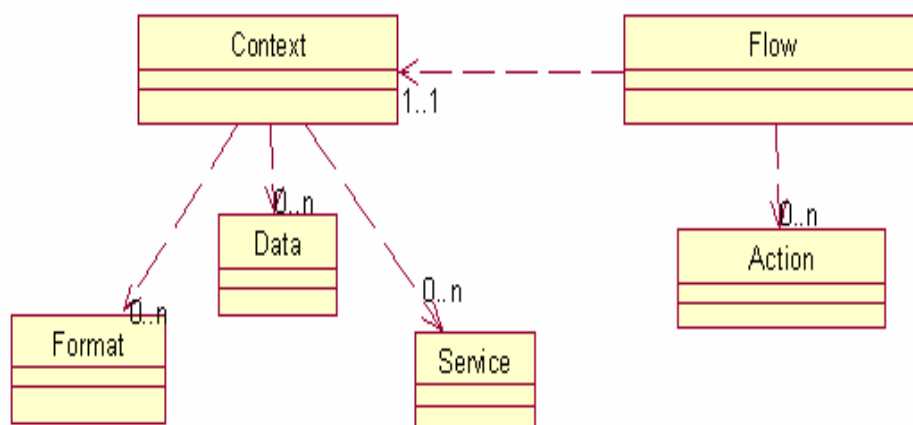
BizLogic是描述的一个完整的业务处理逻辑，比如一个转帐，它可能包含多个主机请求和应答来回，可能首先需要进行一次查询，然后填入转帐信息，在发起真正的转帐交易。所以BizLogic中会包含多个operation，一个operation所描述的是一个请求/响应来回的交易流程处理对象。它包含了该请求来回的处理流程flow和请求和响应数据，以及打包/解包的格式化定义等等。我们将一个operation称为一个交易，一个业务处理构件可以由若干个交易组成。

交易是由交易抽象模型（Operation）和交易驱动组成。这里的交易就是一次人机交互——譬如到网点柜员机取款或上网买东西，从登录系统到签退的整个过程中的每一次请求，都是一个交易（Operation）。进行交易就需要一些数据，即数据DATA；这些数据进行交互需要一定的格式即数据格式FORMAT；交易处理过程中交易数据通常需要进行数据存储或其他的一些处理，对这些处理需要提供独立的功能组件，即服务SERVICE；对一个交易而言一系列的流程之间是有关联性的，同时交易与交易之间也可能存在相关性，譬如交易数据共享。因此需要为交易定义上下文，即CONTEXT。

Operation是我们实际应用中的具体功能实现的基本构成单元。EMP平台的核心业务处理要素也是通过operation的构建来完成的。在分解一个交易时，我们可以把一支交易分解为如下几个部分：

操作流程和操作步骤、数据、格式化处理、资源访问组件（数据库/通信服务）

基于以上的分解，EMP的核心交易引擎的处理模型如下所示：



➤ Flow: 交易流程

Flow定义的是一个交易的处理流程，可能会包含记录日志，发送主机，更新日志等等操作步骤。这些操作步骤是一个个最原子的业务处理逻辑实现。通过这些交易步骤的操作结果进行条件判断来实现交易流程的分支跳转和循环处理。

➤ Action: 操作步骤

Action是最原子化的交易处理逻辑实现。它的重用度是很高的，提供的是基于业务处理逻辑上的某一个步骤的实现。

➤ Data: 数据结构

➤ Format: 格式化定义

format根据该交易所需要报文处理进行数据格式定义。在交易执行过程中，通过特定的操作步骤进行格式化定义引用完成对数据的打包/解包处理。

➤ Service: 服务组件

Service是一种服务组件，它是一个纯技术的功能实现，如数据库访问服务，service实现的是低层的技术组件，它一般不提供具体的业务逻辑，service通过在操作步骤中进行参数设置后调用来构成一个具体的业务处理功能。

➤ Context: 资源管理器

在实际应用中，我们需要管理我们的数据、格式定义和服务定义对象。在EMP平台中采用了context资源结点对象来进行以上这些资源的管理。Context根据所拥有的管理属性的不同，有不同的生命周期，如session数据的context的生命周期是伴随着操作人员签到一直到该操作人员退出之间的这段时间均会存在，而一支具体交易的生命周期当这支交易被调用时产生，但这支交易完成时就被销毁了。

2.3.2. 数据操作： **Data**

Data Model 是EMP应用的中心，所有的操作都围绕 Data Model 进行。

表现逻辑（JSP）提交给服务端的数据由 EMP 按照规则放到 Data Model 中存放，同时进行相应的规则检查（数据项是否定义等）。

Operation 处理过程中，在需要进行数据格式化时，会驱动 Format 从 Data Model 中获取数据进行格式化。

Service 同样也根据业务逻辑处理的需要被调度，Service 执行操作时同样也从 Data Model 中获取数据进行操作。

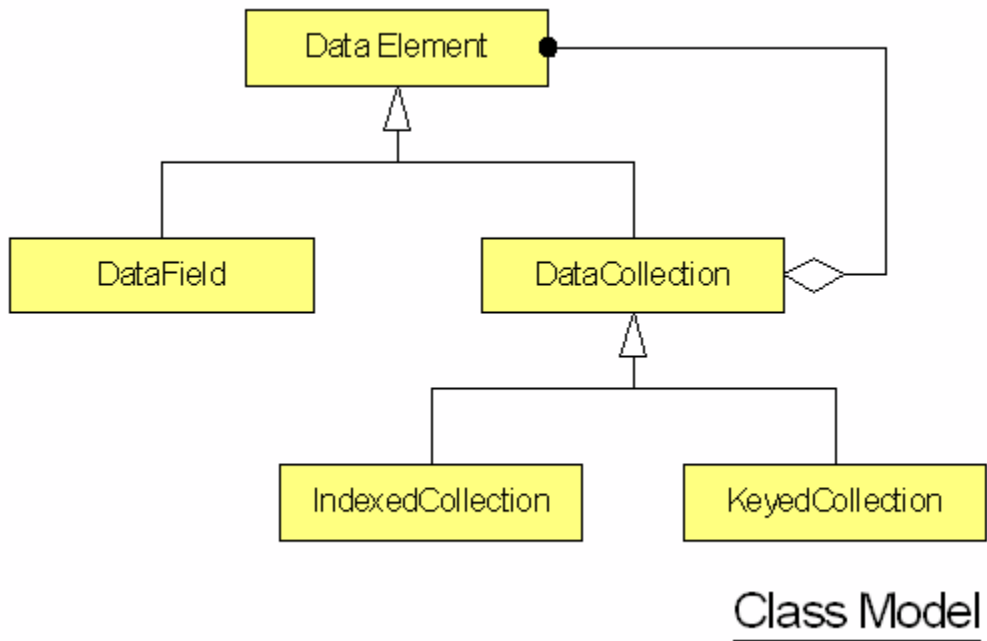
对于一个 请求、处理和结果页面的交易执行过程中，表现逻辑、业务处理逻辑、数据格式化、服务都共享同一个操作的数据模型。

➤ Data

所有的页面交互、后台处理的数据都必须在data中定义。

➤ Data Elements类关系

所有类实现于package: com.ecc.emp.data。数据分为DataField和DataCollection两类，而DataCollection又分为KeyedCollection和IndexCollection。DataField代表每个独立的数据域。KeyedCollection代表一组数据，类似于数据结构，IndexedCollection代表了一组相同的数据，通过位置索引来访问。如下图：



➤ KeyedCollection

注意:

一个KeyedDataCollection集合中不可以存在两个相同名字的元素, 但可以在内部集合中存在相同名字的另一个元素。例如:

```
<kColl id="kName1">
    <field id="field1"/>
    <field id="field2"/>
    <kColl id="kName2">
        <field id="field1"/>
        <field id="fieldOthers"/>
    </kColl>
</kColl>
```

KeyedDataCollection是不同类型的数据元素的集合, 在应用中常用于存取交易请求数据和数据库表中的一条记录的多个字段。

➤ IndexedCollection

IndexedCollection是同类型的数据元素的集合, 其构造特性和使用方法与常用的数组类似, 通过下标进行存取。在应用中, IndexedCollection常用于存取一个数据库表中的多条记录。

```
<iColl id = "customersListData" description = "list of customers" >
  <kColl id= "customerInfo" >
    <field id= "name" />
    <field id= "age" />
    <field id= "mail" />
  </kColl>
</iColl>
```

例子说明:

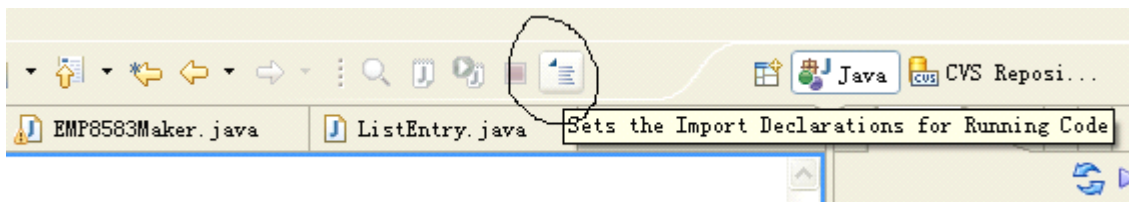
customersListData定义为用户信息列表，列表中的每一个用户定义是该用户的详细定义，包括了用户的年龄、名字和mail地址要素。

代码片段完成对customersListData定义的实例化，并提供了对IndexedDataCollection实例化对象 进行访问的示例。

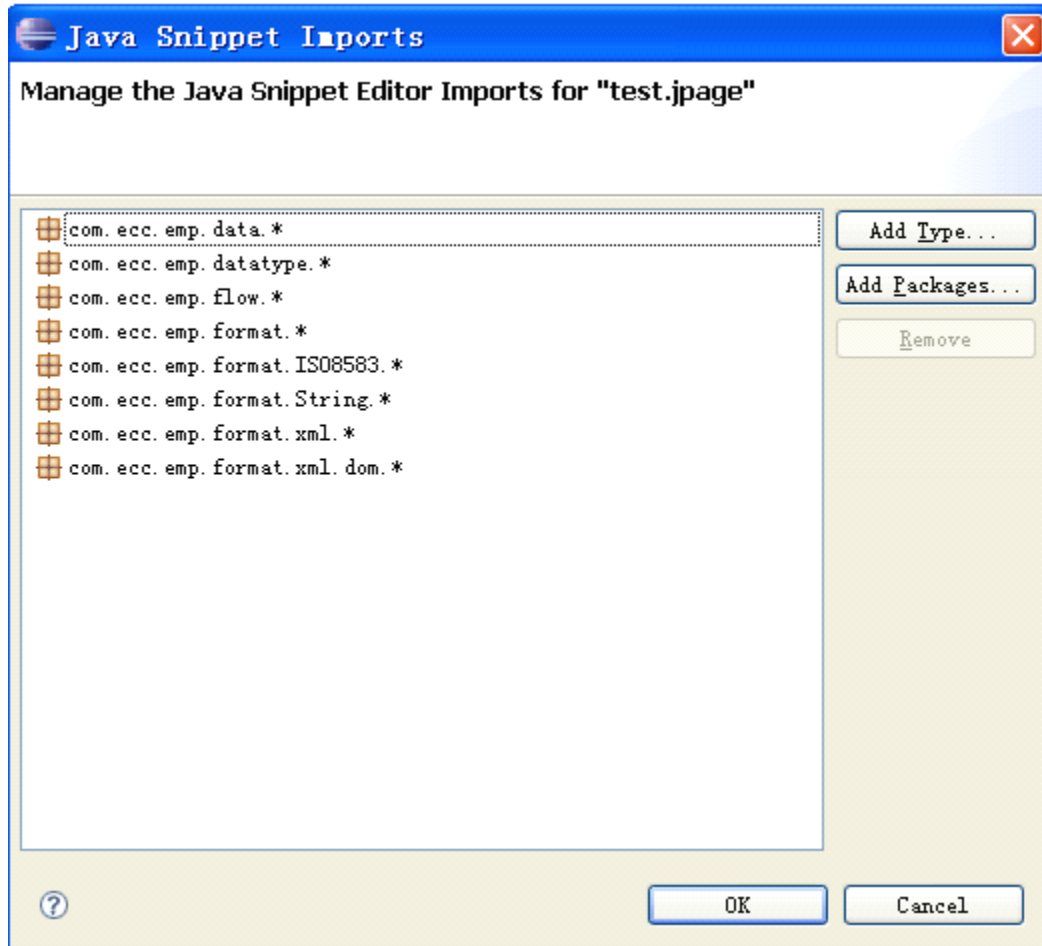
2.3.2.1. 练习一：掌握 Data 操作 API

数据操作是EMP在扩展编程中最常用的功能。我们需要熟练掌握EMP平台所提供的Data操作的相关API。实际应用中，数据对象的构建绝大多数情况下采用的是外部XML文件的注入方式，但在运行过程中对数据进行某些加工处理时，需要使用到Data操作的系列API。

使用Eclipse 的Scrapbook功能来完成一个Kcoll对象的创建并进行数据输出操作，建立一个Scrapbook页面，并加载EMP平台相关package。如下图：



选择我们需要的EMP的package加载到scrapbook页面，加载如下的package:



在scrapbook中输入如下代码:

```
KeyedCollection kcoll=new KeyedCollection();
kcoll.setId("AccountInfo");
DataField field=new DataField("accountNo");
field.setValue("1234567890123456789");
kcoll.addDataField(field);
DataField field1=new DataField();
field1.setId("accountName");
field1.setValue("testData");
kcoll.addDataField(field1);
System.out.println(kcoll.size());
for(int i=0;i<kcoll.size();i++){
    DataField element=(DataField)kcoll.getDataElement(i);
    System.out.println(element.getName()+"="+element.getValue());
}

DataField data=(DataField)kcoll.getDataElement("accountName");
System.out.println(data.getValue());
System.out.println(kcoll.getDataValue("accountName"));
```

运行该片段, console输出结果:


```
2
accountNo=1234567890123456789
accountName=testData
testData
testData
```

以上片段是通过编码的方式来操作数据对象, kColl中可以包含若干DataField对象, 可以通过addElement方法向kColl中增加数据元素, 并通过getDataElement(int)来遍历整个集合中的数据。

Kcoll可以通过getDataElement(String)方法通过输入数据元素名称来获取数据元素对象, 也可以直接调用getDataValue (String)方法来直接获得制定数据元素的值。

在实际运行中, 很少采用直接硬编码的方式来实现KeyedCollection对象, 多是采用XML配置文件方式进行数据对象的定义, 并通过组件工厂容器来获得数据对象, 以上通过代码创建的数据对象采用XML片段来表示为:

```
<kColl id="accountInfo">
  <field id="accountNo" value="1234567890123456789"/>
  <field id="accountName" value="testData"/>
</kColl>
```

2.3.2.2. 练习二：掌握对 IndexedCollecion 的操作

IndexedCollection数据元素对象代表着一个列表数组对象, 在该对象中的每一个元素都是一个KeyedCollection对象, 它们的结构完全相同, 但每一个KeyedCollection中的数据可以不同。可以想象为一个数组对象或者一个表对象 (其中的每一行数据就是一个KeyedCollection对象)。

IndexedCollection数据对象的定义在XML配置文件中一般只是定义该IColl的数据结构, 数据是在运行时注入或进行操作处理的。下面是一个iColl对象的定义:

直接在iColl中定义数据元素:

```
<iColl id="accountList">
  <kColl>
```

```

    <field id="accountNo" value="1234567890123456789"/>
    <field id="accountName" value="testData"/>
  </kColl>
</iColl>

```

也可以采用引用的方式进行定义，如下：

```

<iColl id="accountList">

  <kColl refId="accountInfo"/>

</iColl>

<kColl id="accountInfo">
  <field id="accountNo" value="1234567890123456789"/>
  <field id="accountName" value="testData"/>
</kColl>

```

在这里我们通过练习来了解一下Icoll的操作API，我们可以手工构建一个Icoll，并向其中增加、删除元素对象（kColl）：

```

//create a iColl
IndexedCollection icoll=new IndexedCollection();
icoll.setName("testIColl");
//create a kColl to iColl
KeyedCollection kcoll=new KeyedCollection();
kcoll.setId("AccountInfo");
DataField field=new DataField("accountNo");
kcoll.addDataField(field);
DataField field1=new DataField();
field1.setId("accountName");
kcoll.addDataField(field1);
//set dataElement to icoll
icoll.setDataElement(kcoll);
//add second element
KeyedCollection
secondKColl=(KeyedCollection)icoll.getDataElement().clone();
secondKColl.setDataValue("accountNo","4484949494949494");
secondKColl.setDataValue("accountName","pengjizhou");
icoll.addDataElement(secondKColl);
//add thrid element
KeyedCollection
thridKColl=(KeyedCollection)icoll.getDataElement().clone();
thridKColl.setDataValue("accountNo","2222233333444445555");
thridKColl.setDataValue("accountName","pengjz");

```

```
icoll.addDataElement(thridKColl);  
//System.out  
System.out.println("icoll size = "+icoll.size());  
System.out.println(icoll.toString());  
for(int i=0;i<icoll.size();i++){  
    KeyedCollection kColl=(KeyedCollection)icoll.getElementAt(i);  
    System.out.println("the "+i+" element content is ");  
    System.out.println(kColl.toString());  
}
```

如上面的例子，通过一个scrapbook片段来编写以上代码并运行，基本Icoll的操作API均可掌握。在实际应用中，很少直接去New一个新的Icoll，而是通过context对象去getDataElement方法去获取一个通过外部配置文件注入的Icoll对象。

在下面章节，我们将学习资源节点Context对象的内容。

2.3.3. 资源结点：Context

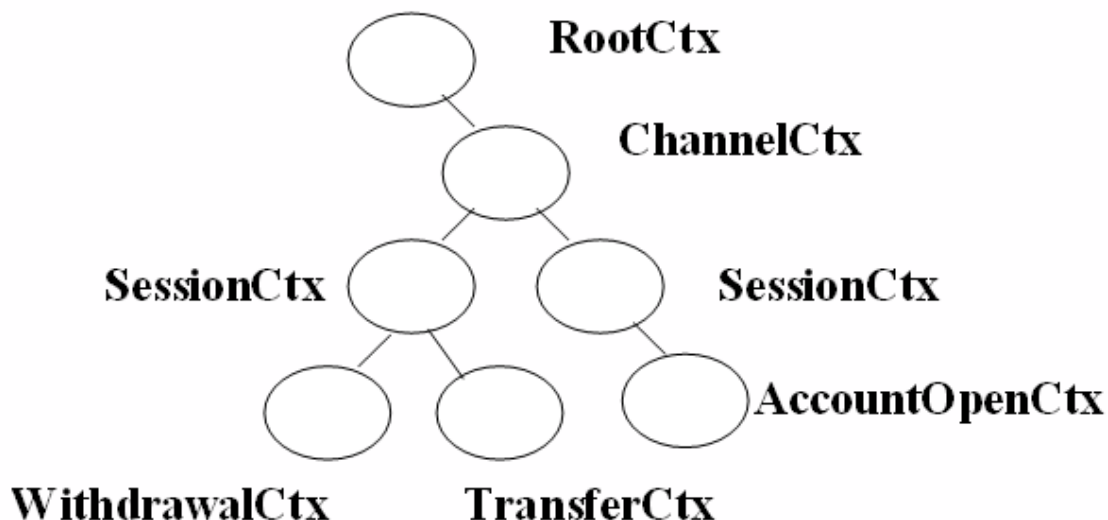
它是维护操作处理过程中所需的数据模型的一种基本框架实体。以树状结构来管理资源，子结点可以访问父结点的资源，反之不能——实现了数据共享。每一个交易都会有一个对应的context，并从中获得资源（数据和服务）动态加载资源结点。

Context是EMP运行时资源的管理对象，在其上可以管理数据元素对象、格式化对象和服务对象。所有的对象都可以通过Id的方法来获取。Context提供了三个基本方法来获取对应的资源对象：

```
getDataElement (String) //得到一个数据元素对象  
  
getFormat (String) //得到一个格式化处理对象  
  
getService(String) //得到一个服务对象
```

当要在Context节点链上查找资源时，被应用访问的资源从底层结点到根结点在每个结点中查找，第一个匹配的资源被返回。注意：返回的是第一个匹配的，但不一定是应用需要的。

Context在应用中的基本结构由以下四个层次组成：



➤ RootCtx

一个项目的根结点，用于存放该应用的一些共享的数据和服务对象。它是一个静态的对象在应用初始化后就一直存在，主要用于保存一些应用共享的数据和服务，和数据源对象定义、事务管理处理器服务定义等等。

➤ ChannelCtx

EMP平台是一个多渠道整合支持的应用平台，在一个项目中可以提供多个渠道的访问处理。在跟结点之下，平台会根据应用的渠道属性创建相应的渠道结点对象，用于存放本渠道内可共享的数据及应用服务对象。

➤ SessionCtx

一个操作员登录到系统后，会有一个会话数据，该会话数据会一直存在直到该操作员退出系统。在EMP平台中采用一个session类型的context进行用户的会话数据保存。在应用过程中，用户可以随时访问他自己的session会话数据，但不能访问别的用户的session数据。

当一个session被创建时，sessionContext 将根据当前访问的应用信息将session会话对象连接到对应的应用结点上，以使用户能够共享应用信息数据和服务。

➤ TransferCtx

一支具体交易所拥有的资源结点，该结点在该交易开始执行和初始化时产生一个context

对象并将该对象连接到操作员对象的sessionCtx对象下，在该交易执行完成后，将从context结点链中除去并销毁。

2.3.3.1. 练习三：掌握对 Context 的操作 API

Context对象是EMP运行时的资源管理者，所有的数据元素、格式化处理对象和服务对象均是通过Context对象来获取的。

在实际运行中，通过外部化配置文件的方式将Context对象注入到运行时，可以通过flow对象来获得Context对象，运行时外部配置文件已经将相关资源挂接到对应的Context节点中，一般不需要也不推荐采用编码方式来创建和改动Context对象的信息。在这里我们通过手工编码的方式来了解Context对象的操作API，我们在扩展EMP应用的Action（操作逻辑单元）时，打交道最多的就是Context对象，只有通过它才能拿到我们所需要的资源对象。

建立一个新的srapbook页面，输入如下代码并运行：

```
//create a context
Context ctx=new Context();
ctx.setId("testCtx");
//set data Element
KeyedCollection kcoll=new KeyedCollection();
kcoll.setId("AccountInfo");
DataField field=new DataField("accountNo");
kcoll.addDataField(field);
DataField field1=new DataField();
field1.setId("accountName");
kcoll.addDataField(field1);
ctx.setDataElement(kcoll);
//add Service
JDBCDataSource dataSource=new JDBCDataSource();
dataSource.setUserName("testJDBC");
ctx.addService("testSrv",dataSource );
//System.out
System.out.println(ctx.getDataElement("accountNo").toString());
JDBCDataSource
jdbcSource=(JDBCDataSource) ctx.getService("testSrv");
System.out.println(jdbcSource.getUserName());

//set parent ctx
Context parent=new Context();
parent.setId("parent");
KeyedCollection kcoll1=new KeyedCollection();
kcoll1.setId("sessionInfo");
DataField field2=new DataField("userId");
field2.setValue("testEMP");
kcoll1.addDataField(field2);
```

```
ctx.chainedTo(parent);  
//从子节点获取父节点的资源对象  
System.out.println(ctx.getDataValue("userId"));
```

上面的例子描述了如何从context对象中获取数据对象、服务对象，如果将一个Context连接到一个父节点Context上，子节点自动从父节点中获取资源对象的操作。

在实际运用中，通过外部化配置文件的方式来注入资源对象，如下所示是一个Context对象外部化的配置内容：

```
<context id="rootCtx" type="root">  
    <refKColl refId="rootKColl"/>  
    <refService alias="transactionManager" type="service"  
refId="transactionManager"/>  
    <refService alias="tableService" type="service"  
refId="tableService"/>  
    <refService alias="sqlService" type="service"  
refId="sqlService"/>  
</context>
```

2.3.4. 服务组件：Service

- 与流程无关，通过context获得
- 提供大量内建的Service：
 - 金融外设
 - 电子日志
 -
 - 数据库访问服务
- 通过XML文件定义
- 用户可扩展
- 一个服务对象定义的例子

```
<JDBCConnectionPool id="JDBCPool">

    <JDBCResource driverName="oracle.jdbc.driver.OracleDriver"    dbUserName="ecc"
    dbURL="jdbc:oracle:thin:@localhost:1521:EMPDEMO"    maxConnection="5"
    resourceID="ORACLEJDBC"    dbPassword="eccc" jndiURI=""/>

</JDBCConnectionPool>
```

这是JDBC的连接池Service，id是JDBCPool，配置参数如下：

resourceID="ORACLEJDBC"定义当前resource的ID，因为可能存在多个数据源。
driverName="oracle.jdbc.driver.OracleDriver"配置数据库的驱动名，为必须的参数
dbURL="jdbc:oracle:thin:@localhost:1521:EMPDEMO"配置数据库资源路径，为必须项
maxConnection="5"表示数据库的最大连接数为5，dbUserName和dbPassword分别配置数据库连接的用户名和密码。

特别提示：EMP的service对象可以是任意的POJO对象类。只要它是一个满足JavaBean规范的一个Java实现即可。（JavaBean规范的简单判断就是：通过标准的属性设置/获取方法进行属性构造的类）。我们通过外部化配置文件将Service注入到Context对象中，并在调用中通过Context获得。

在配置文件中，我们如上定义一个Service对象，然后在相应的Context中进行引用，如下：

```
<context id="rootCtx" type="root">
    <refKColl refId="rootKColl"/>
    <refService alias="transactionManager" type="service"
refId="transactionManager"/>
    <refService alias="tableService" type="service"
refId="tableService"/>
    <refService alias="sqlService" type="service"
refId="sqlService"/>
</context>
```

Service的具体定义在其他地方，在Context定义中进行Service定义的引用即可。

在实际使用中，通过Context对象获取的Service对象返回的是一个Object对象，我们必须手工将它转换为它的实际对象，才能调用其相应的方法，如下所示：

```
//通过名称获得Service对象，一个POJO对象实例，并强制类型转换为它本身所代表的类对象
JBCDDataSource
jdbcSource=(JBCDDataSource) ctx.getService("testSrv");
//然后可以直接调用该服务对象的方法
```

```
System.out.println(jdbcSource.getUserName());
```

2.3.5. 格式化处理：Format

EMP平台提供的一个独立的、可灵活扩展的组件，实现对交易数据根据不同的通讯格式进行格式化。

EMP平台为确保通信服务适用于不同的类型的通信，通过参数定制的方式灵活的实现了不同系统的数据报文格式处理。EMP平台提供的通用报文格式处理包括XML数据报文格式，String格式，8583报文格式。

一个格式化处理的例子：

```
<fmtDef id="Format">
```

```
<record>
```

```
<fString dataName="data1"/><delim delimChar="#"/>
```

```
<fString dataName="data2"/><delim delimChar="#"/>
```

```
<fString dataName="data3"/><delim delimChar="#"/>
```

```
</record>
```

```
</fmtDef>
```

dataName="data1"中，data1是data中定义的数据

delimChar="#"表示，在这个数据后面用"#"作分隔符，如果data1="emp"，data2="yucheng"，data3="tech"，则格式化后得到的数据为：emp#yucheng#tech#。

具体的格式化定义处理将在后续的开发课程中详细讲解。

2.3.6. 交易流程：Flow

➤ 流程定义：flow

一个交易（Operation）由交易流程和交易资源（交易数据和服务Service）两大部分组成。opflow抽取了交易的处理流程，把交易流程和交易资源的定义分离，以便于更好的交易开发和维护管理。

```
<flow id="testServerOp">
  <action id="Initialize0" >
    <transition condition="$retValue='0'" desc="EJSequenceOpStep0"/>
  </action>
  <action id="EJSequenceOpStep0">
    <transition condition="$retValue='0'" desc="EJOperationStep2"/>
    <transition condition="$record>'0'" desc="EJUpdateOpStep0"/>
  </action>
  <action id="EJOperationStep2" OP="insert"/>
  <action id="EJUpdateOpStep0" />
</flow>
```

➤ 操作步骤：action

一个flow有多个action组合而成。

flow包含多个操作步骤：action，这些action是一些可被重复使用的子操作（如写交易结果到日志）。

交易处理是按照业务逻辑流程来组装交易步骤来实现的。按文件定义的流程来执行的

➤ 跳转步骤：transition

看下面的跳转定义例子：<transition condition="\$retValue='0'" desc="EJOperationStep2"/>

我们在一个action中可以定义多个跳转步骤，每个跳转步骤都有自己的condition跳转条

件，跳转条件采用公式脚本方式进行配置，采用的数据可以为本交易流程所能取得的所有数据。

2.3.6.1. 练习四：扩展一个操作步骤（业务逻辑单元）

操作步骤继承于com.ecc.emp.flow.EMPAction 类。

我们自己编写的Action需要继承于该类，并实现该抽象类的虚方法：

```
public abstract String execute(Context context)throws EMPEException;
```

该方法返回一个String，我们在实际扩展中一般通过设置不同返回值来判断该交易步骤地执行结果，如返回‘0’时，执行正确，其他值，为执行出现异常。

在Action中传入了参数为Context对象，我们在实际操作中，通过该Context对象来获得所有的资源对象。以下是一个扩展的Action实现。

```
package com.ecc.emp.action;

import com.ecc.emp.core.Context;
import com.ecc.emp.core.EMPEException;
import com.ecc.emp.data.IndexedCollection;
import com.ecc.emp.flow.EMPAction;

/**
 * 判断一个iColl中存在的数据元素个数（-1则表示无）。
 * <b>配置示例:
 *
 *      <action      id="haveElementsInIColl"      iCollName="a      iColl
 *      "implClass="com.ecc.emp.action.HaveElementsInICollAction">
 *  *参数说明: iCollName: IndexedCollection数据集合的名称 <br>
 *
 *
 * <b>返回状态:  </b> <br>
 * 0 集合中不存在元素, -1 异常,其他: 集合中存在数据元素, 且onNDo的N为具体的元素个数
 * <br>
 *
 */
```

```
public class HaveElementsInICollAction extends EMPAction {
    String iCollName = null;
    public HaveElementsInICollAction() {
        super();
    }

    /*
     * 判断iColl中存在的数据元素个数（-1表示无）
     *
     *
     */
    public String execute(Context context) throws EMPEException {

        try {
            return String.valueOf(((IndexedCollection) context
                .getDataElement(iCollName)).size());
        } catch (Exception e) {
            return "-1";
        }
    }

    /*
     * 获得各个参数的值
     */
    public void setICollName(String name) {
        this.iCollName = name;
    }
}
```

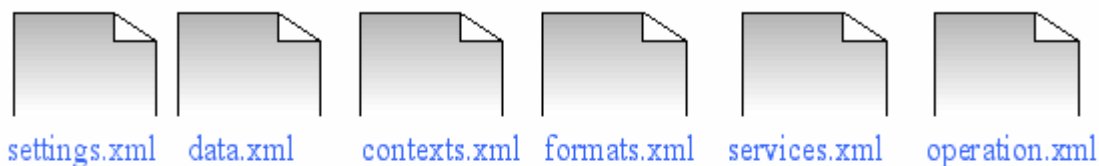
如上，我们实现了一个检查指定的iColl对象是否存在列表元素的对象的EMPAction，其中定义了属性iCollName，需要我们在配置的时候指定所需要检查的iColl的名称，我们提供了标准的JavaBean的属性设置方法：setICollName（String name），那么就可以在外部化配置文件中通过配置属性iCollName=“aIColl”的方式将该属性值注入到Action实例中。

2.3.7. 业务逻辑处理相关配置文件

2.3.7.1. 配置文件概述

如下图，是 EMP 平台在描述业务逻辑构件时所需要用到各种配置文件种类

○ 公共配置文件种类



○ 具体业务实现配置文件



首先是六份公共的配置文件，用来描述整个应用系统的公用的属性和数据。针对具体的业务处理逻辑构件，采用独立的BizLogic.xml 文件进行描述。

➤ Settings.xml

完成EMP平台的类型数据的参数定义和系统参数定义。以下五个文件中所应用的数据类型，都需要在dse.ini中进行预定义处理。

➤ datas.xml

定义公用的交易数据。

➤ context.xml

定义公用的上下文关系。

➤ formats.xml

定义公用的交易数据交互或通行处理中必须的数据报文格式。

➤ **services.xml:**

定义交易处理流程中，需要调度的服务Service的外部参数化配置，在这里定义公用的服务组件对象。

➤ **operation.xml:**

定义公用的交易处理流程——根据具体业务功能的处理逻辑、组合EMP平台提供的交易步骤，形成交易流程外部参数化配置。

2.3.7.2. 公共参数配置文件：Settings

Settings.xml用于描述整个应用所需要的系统参数、目录路径及关键类对象的类映射关系。

我们可以根据需要在settings.xml文件中进行扩展的数据定义。

Settings针对的是公共参数的定义，而不建议定义某个具体业务应用的参数。

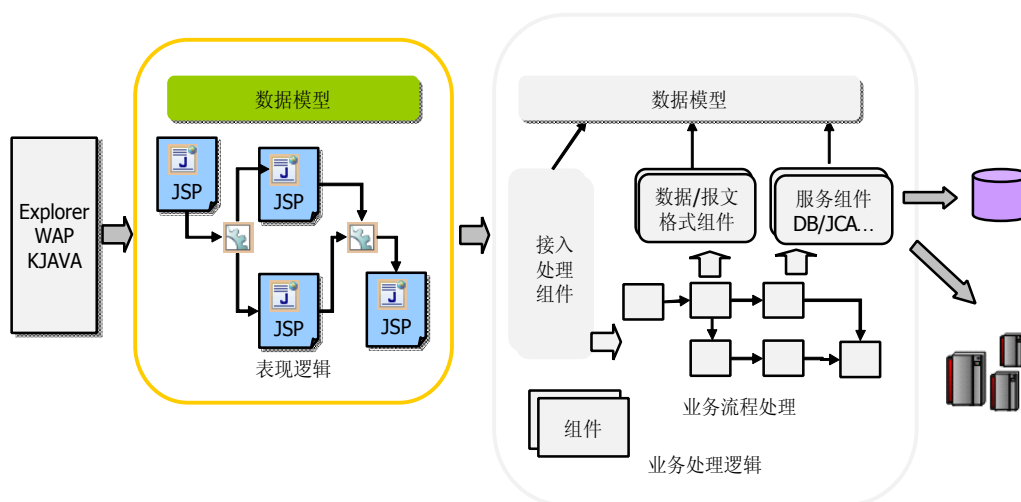
三个关键系统参数组定义：

Files：用于定义系统内配置文件的名称。

Paths：用于定义系统内配置文件的路径。

Tags：用于定义应用所需要用到的关键类对象的名称/类映射。

2.4. EMP web2.0 处理框架



表现逻辑处理容器是EMP平台为应用系统的表现逻辑流程处理而提供的引擎，它解释执行在应用系统开发过程中定义好的表现逻辑流程，在表现逻辑中调用业务处理逻辑完成一定的业务功能，根据业务逻辑的处理结果控制页面流转，在调用业务逻辑结束后，展现逻辑引擎会把数据传送到展现页面上，在页面中可以使用丰富的标签库展现数据。

本章节内容将在后续培训课程中详细介绍。

2.5. EMP 运行环境

2.5.1. 服务器端运行环境

J2EE应用服务器或支持servlet的web应用服务器。

数据库服务器支持：可支持各种数据库软件版本。

2.5.2. 客户端运行环境

基于浏览器的应用环境。浏览器要求：

支持javascript 1.2及以上版本和ajax技术的浏览器版本。对操作系统无要求。

3. EMP IDE 开发实践

3.1.EMP 开发支持

3.1.1. IDE 概述

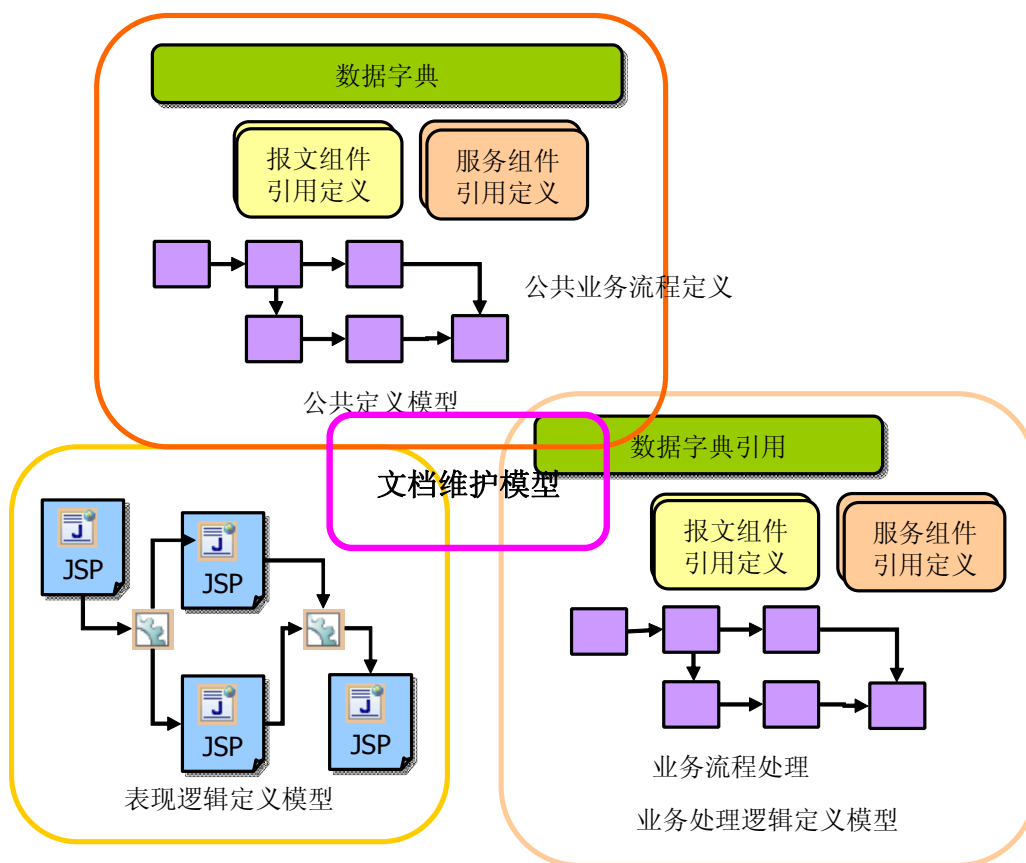
EMP集成开发平台通过对Web应用基础平台EMP的应用模型的二次抽象，通过对这个模型的维护的方式，实现基于EMP平台的具体应用系统的开发与维护，为基于EMP平台的应用系统的开发提供可视化的模型维护工具。

EMP集成开发平台是基于目前流行的J2EE开发平台Eclipse的插件机制实现的一套EMP应用开发平台。同时充分考虑EMP平台的组件式开发的开放结构，开发平台也是一个开放结构的开发平台，在应用中可以为用户提供充分的扩展组件的IDE开发支持。并且支持用户对IDE的扩展。

透过开发平台IDE可以实现：

- 基于 EMP 的 J2EE 应用系统生命周期内的应用模型维护
- EMP 组件维护
- EMP 运行时代码的导出与生成
- 详细设计文档的生成
- 小组协同与版本管理
- JAVA 代码的开发与调试
- 即时的代码运行与调试

3.1.2. IDE 的定义模型



如上图所示，为IDE的应用模型，IDE将EMP应用系统抽象为：公共定义模型，业务处理逻辑模型和表现逻辑定义模型三大部分。通过公共定义模型将整个应用系统的公共部分抽取出来集中定义，具体的逻辑定义模型只是对这些公共定义模型的引用，避免公共部分的修改影响到多处应用逻辑的修改。同时在每个模型中加入文档维护模型，实现在应用系统模型维护过程中维护整个系统的文档。

3.1.3. IDE 的基本功能

➤ EMP 组件维护

IDE提供了EMP组件的可视化维护，让用户维护EMP组件，这些组件包括EMP提供的组件，也包括了客户的扩展组件，通过组件维护，可以得到EMP组件的详细使用信息。

同时IDE也提供了创建新组建的向导，引导客户一步一步的完成扩展组件的创建。

➤ 公共资源维护

实现公共定义模型的可视维护，这些功能包括：

数据字典的定义，数据类型与校验定义，应用系统公共配置信息的维护，公共处理流程的可视化定义，渠道的管理与维护，公共资源访问模块的设计与维护，报文格式处理的可视化设计与维护。

➤ 业务处理逻辑可视开发

提供可视化的开发工具，实现业务处理逻辑的定义，实现包括：业务处理逻辑对数据字典的引用，拖拉式的组件组装工具实现业务处理流程。可视化的后台资源访问定义，包括报文定义以及资源访问。

➤ 表现逻辑可视开发

提供可视化开发工具，完成表现逻辑可视化开发，这些包括：表现逻辑的布局定义，动态菜单的维护与开发，表现逻辑页面的可视开发，实现表现逻辑组件的可视化组装，以及表现逻辑页面的流转逻辑。

➤ 文档维护与生成

IDE通过在模型设定上，提供文档的维护功能，让使用者在定义业务处理模型的过程中维护每个具体模型处理的文档，实现在开发过程中维护应用系统的详细设计文档，最终提供工具透过Word模版生成用户需要的详细设计文档。实现设计文档与应用系统开发的同步，避免在开发过程中对应用系统得修改而没及时修改设计文档的问题。

3.2. 课程所需资源

编号	资源名称	来源	备注
1.	EMP IDE 插件	教程光盘\EMP IDE	
2.	Eclipse	教程光盘\工具	3.2 版本
3.	Tomcat 服务器	教程光盘\工具	5.5 版本
4.	JDK	教程光盘\工具	1.5 版本
5.	Deby 数据库	教程光盘\资源	
6.	Dummy host 虚拟主机	教程光盘\资源	
7.	Client	教程光盘\资源	多渠道测试客户端
8.	课程相关文件	教程光盘\资源	样式，菜单树，数据字典

9.	文档	教程光盘\教程	培训教程
10.	EMP Monitor 项目	搅成光盘 资源	用于监控系统

3.3. 课程 3：第一个 EMP 项目

课程名称	建立开发环境	课程时间	
课程目标	一、EMP 开发环境的搭建 二、熟悉相关工具的安装		
准备工作	一、Eclipse 3.2 以上版本 二、EMP IDE 插件包 三、TOMCAT5.5 服务器		
备注	为了保证教学课程的顺利进行，请使用教程光盘提供的软件版本		

3.4. 练习一：建立 EMP IDE 开发环境

3.4.1.1. JDK 的安装

双击jdk1.5.0.02.exe进行安装。按照安装向导的提示一步步操作，依次安装好JDK和JRE。本教程以安装到c:\JDK 1.5\目录下为例。

3.4.1.2. Eclipse 的安装

Eclipse属于绿色软件，不需要运行安装程序，也不需要Windows注册表中写入信息。只需要将Eclipse压缩包解压就可以运行了。本教程以解压到d:\eclipse为例。

双击d:\eclipse\eclipse.exe启动eclipse。第一次运行时会弹出如下对话框，要求设定工作空间的路径。本教程以e:\workspace作为工作空间目录。确定后开始Eclipse载入过程，完毕后会显示欢迎画面，表示Eclipse已经成功安装。

3.4.1.3. EMP IDE 插件的安装

EMP IDE作为Eclipse插件，采用标准的安装插件的方法。将以下文档压缩包解压到D:\eclipse\plugins\目录下：

- `com.ecc.ide.module.core_ X.X.X.vYYYYMMDD`
- `com.ecc.ide_ X.X.X.vYYYYMMDD`

若看到Eclipse的工具栏中出现了图标，表示IDE插件安装成功。

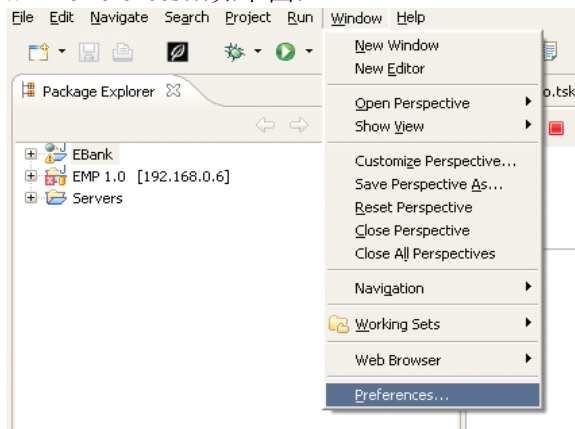
注：安装不成功的话，检查一下目录解压时，例如是否出现了重复目录，即 `com.ecc.ide.help_ X.X.X.vYYYYMMDD` 目录下，还有一个 `com.ecc.ide.help_ X.X.X.vYYYYMMDD`。

3.4.1.4. Tomcat 的安装

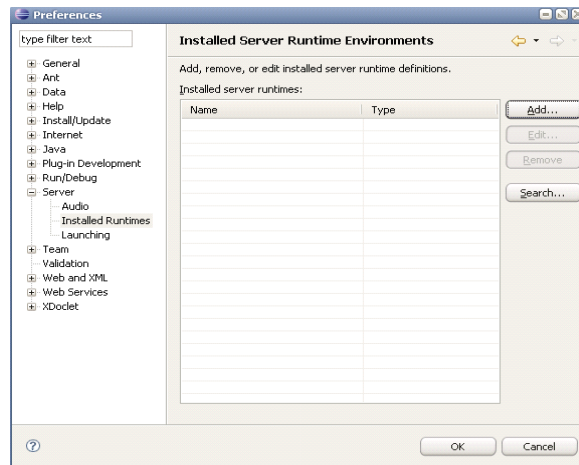
Tomcat的安装也是将压缩文件展开就可以了。本教程以解压到`c:\tomcat_5.5\`为例。

3.4.1.5. Tomcat 在 Eclipse 上的配置

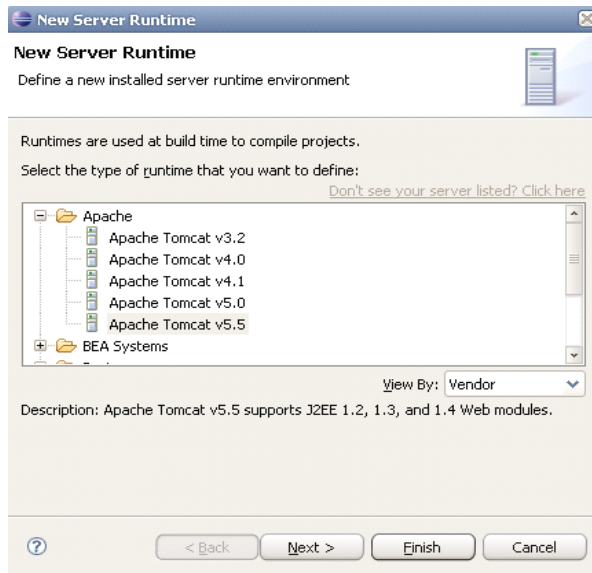
打开 Eclipse 里 window→Perferences...如下图：



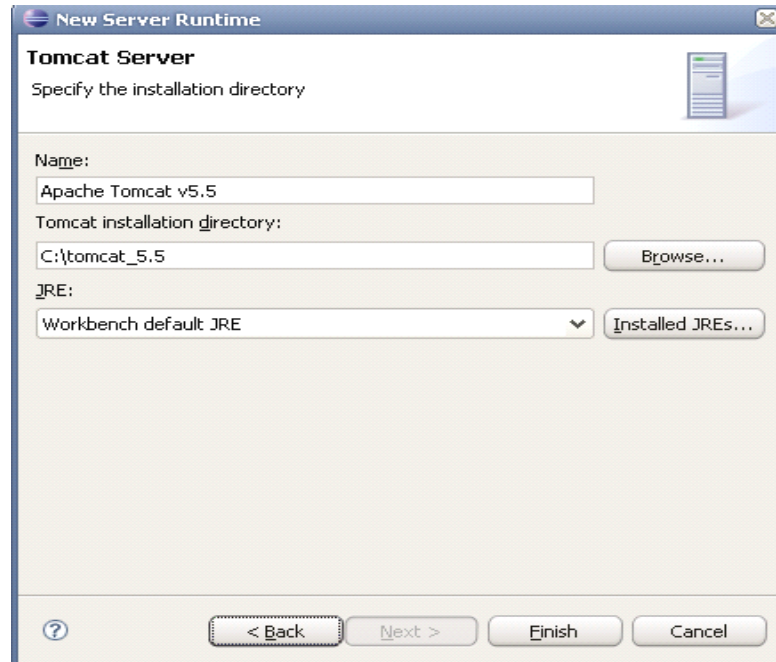
会弹出如下对话框，选择 Server→Installed Runtimes 如下图：



点击Add...按钮，将弹出如下对话框，以View By 为Vendor为例选择Apache→Apache Tomcatv5.5:



点击Next>按钮，将弹出如下对话框，填入Tomcat5.5的安装路径，如下图:



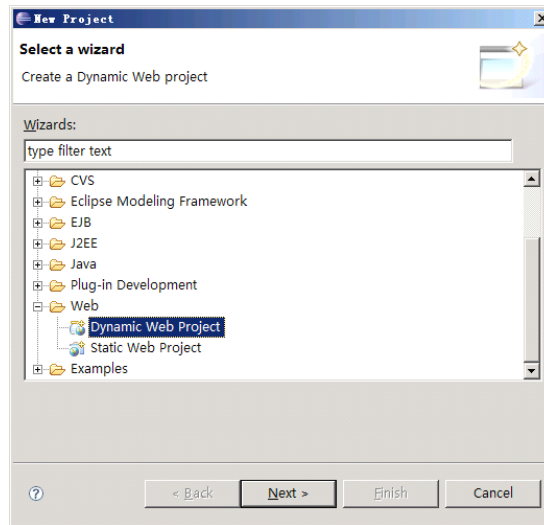
点击完成，到此Tomcat5.5配置完毕。

3.5.练习二：建立第一个 EMP 项目

3.5.1.1. 建立第一个 EMP 项目

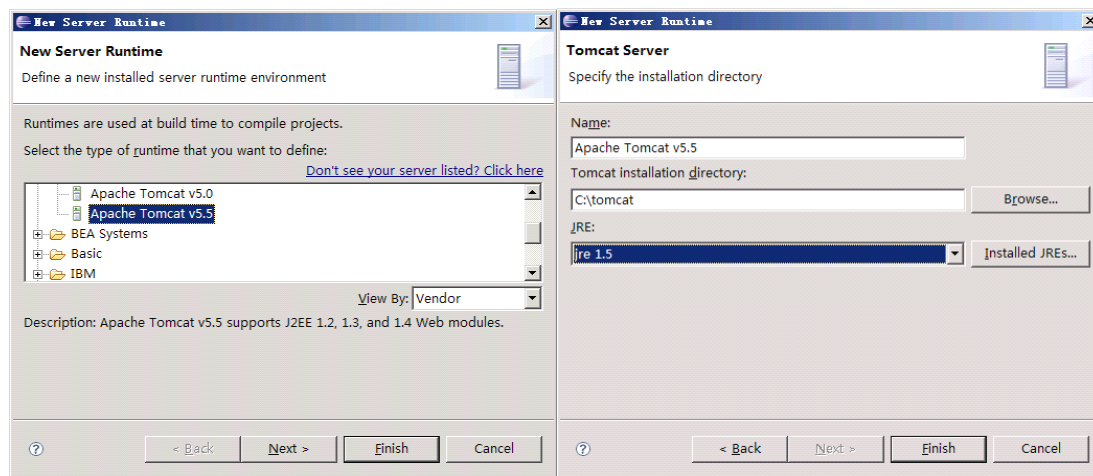
➤ 新建一个动态的Web项目

在Eclipse菜单中选择File-New-Project，在弹出的向导中选择Dynamic Web Project（动态Web项目）。

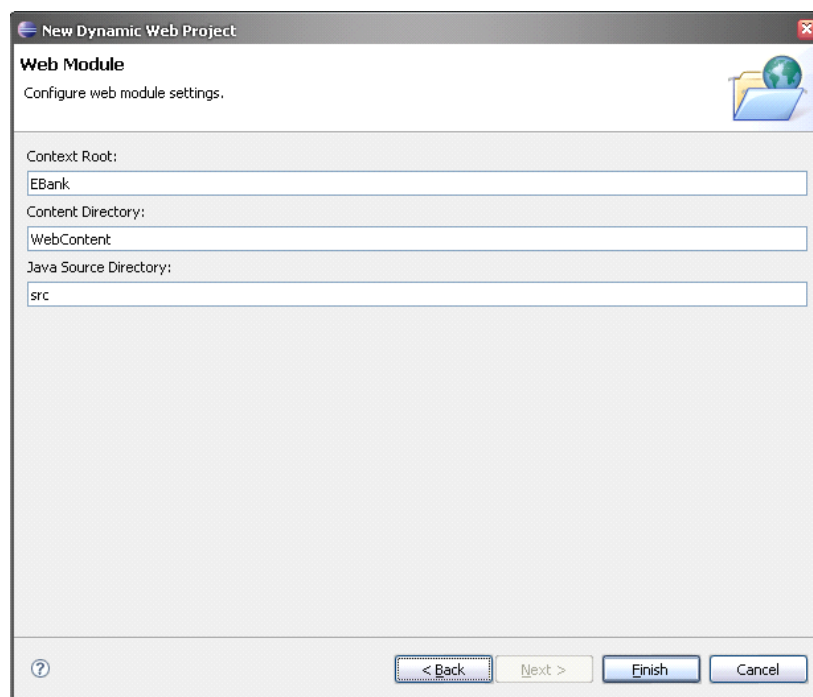


在向导下一页进行如下设置：项目名为**EBank**，使用默认的项目路径。如果未在Eclipse中配置Tomcat，可以按如下方法配置：

点击Target Runtime右边的New... 按钮，会弹出一个新的向导，在第一页选择Apache Tomcat v5.5，第二页设置好服务器名、tomcat安装路径和JRE，如下图所示：



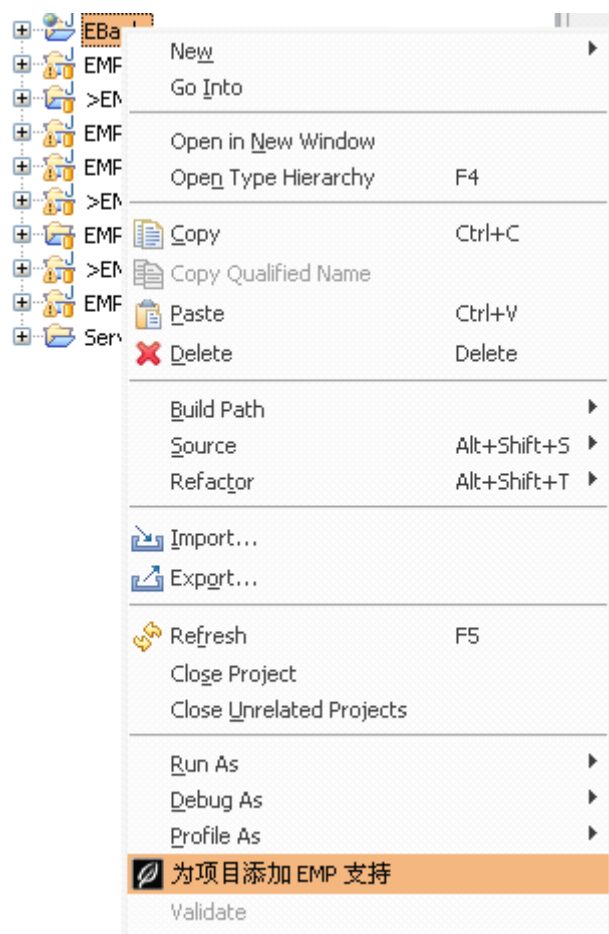
点击Finish回到刚才的新建项目向导，点击两次Next（中间的一页按照默认设置即可）。在最后一页设置Web项目的路径，如下：



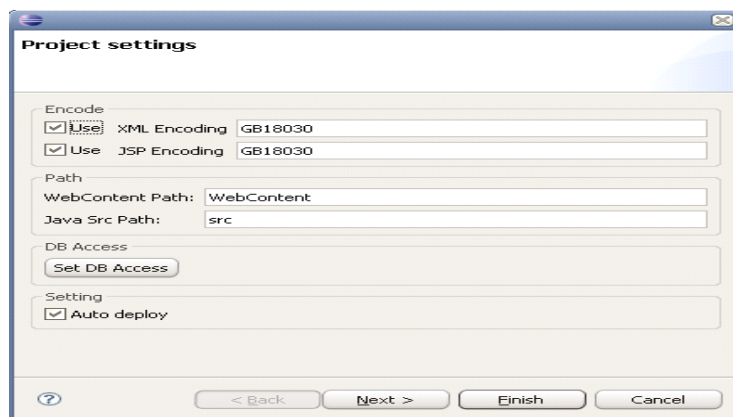
点击Finish，动态Web项目就创建好了。

➤ 为项目添加EMP支持

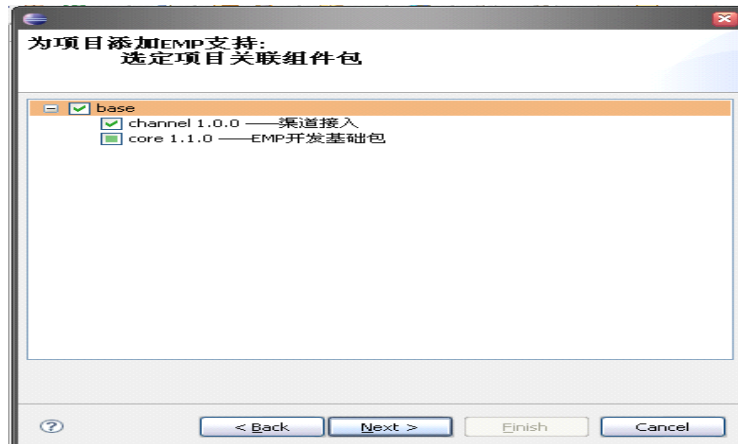
在Package Explorer中右键点击创建好的HelloWorld项目，在弹出菜单中选择EMP IDE-为项目添加EMP支持。如下图：



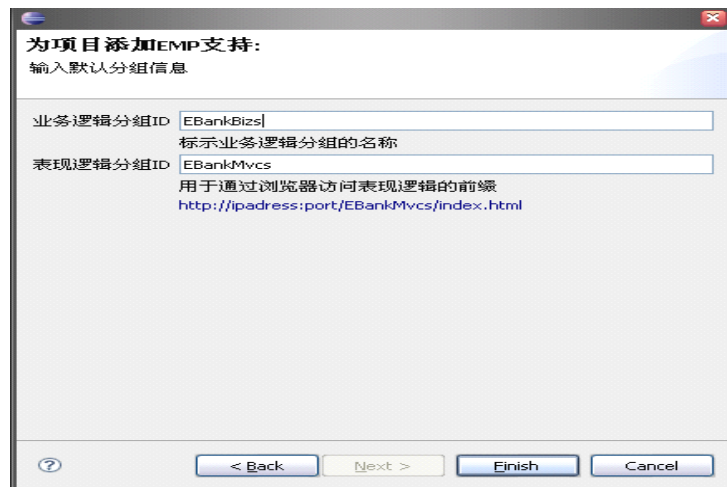
选择后弹出如下向导，在第一页进行项目全局设置：XML和JSP的文件编码按照默认的编码即可；路径要和Web项目的路径保持一致；数据库连接可以暂时略过，以后再进行设置；为了开发方便，最好选中自动编译。点击Next进入下一页。



在这一页中选中需要导入的EMP模块。从头开发一个EMP项目只需导入默认选中的core 1.1 开发基础包即可。



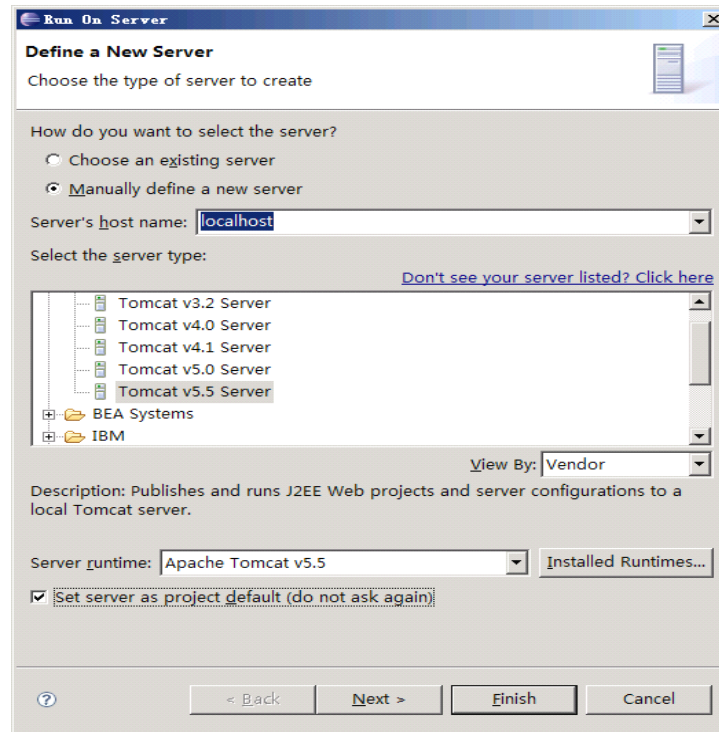
点击下一步，设置该项目的表现逻辑分组和业务逻辑分组的名称，如下所示：



点击Finish后，EMP开发的所需文件就会被导入项目中，同时在控制台中会显示导入信息。

➤ 试运行项目


在项目上点击右键，选择Run As...-Run on Server，在弹出向导中设置Tomcat服务器。若之前已经建好了服务器，选择Choose an existing server（选择现存服务器）一项即可；否则需要选择Manually define a new server（手工定义新服务器），并按照下图进行设置。



使用浏览器访问`http://localhost:8080/EBank/`，看到缺省的演示页面，表示服务器配置运行成功。



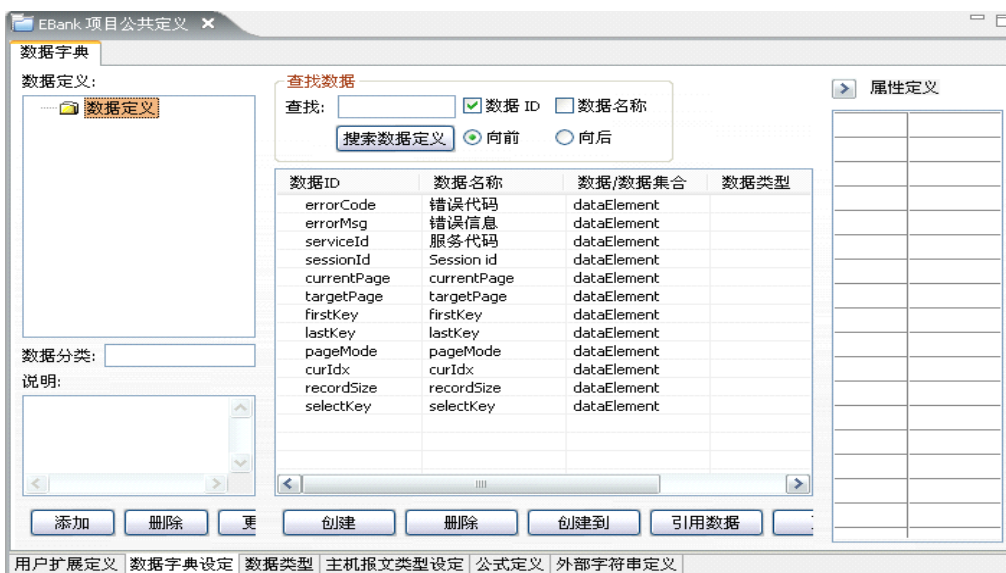
3.5.1.2. 掌握数据字典的定义

成功为新建的项目添加了EMP支持后，点击工具栏上的按钮，显示EMP Explorer(出现在Eclipse操作界面的右下方)。这个视图是IDE的开发视图，屏蔽了与开发不相关的文件，并提供

IDE开发的各种操作入口。将EMP Explorer视图拖拽到左方的区域，如下图：

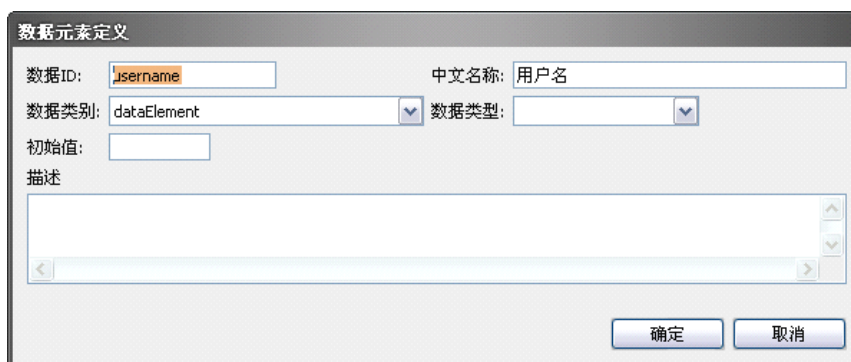


点击左边菜单树的“项目公共定义”节点，Eclipse操作界面上会出现一个“Ebank项目公共定义”的编辑器，在项目设定编辑器上选中数据字典设定，如下图：



注：在EMP项目里所用到的所有数据都是从数据字典应用的，因此EBank项目中所用到的一个数据用户名（username）需要先在这里定义。

点击设定编辑器的创建按钮，会弹出一个数据元素定义对话框，如下图：

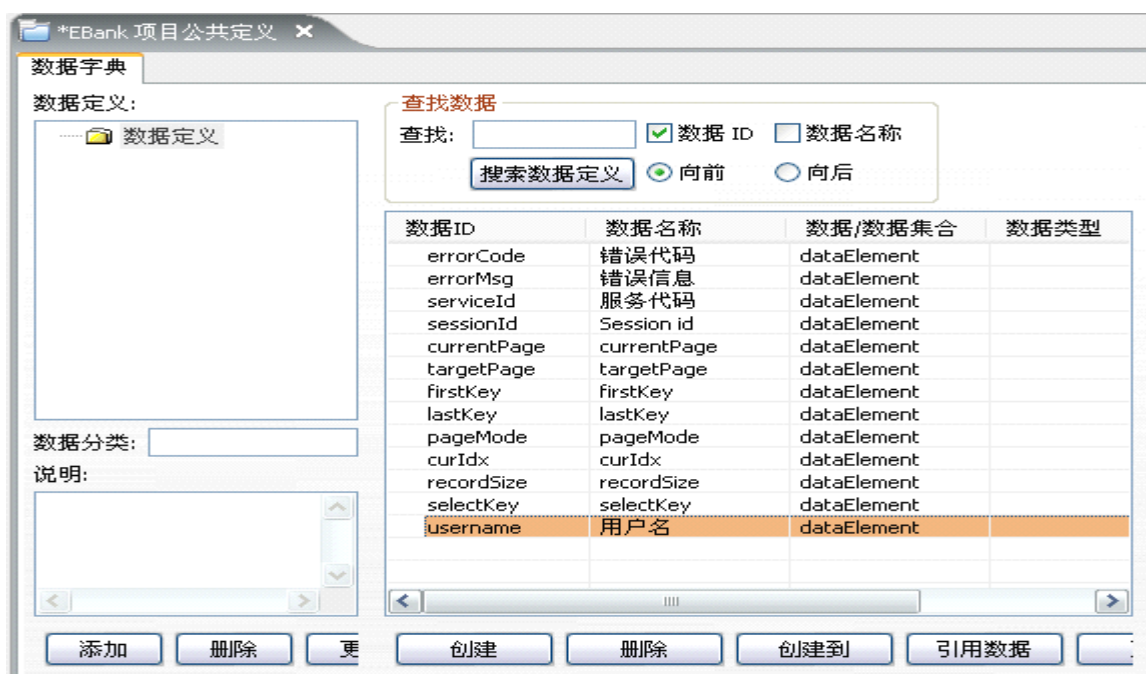


数据元素定义对话框，包含以下字段：

- 数据ID:
- 中文名称:
- 数据类别:
- 数据类型:
- 初始值:
- 描述:

底部有“确定”和“取消”按钮。

数据ID为：username, 中文名称为：用户名，其他设置默认，然后点击确定，查看数据字典，会发现数据字典上多了一项username的数据项，如下图：



数据字典界面，显示数据定义列表。左侧为“数据定义”树，右侧为“查找数据”搜索框和“数据字典”列表。

查找数据：

☒ 数据 ID
 ☐ 数据名称

☒ 向前
 ☐ 向后

数据ID	数据名称	数据/数据集合	数据类型
errorCode	错误代码	dataElement	
errorMsg	错误信息	dataElement	
serviceId	服务代码	dataElement	
sessionId	Session id	dataElement	
currentPage	currentPage	dataElement	
targetPage	targetPage	dataElement	
firstKey	firstKey	dataElement	
lastKey	lastKey	dataElement	
pageMode	pageMode	dataElement	
curIdx	curIdx	dataElement	
recordSize	recordSize	dataElement	
selectKey	selectKey	dataElement	
username	用户名	dataElement	

底部有“添加”、“删除”、“更改”、“创建”、“删除”、“创建到”、“引用数据”等按钮。

至此，username的数据项，已经创立完成，依照上述方法，往数据字典中添加userid，password，email和phonenum。最后保存项目设定编辑器即可。

以上介绍的是数据字典定义的IDE的用法，其实数据字典对应配置文件designFiles/commons/dataDict.xml。IDE对数据字典的增删改，即是对此dataDict.xml的修改。

3.6. 课程 4：第一个开发实践：客户注册

课程名称	EMP 的第一个实例客户注册	课程时间	
------	----------------	------	--

课程目标	一、熟悉掌握 IDE 的基本机制 二、熟悉掌握 EMP 的基本机制 三、掌握 EMP 开发平台的基本应用
准备工作	完成课程 3

3.6.1. 客户注册的功能分析

用户在浏览一个网站时，如果需要更进一步的使用网站功能往往需要在该网站注册用户资料和登录密码，以便用户进行身份验证和保存用户相关资料信息。

- 首先建立一个用户注册页面，该页面需要填写用户的基本信息和登录密码，用户基本信息包括：用户姓名、用户 Id、用户密码、用户住址、用户电话、用户电子邮件地址等信息。
- 提交该页面，记录用户信息到数据库中，并检查用户名是否已经存在，如果存在则报错。
- 根据注册操作成功/失败信息，显示操作结果页面。

3.6.2. 练习一：实现业务逻辑构件：customerManager

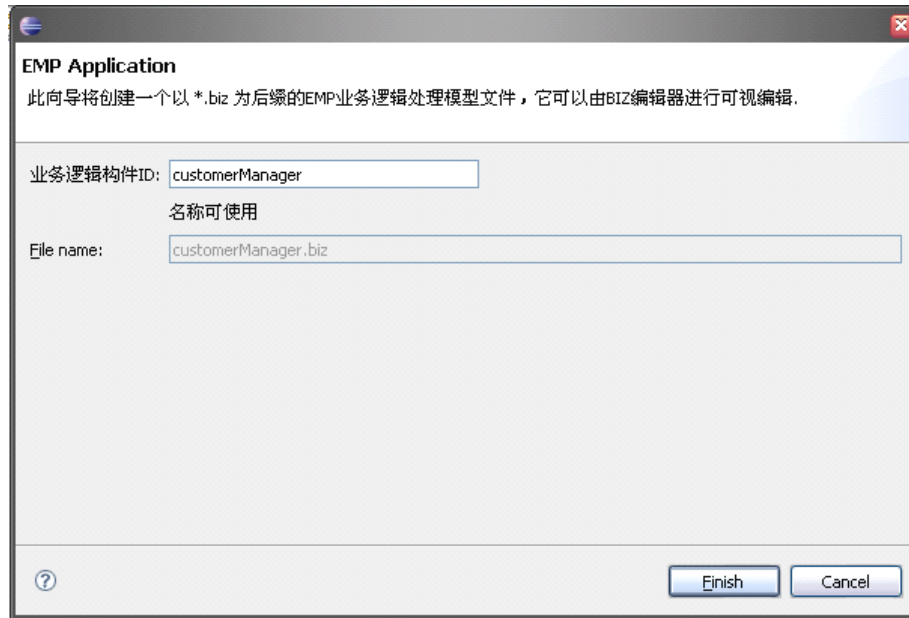
3.6.2.1. 建立业务逻辑构件

EMP的业务逻辑都是建立在业务逻辑分组里面的，因此在EBankBizs分组中，新建业务逻辑构件customerManager.biz，具体操作如下：

- 选中EbankBizs分组，点击右键，选择新建→新增业务逻辑构件

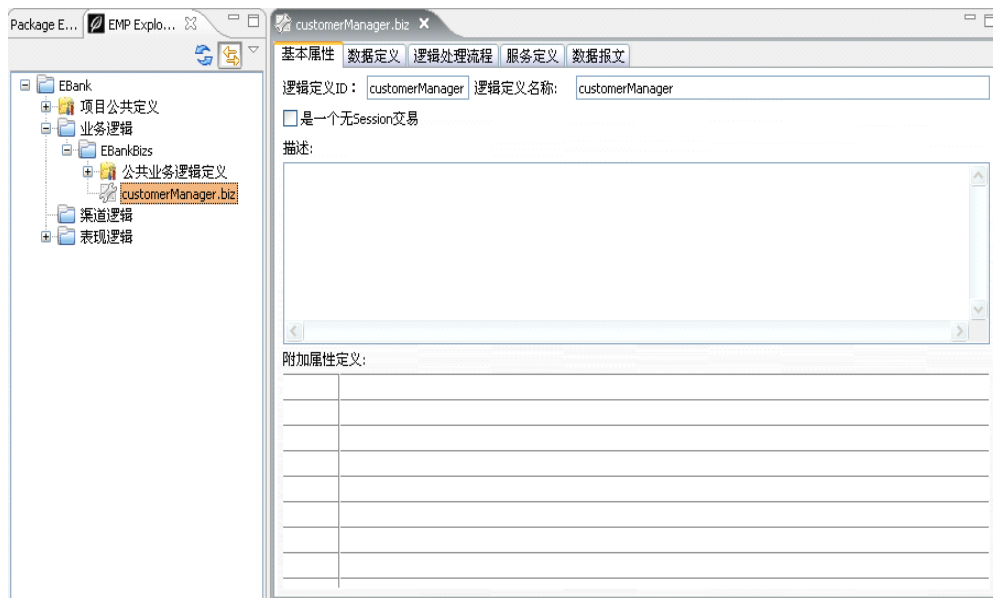


- 点击新增业务逻辑构件后，弹出如下对话框：



在对话框上填写要创建的业务逻辑构件ID为customerManager。

- 点击Finish后，在EMP Explorer上的helloworldbizs分组里会有一个customerManager.biz，且在Eclipse操作界面的右边会弹出一个customerManager.biz的编辑器，如下图：

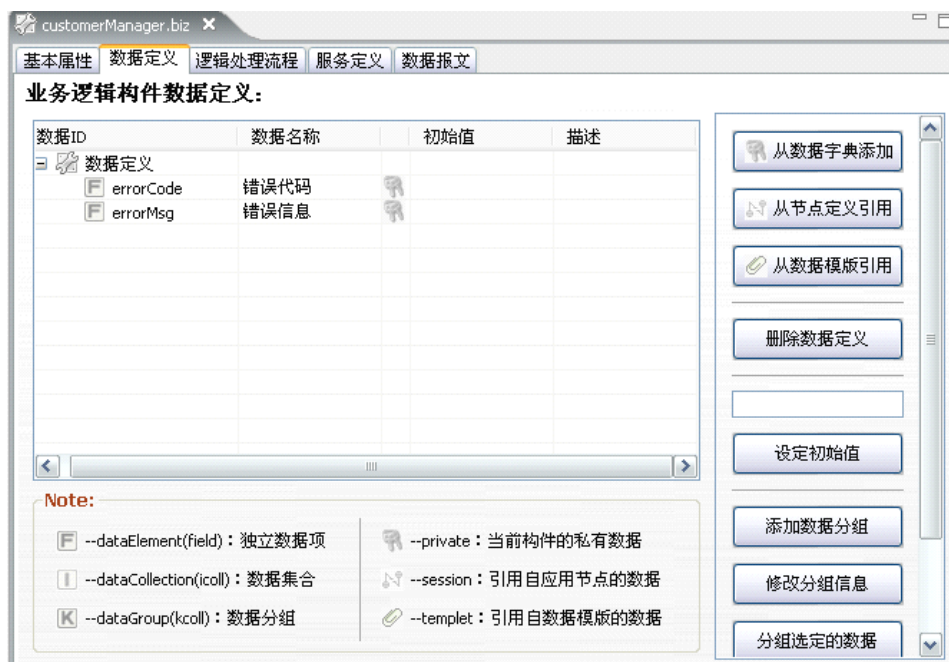


到此业务逻辑构件建立完成。

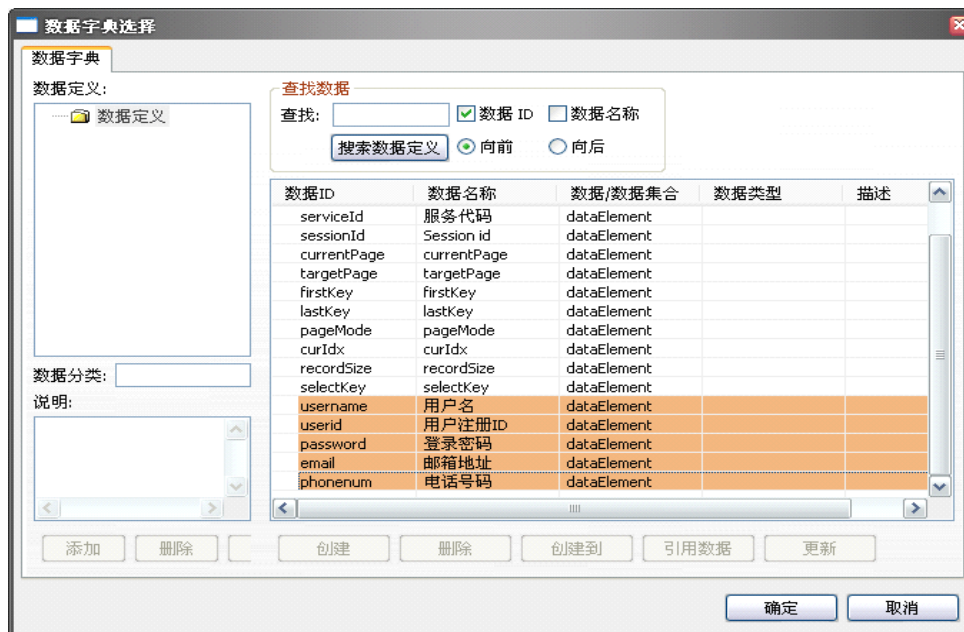
注：在业务逻辑构件编辑器的基本属性中需要选中“是一个无Session交易”，且Context链接到root节点上。

3.6.2.2. 数据字典定义与录入

业务逻辑构件customerManager建立好后，接下来将数据字典中需要用到的数据项，添加到该业务逻辑构件中。点击customerManager业务逻辑构件编辑器的“数据定义”页签，如下所示：



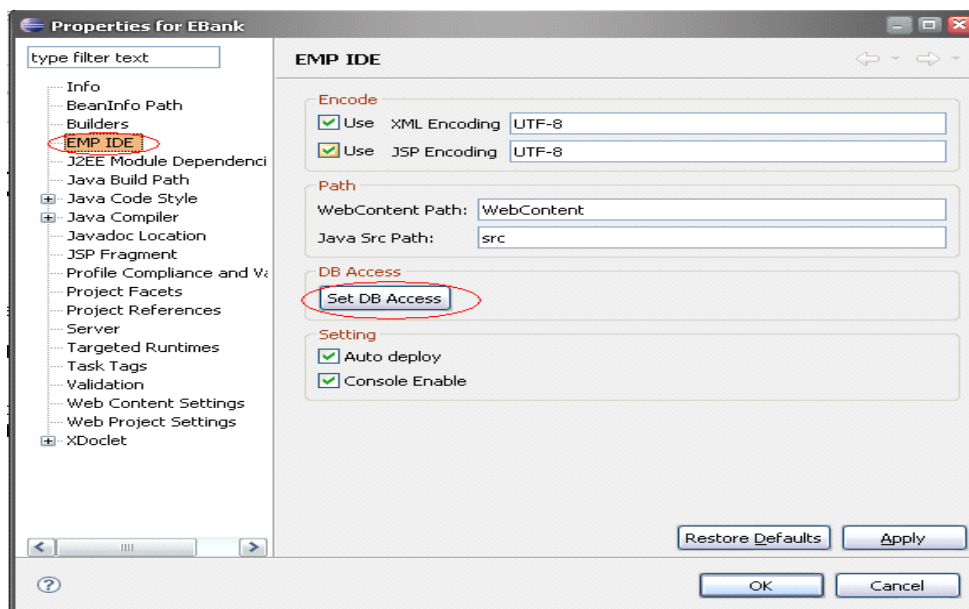
点击“从数据字典添加”按钮，会弹出一个对话框，显示了数据字典中所有的数据项，如下所示：



运用shift和ctrl键，将需要用到的数据项选中，然后点击确定。则数据项就能成功的录入到customerManager.biz的业务逻辑构件中。

3.6.2.3. 实现数据库的访问

用户注册这个业务处理流程是将用户的注册信息保存到数据库中，所以我们在我们使用数据库之前，先要对数据库的访问进行设置。这里我们以derby数据库为例。下面我们先来介绍一下IDE中，如何设置数据库。我们将左边的窗体切换到package视图，然后点击右键->properties，出现如下对话框：

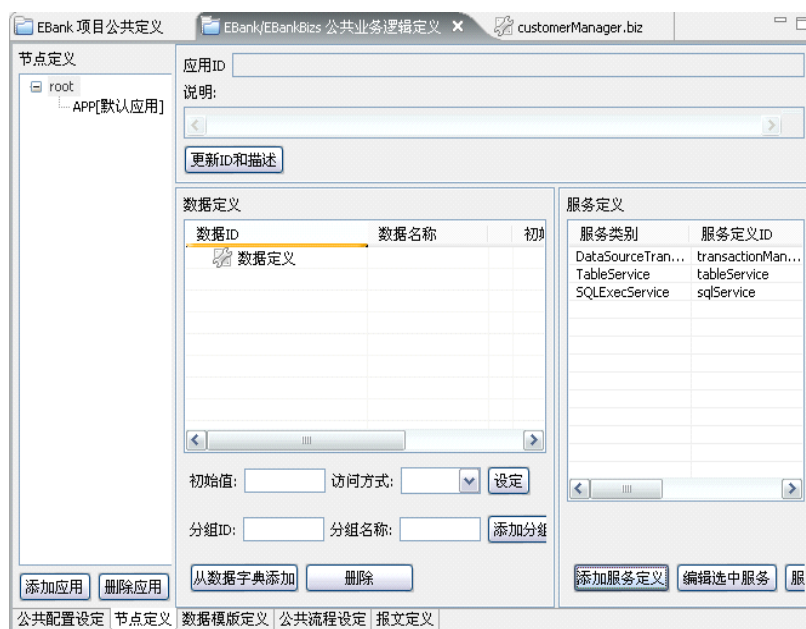


选中EMP IDE，进入EMP IDE的属性设置框，点击“Set DB Access”按钮，弹出数据库设置对话框如下所示：

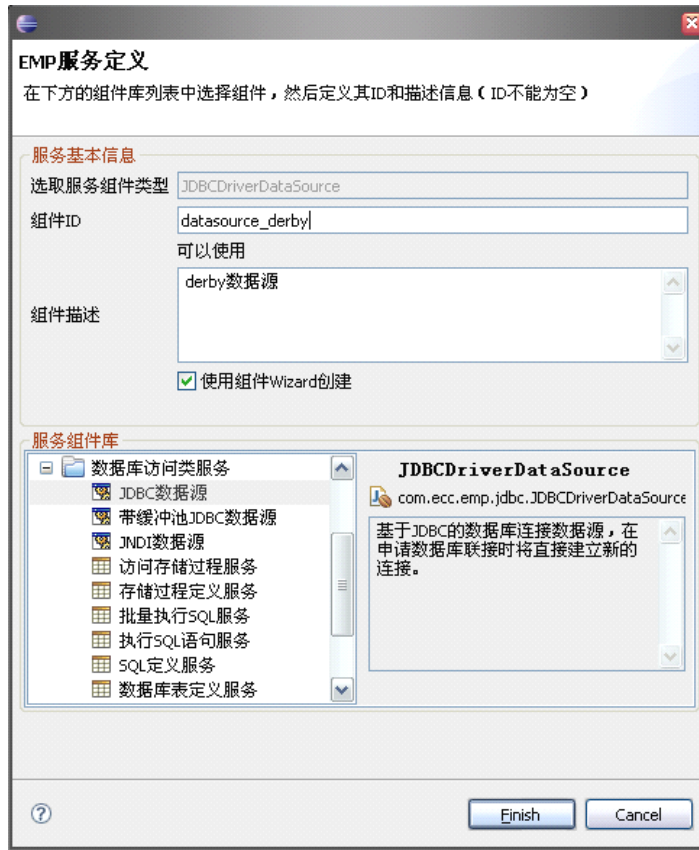


在该对话框中，可以设置连接某个数据库的数据源参数，这里默认的是derby的数据源，可以通过“连接”按钮，测试对数据库的连接是否成功。当配置连接成功后，点击“确定”按钮后，EMP IDE的数据源就算是配置成功了。

EBank应用现在需要往数据库的一张表里插入信息，这里EMP是通过提供两个服务datasource和tabledefine来完成对这一需求的支持的。首先需要有一个数据源服务，该服务比较通用，我们可以放在“公共业务逻辑定义”中完成datasource的定义，打开“公共业务逻辑定义”编辑器，选择“节点定义”页签，如下所示：



选中左边菜单的root节点，然后点击“添加服务定义”按钮，选中需要添加的服务JDBC数据源，如下所示：

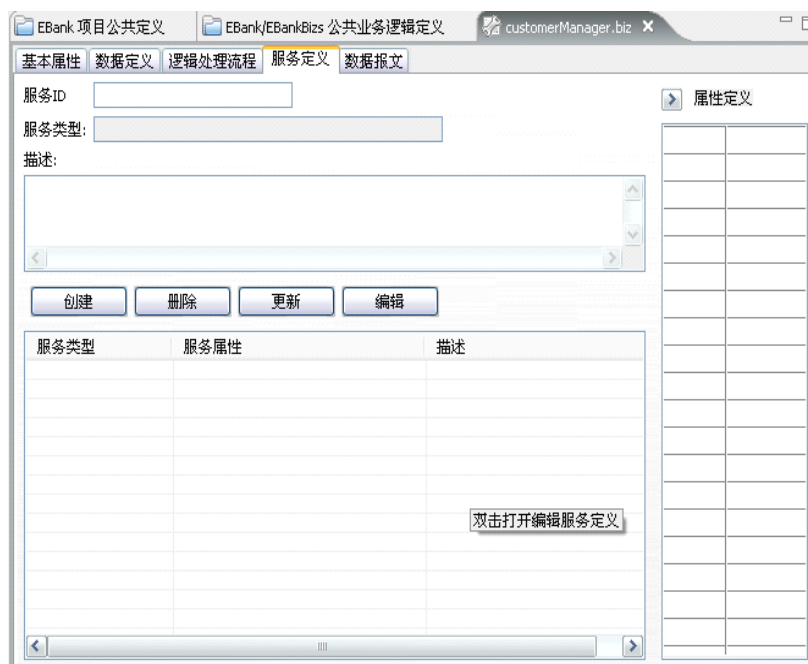


点击“Finish”按钮，会弹出如下对话框：

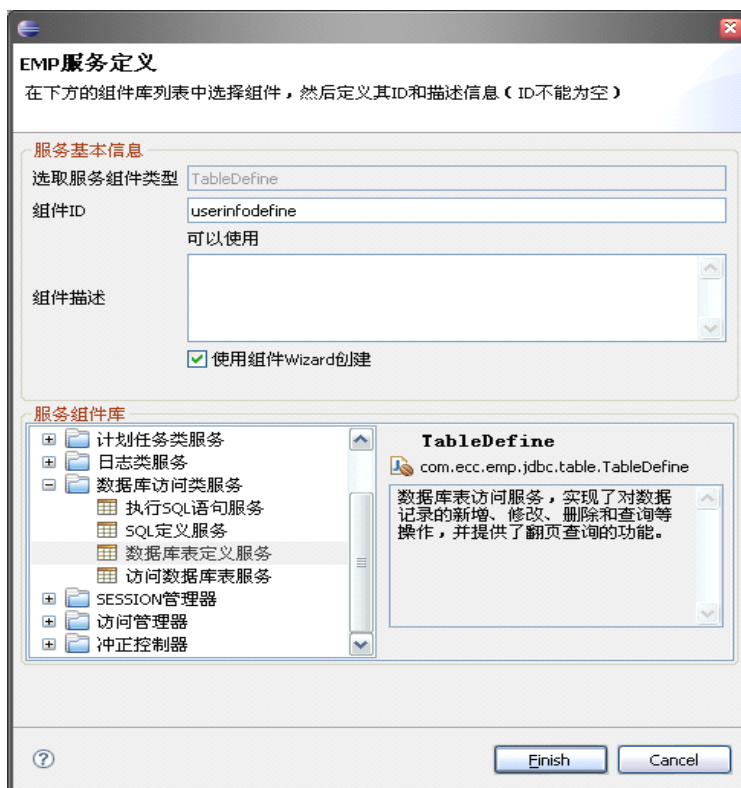


因为IDE的配置已经成功了，所以这里点击“连接”按钮，然后点击“Finish”按钮，即可完成对数据源的服务的定义。

介绍完datasource服务的配置，下面介绍一下如何在业务逻辑构件编辑器中配置tabledefine，使其可以访问到数据库的表。点击customerManager业务逻辑构件编辑器，打开“服务定义”页签，如下所示：



点击创建按钮，弹出创建服务的对话框，我们需要创建数据库表定义服务，如下所示：



点击“Finish”按钮，会弹出如下对话框：



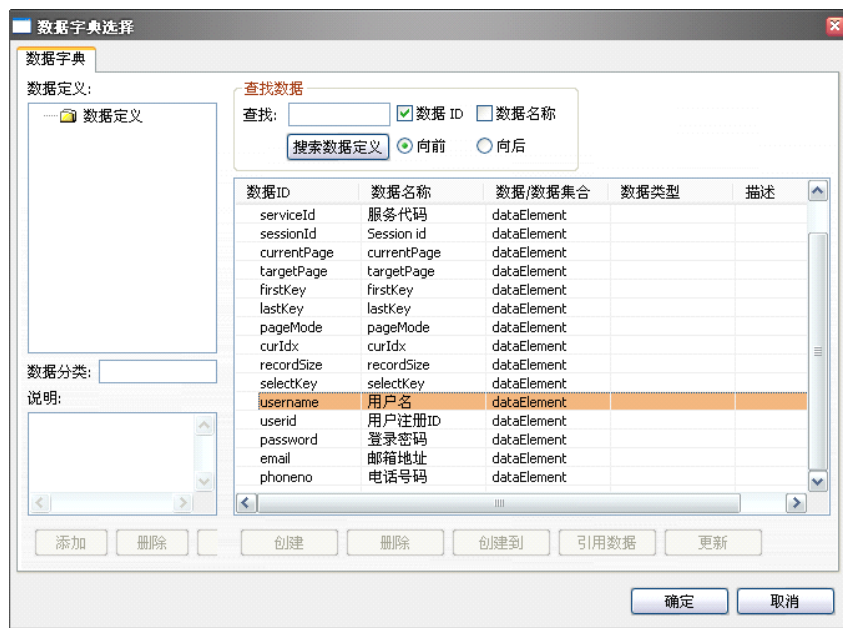
因为在前面我们已经将EMP IDE的数据库连接配置成功，这里只需要点击“连接”按钮，连接成功，然后点击“下一步”按钮，进入到下图所示：



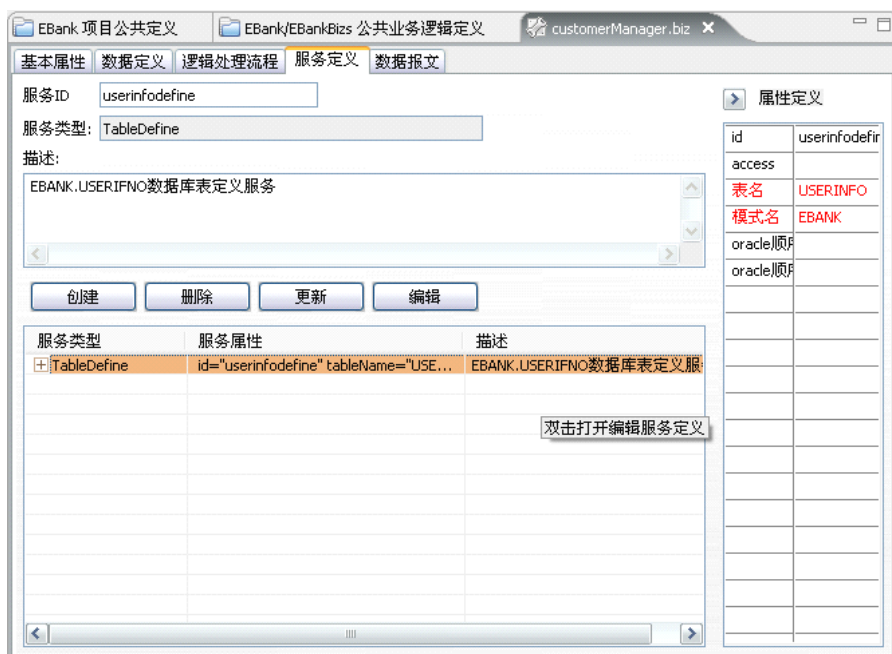
在这个对话框中，我们选择需要用到的数据库表EBANK.USERIFNO，点击“Next”按钮，我们接下来是要将数据库表中所要用到的数据选中，并映射到数据字典的数据项上，如下所示：



这里我们发现一个问题所用到的数据项中有一个NAME项没有和数据字典中的username对应起来，这样我们就要手动的将其映射上去，双击NAME，弹出如下对话框：

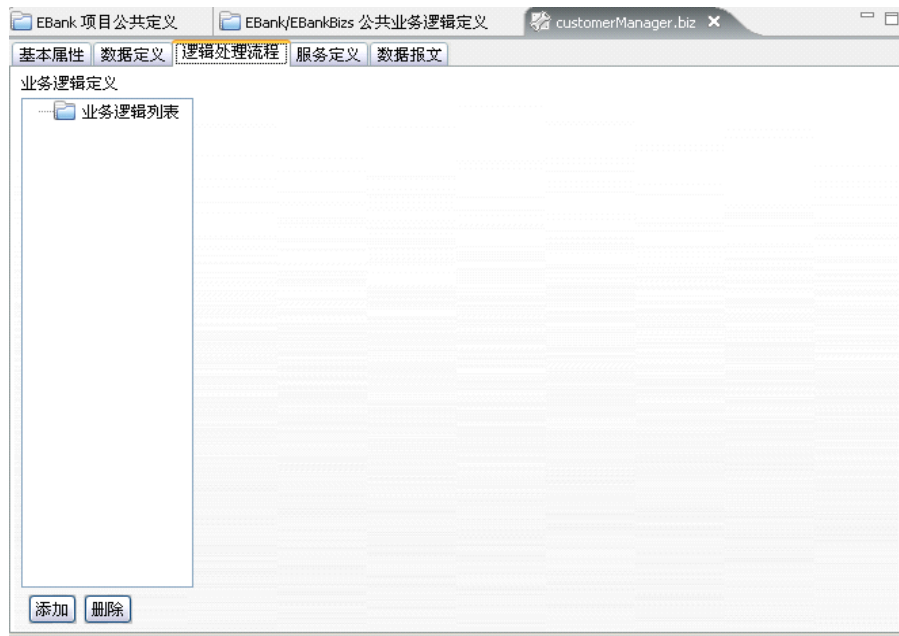


选中username然后点击“确定”按钮，则NAME就成功的映射到username上，然后点击“Finish”按钮，则EBANK.USERIFNO的表定义服务创建成功，如下所示：



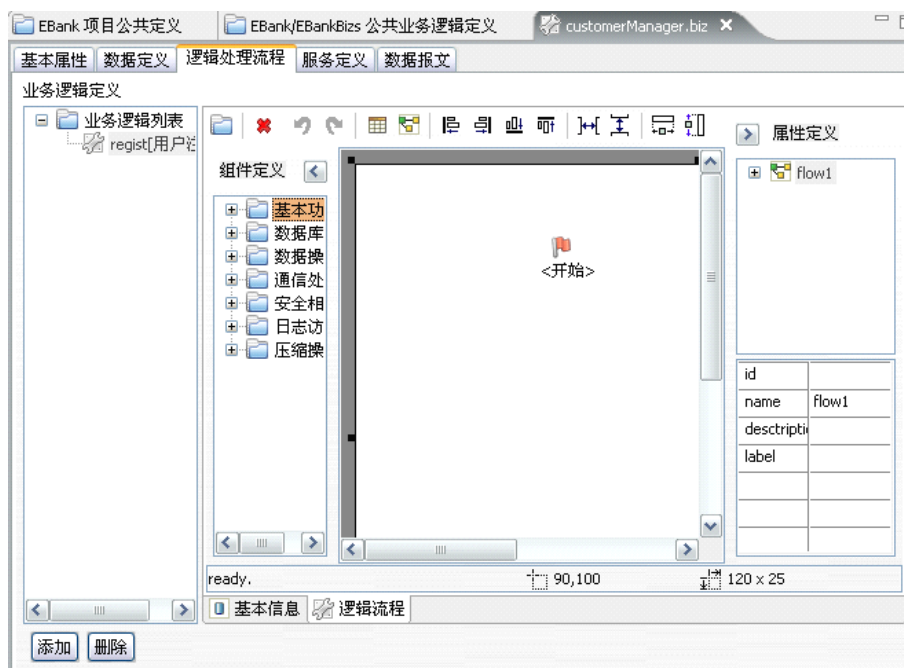
3.6.2.4. 定制业务逻辑实现

EMP IDE的数据源配置好后，我们可以实现“用户注册”这个业务逻辑了。打开业务逻辑构件customerManager.biz编辑器的“逻辑处理流程”页签，如下所示：

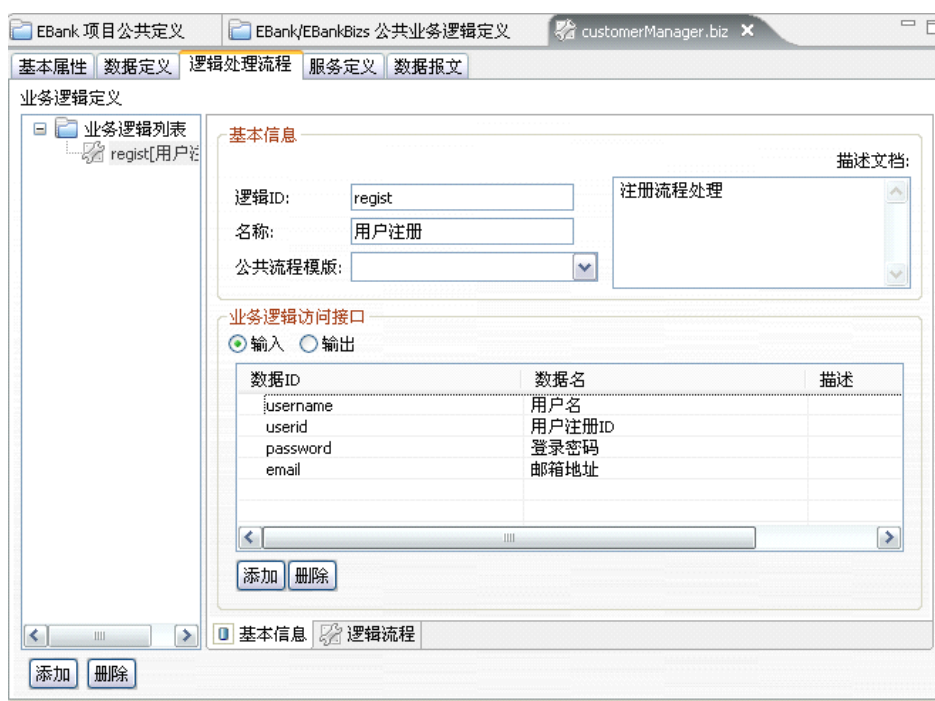


点击添加按钮，如下所示：

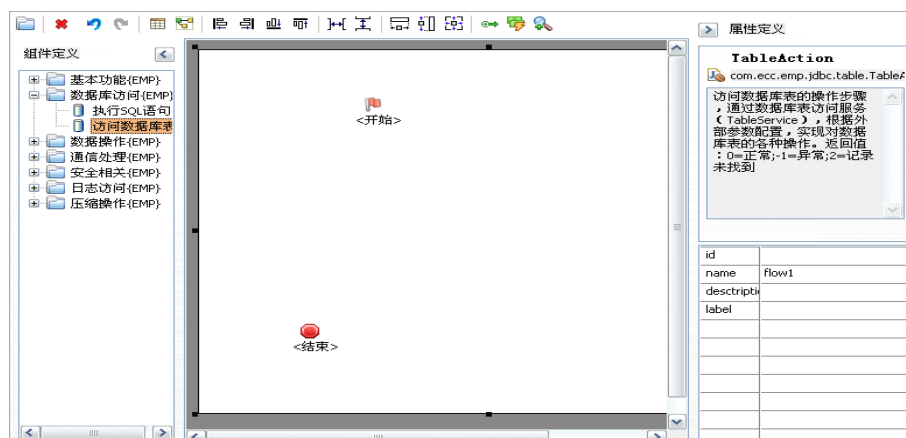
填写完业务逻辑的基本信息后，点击“Finish”按钮，完成“regist”业务流程的创建，如下所示：



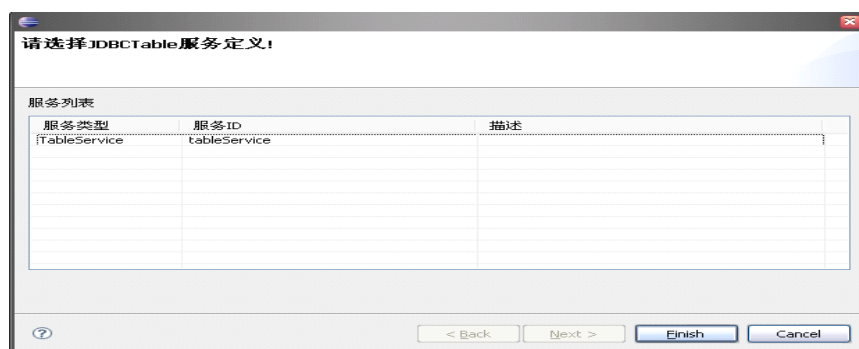
接下来点击“regist”业务流程编辑器的“基本信息页签”，添加该流程需要输入的数据项，如下所示：



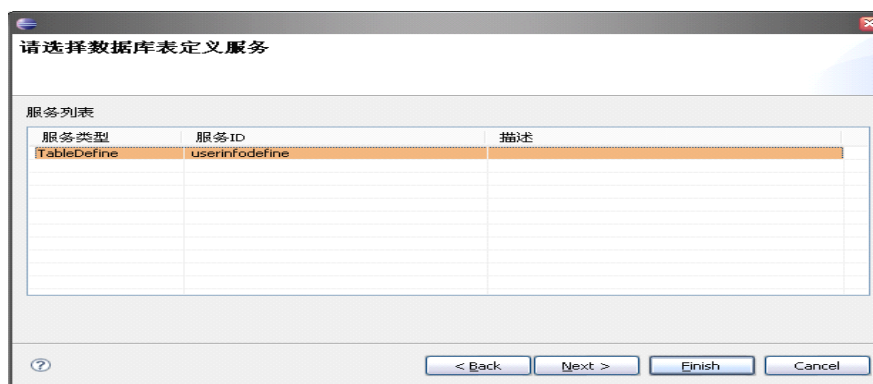
然后回到逻辑流程页签，对“regist”的逻辑流程进行定义，该操作流程很简单，就是一个数据库的插入操作，所以我们要往该流程中添加一个“访问数据库表”的action，如下所示：



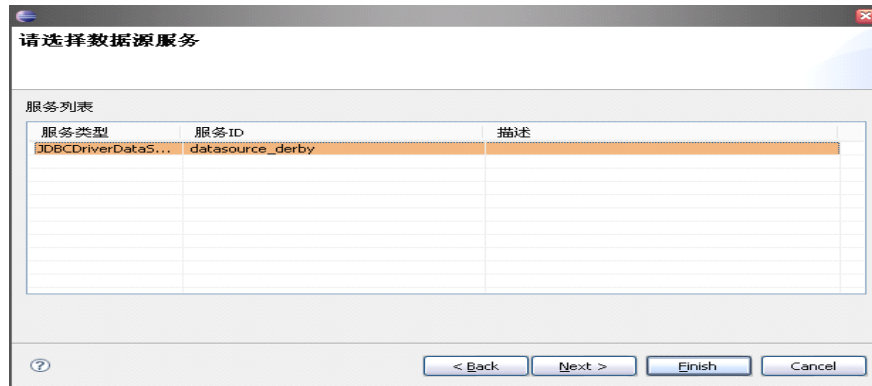
选中左边“访问数据库表”的节点，然后在右边的流程编辑框的空白处，点击左键，会弹出如下对话框：



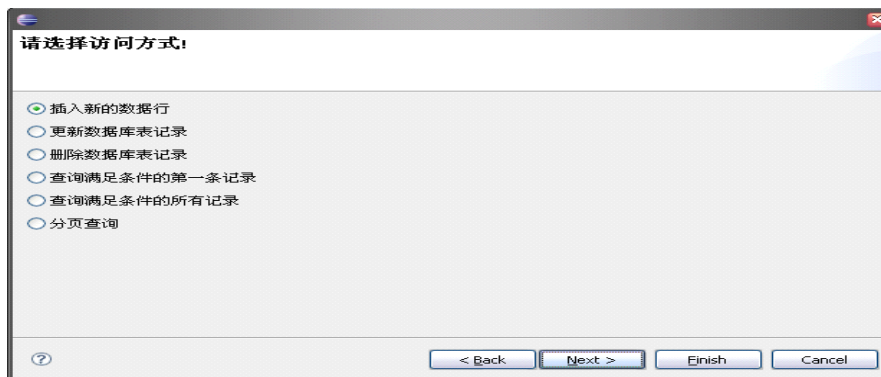
在该对话框中选中TableService服务，然后点击“Next”按钮，如下所示：



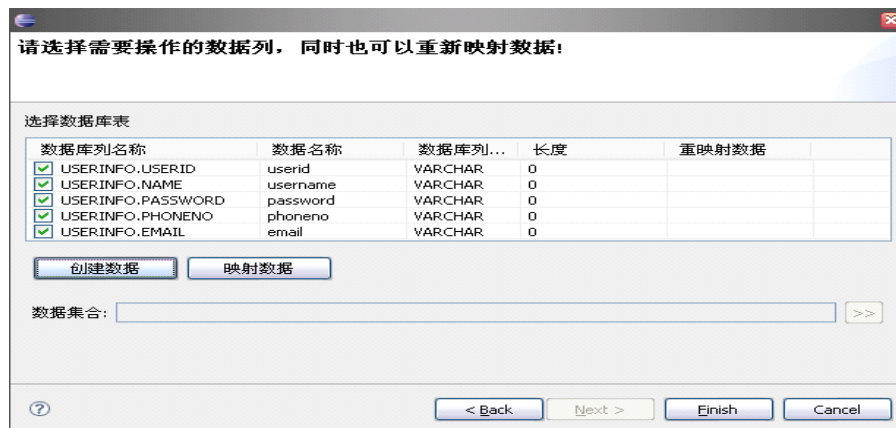
在该对话框中选中的数据源服务，点击“Next”按钮，如下所：



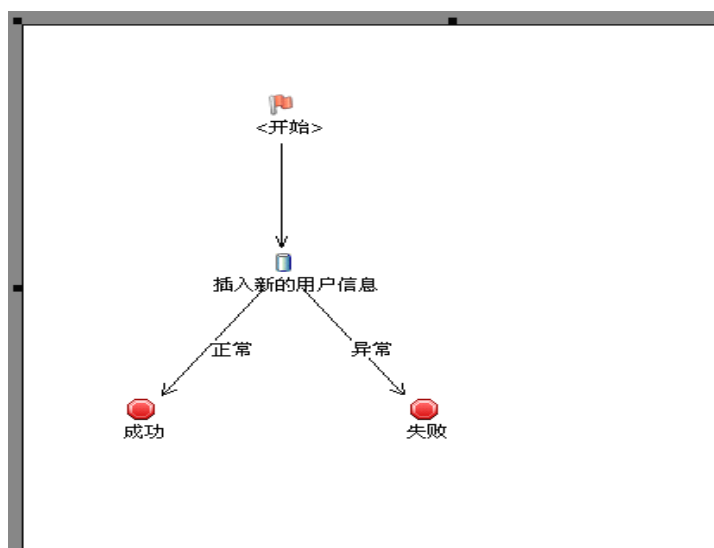
选择要进行的数据库操作，这里我们选择“插入新的数据行”，如下所示：



点击“Next”按钮，弹出如下对话框：



点击“Finish”按钮，插入数据库的action被成功添加，然后将该action组织到业务逻辑当中，如下所示：



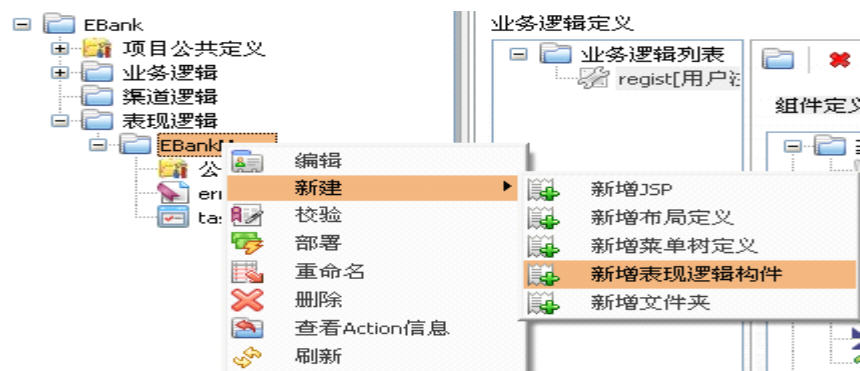
其中成功结尾的返回值为success，失败结尾的返回值为：failed。

注：当“插入新的用户信息”action执行成功的时候，正常的转移条件为：`$retValue='0'`，执行失败的时候，异常的转移条件为：`$retValue='-1'`。当然action失败\$reValue不为-1，可以将转移条件设置为：`exception:com.ecc.emp.jdbc.EMPJDBCException`。

3.6.3. 练习二：实现表现逻辑构件

3.6.3.1. 建立表现逻辑构件

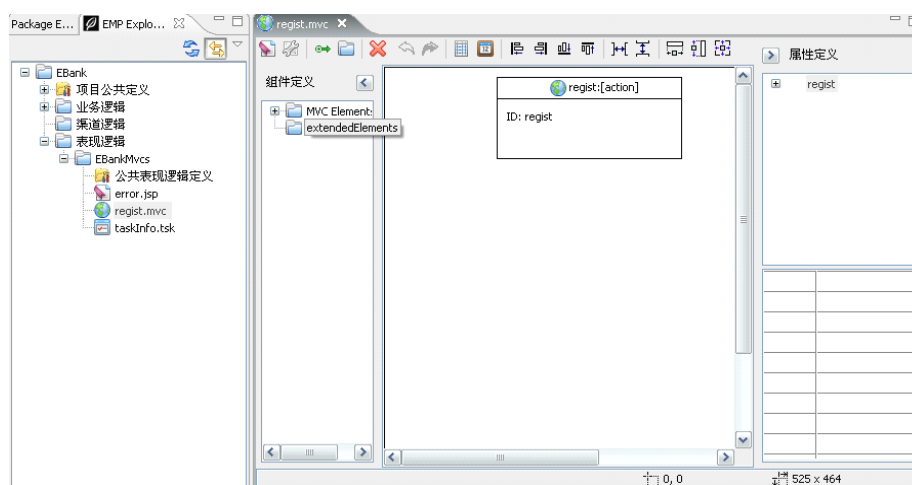
选中表现逻辑分组“EBankMvcs”，点击右键—》“新增”—》“新建表现逻辑构件”，如下所示：



按照下面的方式填写相关信息，然后点击“Finish”按钮：



成功添加了一个新的表现逻辑构件，如下所示：



3.6.3.2. 表现逻辑的实现

该表现逻辑需要一个输入页面，两个结果页面，其具体流程如下所示：


```
注册成功  
</body>
```

➤ registFail.jsp 页面的编辑

```
<body>  
注册失败  
</body>
```

3.6.4. 练习三：项目部署运行

在EMP Explorer视图中右键点击HelloWorld项目，选择部署。部署的过程就是将designFiles中的所有开发中文件(*.biz，*.mvc等)编译为EMP运行平台文件放到WebContent/WEB-INF目录下，同时把*.jsp等不需要编译的文件直接复制过去。

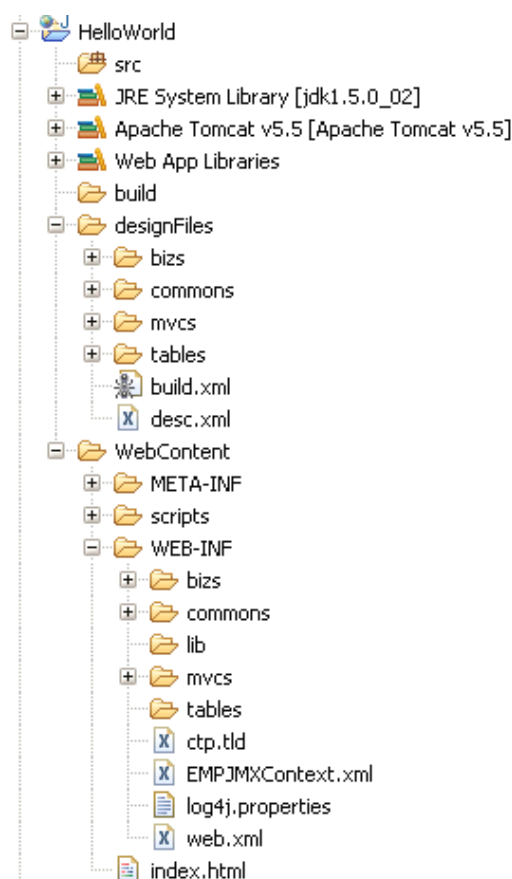
部署完毕后启动Tomcat服务器（若提示同步错误，试着刷新项目），在浏览器中打开<http://localhost:8080/EBank/regist.do>，填入注册信息并提交，查看运行结果，如下图所示：

注册ID:	<input type="text"/>
用户名:	<input type="text"/>
密码:	<input type="password"/>
EMail:	<input type="text"/>
<input type="button" value="注册"/>	<input type="button" value="重置"/>

填入注册信息，然后点击“注册”按钮进行注册。用户的注册信息被成功插入到数据库时，返回成功页面，显示“注册成功”，执行失败的话，返回到失败页面，显示“注册失败”。

3.6.5. EMP 理论知识介绍

3.6.5.1. EMP 项目文件结构



EMP运行平台及其他第三方组件的jar包都放在lib目录下。

在EMP Explorer视图中，有业务逻辑、表现逻辑和数据库表三个子目录，其中业务逻辑对应designFiles/bizz，表现逻辑对应designFiles/mvcs，而数据库表对应designFiles/tables。

3.6.5.2. EMP 项目全局定义

在EMP Explorer 视图的HelloWorld 项目上点击右键，选择“项目扩展定义”，打开项目设

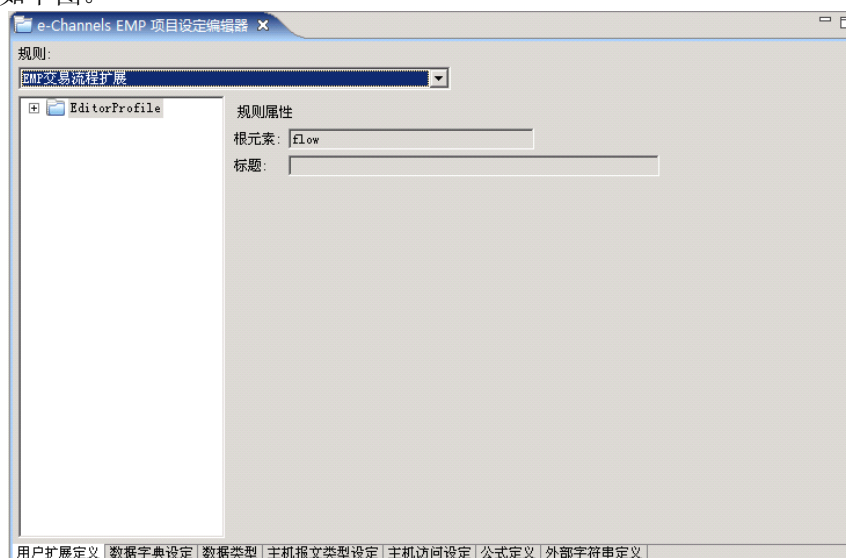
在Package Explorer中展开项目目录树，可以看到项目的目录结构。

src: Java源代码目录，用于存放用户自己开发的扩展功能代码。

designFiles: IDE开发中文件目录。用于IDE开发的配置文件就存放在这个目录，其下又分为bizz（业务逻辑）、mvcs（表现逻辑）、commons（公共配置）和tables（表）几个子路径。

WebContent/WEB-INF: Web项目实际发布路径，存放由IDE的开发中文件编译而来的EMP运行平台实际的配置文件及页面等。其下的bizz和mvcs目录与designFiles中的同名目录相互对应，而commons存放的是日志和资源文件。

定编辑器，如下图。



在该编辑器中可以进行EMP应用项目的全局定义，包括：

用户扩展定义：在此定义用户扩展的EMPAction业务逻辑步骤和Service服务的基本信息、设定属性、允许子元素等，供IDE开发使用。

数据字典设定：定义项目的数据字典，保证开发的规范性和一致性。这一数据字典对该项目中所有的业务逻辑/表现逻辑生效。

数据类型：定义项目中所用到的数据类型，对应服务器/客户端的数据校验与转换。

主机报文类型设定：定义主机报文类型、生成类和设定属性，用于配置后台主机报文。

公式定义：定义常用函数。

外部字符串定义：定义字符串外部化，支持多语言。

在我们本次课程中只需要进行数据字典和数据类型的设定即可。

➤ 定义数据字典

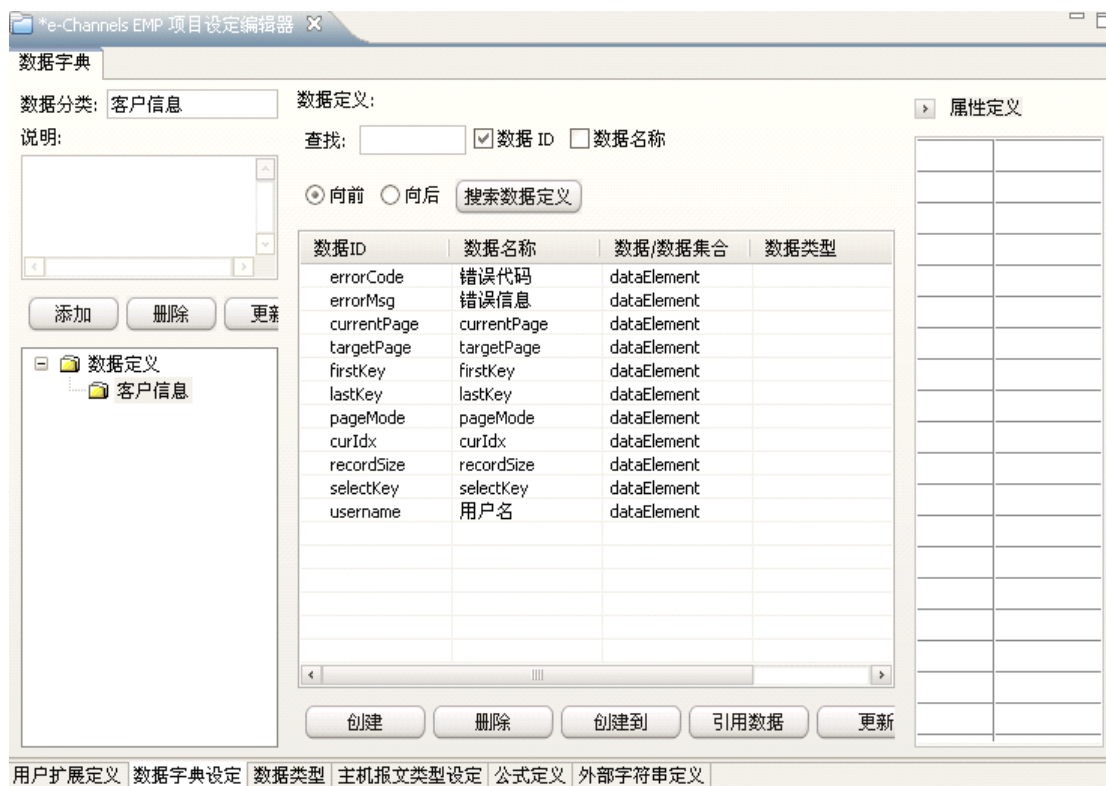
使用IDE进行应用开发时，必须遵循数据字典（IDE强制要求），每个业务逻辑的交易数据都是从数据字典中选取出来的，保证了整个开发工程的数据、变量名用词统一。所以在进行开发之前，需要预先对数据字典进行设计。在交易开发过程中，也可以随时为数据字典添加需要的元素，但为避免为小组协作带来麻烦，最好至少对各模块公共使用的数据进行预先设计。数据字典定义保存为XML文件形式（designFiles/commons/dataDict.xml），并可以通过CVS进行开发

同步。

所有数据分为两种，数据集合DataCollection和数据元素DataElement，分别对应EMP中的indexedCollection（简称iColl）和dataField。数据元素为单个独立数据，而数据集合中包含（引用）多个数据元素，用于体现重复性数据。

定义数据字典的部分内容在HelloWorld的设计过程中已经用到过，这里再详细的介绍一下数据定义的方法：

为了方便起见可以将数据划分为几个分类。例如，在左上角数据分类文本框中填写“客户信息”，点击“添加”，则在下方的数据定义分类树中添加了一个新分类。如下图：



点击“客户信息”分类，点击下方的“创建”，在弹出的对话框中对数据的ID，中文名称，数据类别（集合/元素），数据类型等进行定义，点击“确定”就在该分类中添加了一个数据定义。点击取消结束数据定义。一旦一个数据被创建后，除了数据类别不能再被修改之外，其它信息等能够被修改。



3.6.5.3. EMP 的表现逻辑层机制

➤ MVC机制的概念和原理

MVC(Model-View-Controller，模型—视图—控制器模式)软件架构模式把软件系统分为了三个基本部分：模型 (Model)、视图(View)和控制器(Controller)。模型 (Model)。

数据模型 (Model) 中存有持久性的应用数据。“模型”有在备份数据上“写”的权利，例如在数据库上。“模型”并不了解“视图”和“控制器”对它进行的操作，也就是说，模型被显示和被修改时，它是不知情的。但是在模型上进行的修改会通过一种刷新机制被公布。为了实现这种机制，那些用于监视此模型的视图必须事先在此模型上注册，通过这种形式，视图就会了解在数据模型上进行的修改。

视图 (View) 在视图层能够实现数据有目的的显示 (理论上，这不是必需的)。程序上的逻辑应该在此层面上被清除。为了实现在视图上的刷新功能，视图应该知道它监视的数据 (Model)，并且视图应该事先在被它监视的数据那已经注册了。

控制器 (Controller) 控制器起到在不同层面上的组织作用，它接受用户的操作，并对之进行解析，它有在数据模型上写的权利。它包含有“智能”并且能控制应用的流程。

模型—视图—控制器模式的目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。除此之外此模式通过对复杂度的简化使程序结构更加直观。

➤ EMP的MVC机制

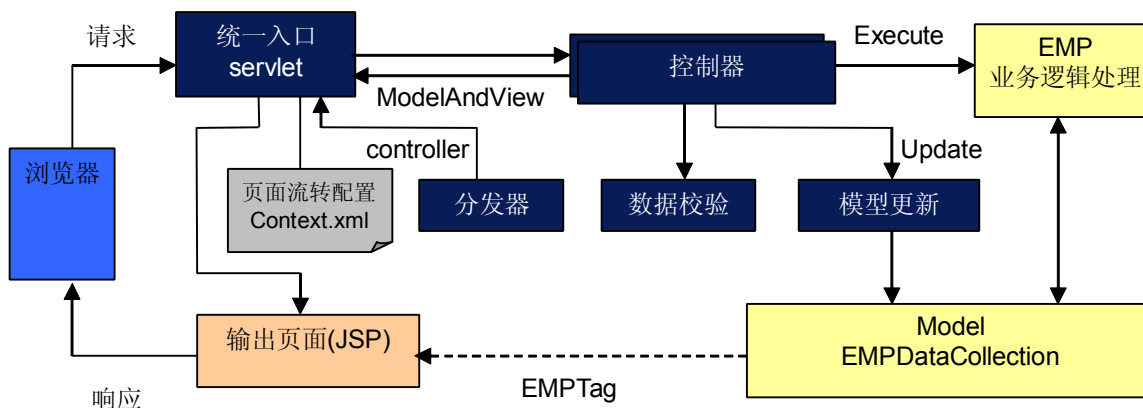
作为基于J2EE体系的Web应用基础框架的EMP平台，采用了如下的MVC架构模式：

数据层(M): EMP采用indexCollection/KeyedCollection/DataField结构的数据模型, 并把业务逻辑看作是数据层的一部分, 通过业务逻辑对数据进行处理, 在数据库/主机和数据模型之间进行数据交换。

视图层(V): EMP使用标准的JSP作为页面表现, 使用内建的Ajax支持管理页面布局; 提供了与数据层通讯(获取、显示、提交数据)、支持多语言资源并且能够简化开发的taglib标签库。

控制层(C): 从视图层页面提交的请求由一个统一入口Servlet接收, 通过分发器获得相应的控制器, 调用相应的数据层业务逻辑, 并可在其中对数据进行校验和更新。

以下是EMP MVC机制的示意图:



在IDE中, EMP应用系统被重新抽象为公共定义模型, 业务处理逻辑模型和表现逻辑定义模型三部分, 其中表现逻辑对应着mvc模型的视图层和控制层, 并对数据层引用进行定义; 而作为EMP特色和重点的核心交易组件(属于数据层)则被单独拿出来成为业务处理逻辑模型。

3.6.5.4. EMP 的业务逻辑层机制

3.6.5.4.1. EMP 应用项目—业务逻辑分组—业务逻辑构件—业务逻辑定义

EMP应用项目就是一个J2EE Web项目, 一个EMP应用项目包含一个或多个业务逻辑分组, 还包含一个或多个表现逻辑分组。

业务逻辑分组是EMP业务逻辑应用的一个完整组织，可以近似的认为是一个文件夹，文件夹下包含若干配置文件。每个业务逻辑分组拥有自己独立的一组公共配置文件，包括：

- 独立的公共配置设定
- 独立的节点定义
- 独立的公共流程定义
- 独立的公共报文定义
- 独立的公共服务定义

一个业务逻辑分组是不依赖于其他业务逻辑分组的。一个业务逻辑分组中包含任意多个业务逻辑构件。

业务逻辑构件是EMP完成业务逻辑定义的最小单元，可以类比为一个文件，在一个业务逻辑构件中，可以实现多种业务逻辑定义（类比为在一个Java类中包含多个方法）。在业务逻辑构件中会定义相关数据、服务、报文等资源。

业务逻辑定义即完成一个功能的定义，包含输入输出接口，处理流程等。

3.6.5.4.2. 数据和服务

我们描述一个业务功能时，可以将其抽象为几个要素：流程、数据、接口、服务，所谓服务就是一些资源，例如数据源。

流程一定是与一个业务处理紧密相关的，而接口也是与一格具体业务处理关联的，所以这两者在**业务逻辑定义**中进行描述。

而**数据和服务**都是一种资源，流程访问这些资源来实现其特定功能。是资源就存在组织的要求，EMP 是按照一种资源链的模式进行组织的。

资源链（Context）前面章节已经讲解过了，EMP 通常按照四级模型来组织资源链，参阅 2.2.6 章节。资源链中某个节点可以向上访问到其父节点（包括父节点的父节点）的数据和服务，但是不能访问兄弟节点的数据和服务。

在进行应用开发时，首先要对资源链进行设计，即明确哪些数据和服务将定义在公共节点（即根节点）上，哪些数据和服务要定义在应用节点上（即 Session 节点），哪些数据和服务要

定义在交易节点上。全局唯一的数据和服务通常定义在根节点上，而会话数据（如登录用户 id 等）通常都定义在 session 节点上，而交易范围内数据定义在交易节点上。

而在进行业务逻辑构件定义时，首先要做的是分析本构件会涉及到哪些数据和哪些服务。将本构件中各个业务逻辑定义中涉及到的数据一并定义在构件编辑器的“数据定义”中。该过程可以类比为编写程序时，首先要进行变量声明和引用方法声明，声明后，数据才可以被使用。

3.6.5.4.3. 流程和组件

业务功能的另一要素：流程，定义在业务逻辑构件中的一个逻辑定义下。流程通过可视化的编辑器进行定义，定义的方法是将EMP提供的若干基础组件进行装配，形成处理流程。

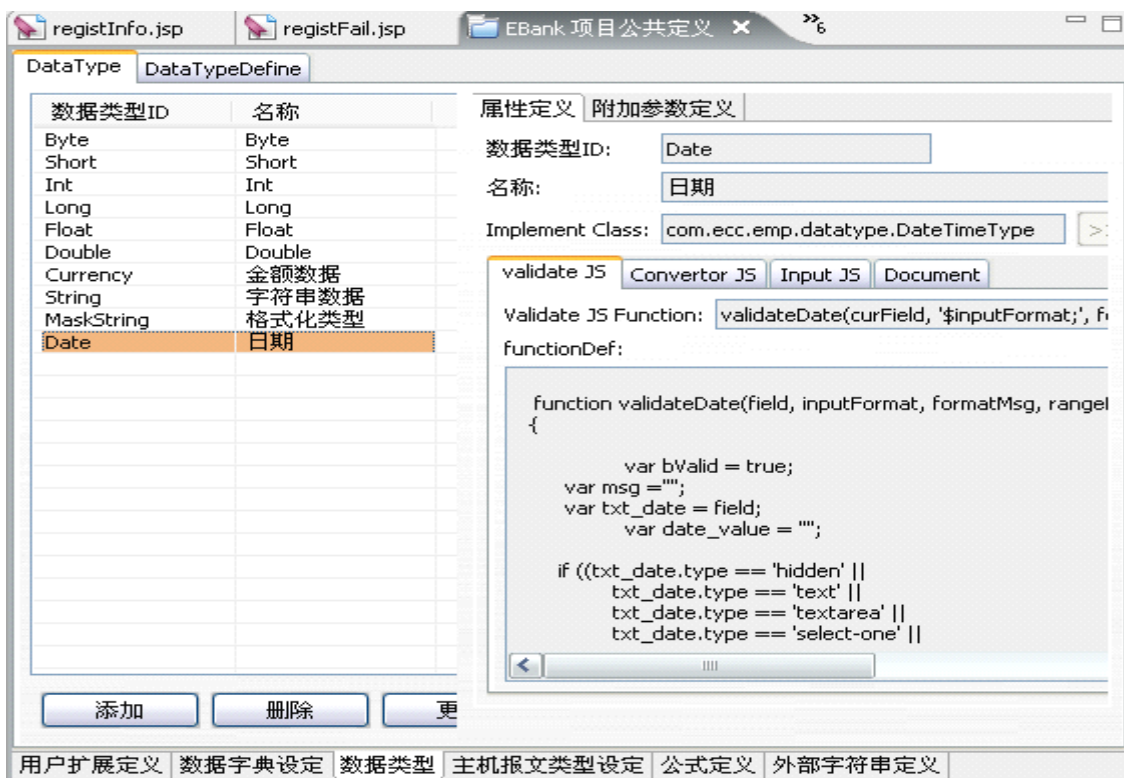
EMP提供了很多基础组件，无须编码即可完成诸如数据库操作、数据计算、通讯等功能。此外这样的基础组件还可以进行扩展（编写自己的Java代码，并包装为组件后，即可与EMP基础组件一样使用来组装处理流程）。

3.6.6. 练习四：应用优化

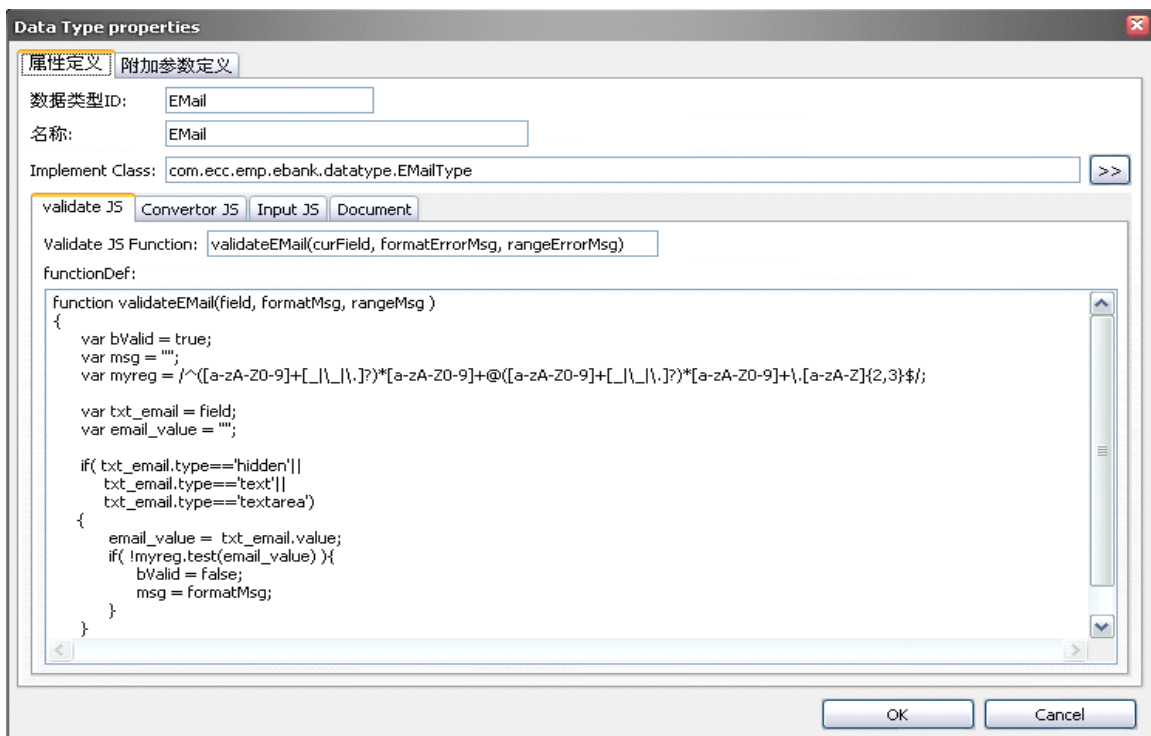
3.6.6.1. 数据类型实践——email 数据类型处理

通过前面的练习，我们已经完成了一个简单的注册流程，当该流程离实际应用还有一定距离，比如说，实际当中会做一些输入文本的格式校验。在这里我们会介绍一下如何使用EMP的校验机制来完成对输入格式中邮件的校验。

打开EBank项目公共定义的数据类型页签，如下图所示：



在DataType页签上点击“添加”按钮，为Ebank应用添加一个DataType名为：Email，如下图所示：



如上图所示：填写Email的基本信息，其中Implement Class 是负责对该数据项的后台校验

的，而validate JS, Convertor JS, Input JS是负责在数据项在提交前做的前端处理。这里我们只需要做一个校验的方法。具体内容如下所示：

```

Validate JS Function: validateEMail(curField, formatErrorMsg, rangeErrorMsg)
FunctionDef:
function validateEMail(field, formatMsg, rangeMsg )
{
    var bValid = true;
    var msg = "";
    var myreg =
    /^[a-zA-Z0-9]+[_\.\-]?[a-zA-Z0-9]+@([a-zA-Z0-9]+[_\.\-]?[a-zA-Z0-9]+\.[a-zA-Z]{2,3})$/;

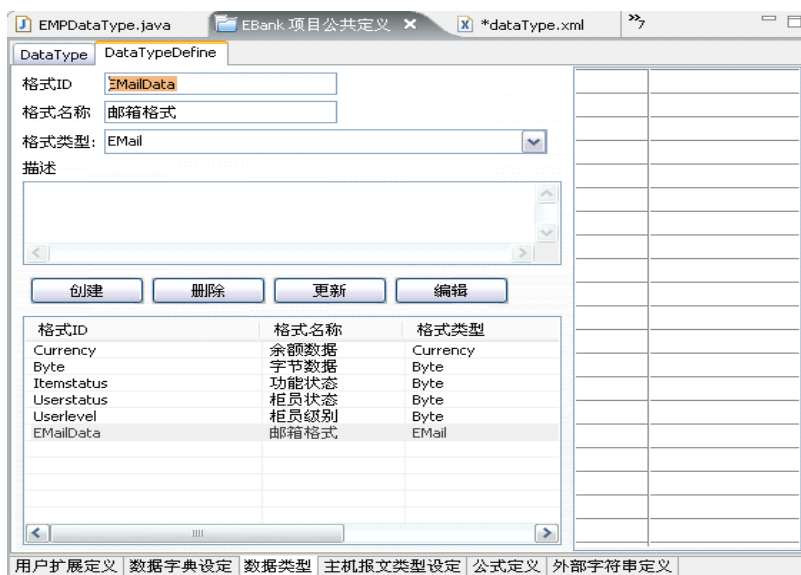
    var txt_email = field;
    var email_value = "";

    if( txt_email.type=='hidden'||
        txt_email.type=='text'||
        txt_email.type=='textarea')
    {
        email_value = txt_email.value;
        if( !myreg.test(email_value) ){
            bValid = false;
            msg = formatMsg;
        }
    }
    return msg;
}

```

注：这里我们虽然不需要后台校验，但是还是需要给Implement Class设置一个类EmailType，该类必须继承com.ecc.emp.datatype. EMPDataType。另外虽然我们只需要validate JS，但是如果设置另外两个JS的话，可能出现显示错误，所以可以Convertor JS和 Input JS写上两个空的方法。

完成上述步骤后，我们已经成功的为EBank应用添加了一个叫EMail的DataType，但是DataType是不能直接参与校验的，参与校验工作的是DataTypeDefine。因为DataType可能还扩展出一些参数，给这些参数赋上固定的值，就是一个具体的DataTypeDefine。而这里EMail没有扩展的参数，所以我们不需要定制参数，我们可以在DataTypeDefine页签定义一个EMailData直接引用EMail格式类型即可。如下所示：



按照上述配置完称后，我们就可以在前段页面来引用EMailData进行校验了，例如，我们要将registInfo.jsp中提交的EMail内容进行一下邮件格式的校验只需要做如下修改：

```
....
<ctp:input dataName="email" type="text" dataType="EMailData" />
....
```

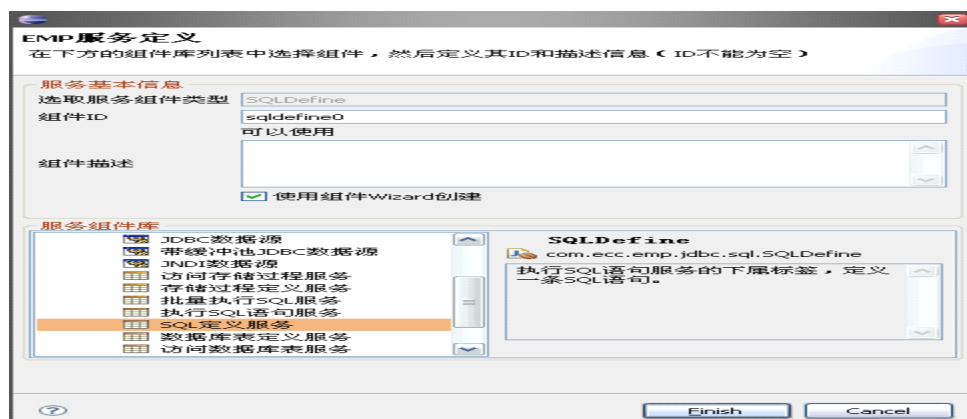
3.6.6.2. Ajax 实践——用户名是否存在校验

用户注册的时候，因为用户名不能重复，这个时候我们可以在用户写好注册ID以后，发送请求给客户端，然后在客户提交注册信息之前告诉客户注册的ID是否存在。

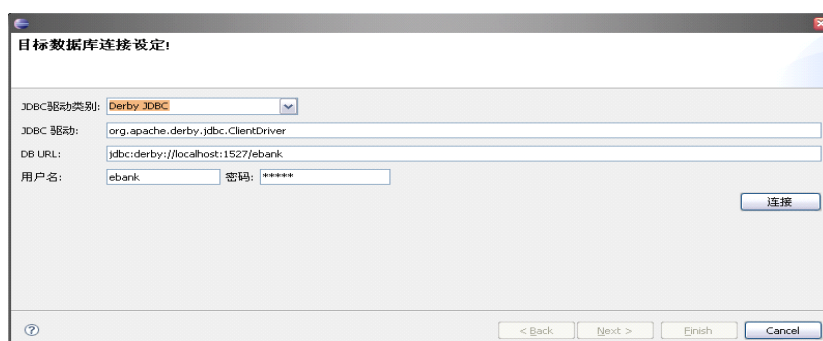
为了完成上述的需求，我们需要一个校验用户的表现逻辑和一个验证名称的业务逻辑。其中校验名称这个业务逻辑也可能在其它的交易流程里用到，所以我们可以将其定义为一个公共流程。接下来就介绍一下如何定义校验用户名是否存在的公共流程。

其中定义的公共流程的方法和定义注册流程的方法类似，当然这里也可以使用访问数据库表操作的action，但是表定义服务被我们定义到customerManager.biz业务逻辑构件当中了，因此公共流程是无法使用到的，所以在这里我们使用执行SQL语句的action来完成该公共流程。

要让公共流程能使用执行SQL语句的action，我们还需要在root节点上添加一个sqldefine的服务定义sqldef_checkuser。



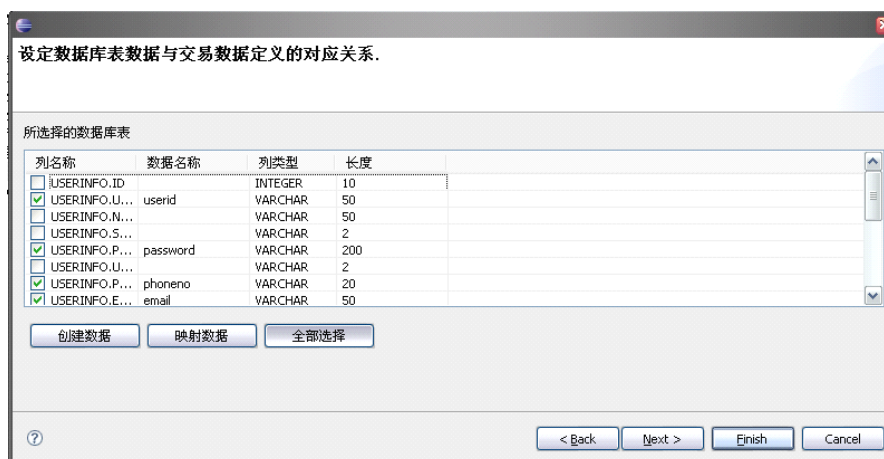
选择“SQL定义服务”，命名组件ID, 点击“Finish”按钮，完成。会弹出如下对话框：



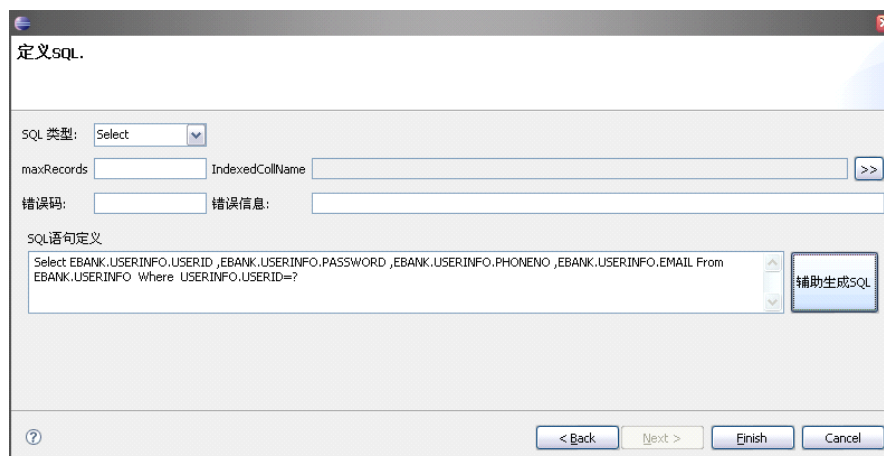
点击连接，选择Next，如下所示：



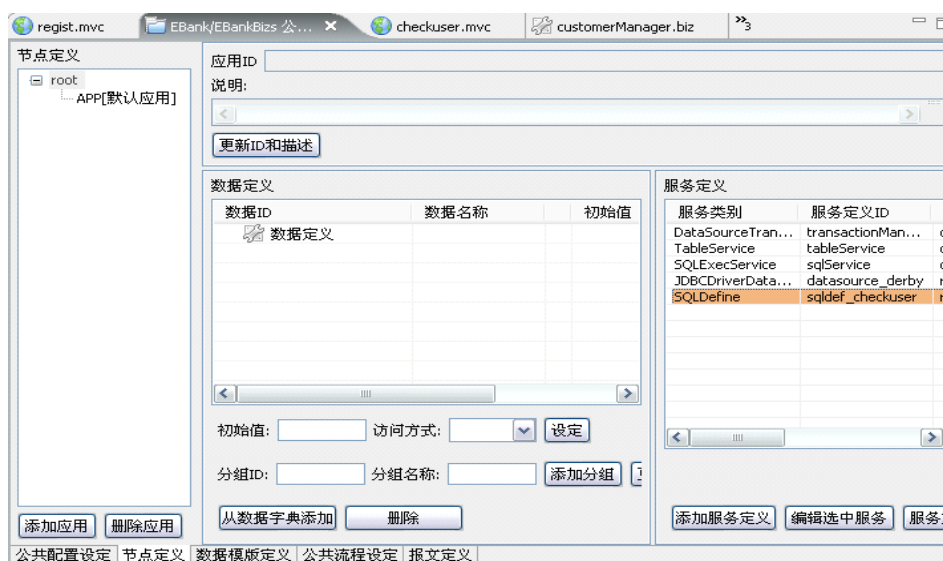
选中需要操作的EBANK.USERINFO表后，点击Next，如下所示：



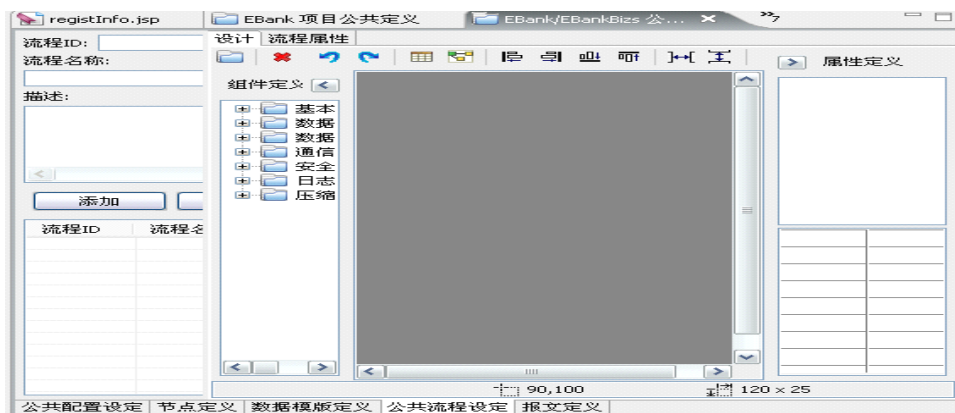
选中表中要操作的列，然后点击Next按钮



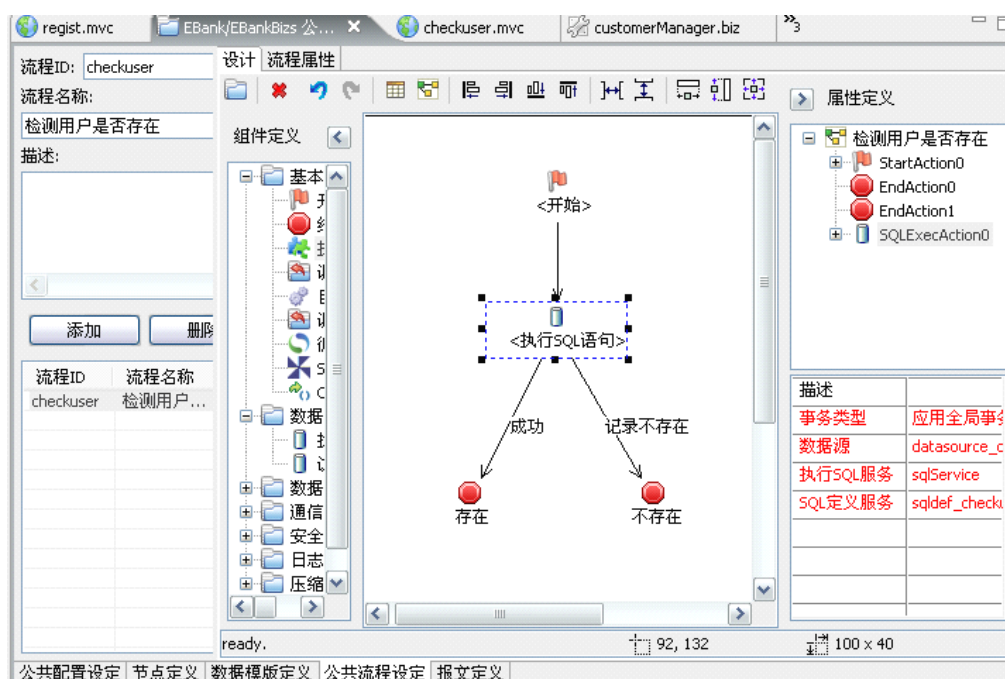
如上所示生成SQL语句，然后点击Finish按钮。至此sqldef_checkuser服务添加成功。如下所示：



接下来是定义公共流程：checkuser。打开“公共流程设定”页签，如下所示：



定义公共流程的方法如定义注册流程类似，结果如下所示：



其中“执行SQL语句”的action属性如下所示：

数据源: datasource_derby
 执行 SQL 服务: sqlService
 SQL 定义服务: sqldef_checkuser

接下来，我们需要在customerManager.biz的业务逻辑构件中建立一个checkuser的处理流程，该处理流程使用公共流程的方式有两种，一种是引用公共流程，一种是调用公共流程。

应用公共流程的方式为，在创建的流程的时候，在“公共流程引用”中选择checkuser这个公共流程，如下所示：



EMP业务构件定义

输入业务逻辑ID、名称、描述等信息，并可以选择 本业务逻辑所要引用的公共流程

业务逻辑ID: checkuse
可以使用

业务逻辑名称: 检查用户

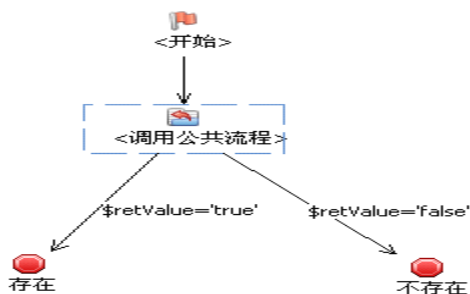
公共流程引用: checkuser

描述:

Finish Cancel

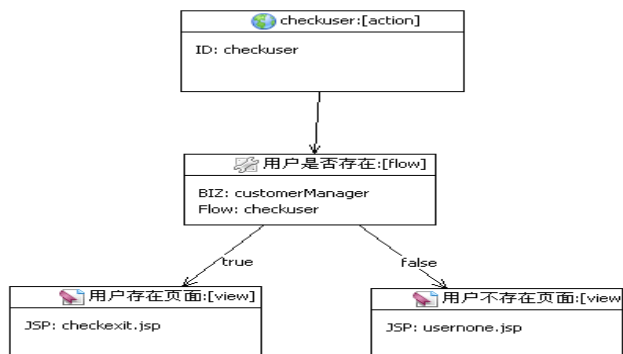
则customerManager.biz的业务逻辑构件中的checkuser处理流程完全引用了公共流程的逻辑结构。

另外一种调用公共流程，则是在编辑逻辑处理流程的时候，添加一个名为：“调用公共流程”的action组件，该组件的属性“引用公共流程”选择公共流程“checkuser”也可以达到调用公共流程的结果如下所示：



因为逻辑构件的checkuser流程处理操作与公共流程基本一致，不需要添加其它的处理方式，所以在这里应该选择第一种引用方式，比较简便。

定义完业务逻辑处理流程，接下来就是表现逻辑流程，表现逻辑流程的内容没有什么新的，具体定义如下所示：



现在要让registInfo页面能够异步的上后台检查用户是否存在，需要做几点修改，

- 首先要在页面中添加一个div来展现用户是否存在的信息，该div的ID为“isUserDiv”；
- 添加一个JS函数来向服务器端发送请求和刷新isUserDiv的信息，函数具体内容如下：

```
function isUser(){
    var userid = document.getElementById('userid').value;
    var url = 'checkuser.do?userid='+userid;
    updateDivContent(url, 'isUserDiv', '');
}
```

可以看到上面的函数里用到了一个updateDivContent的方法，要使用该方法，还需要往页面里面添加给一个标签，如下所示：

```
<ctp:yuiInclude/>
```

这样该方法关联的JS文件都会被引入到该页面内；

- 最后就是要给，注册ID的文本输入框添加相应方法，如下所示：

```
<ctp:input dataName="userid" type="text"
customAttr="onblur='isUser()'"/>
```

4. EMP 应用模拟实战

4.1.应用项目模拟开发方式

本次培训采用模拟项目开发方式进行。通过一个应用项目的开发实战逐步学习EMP平台的开发功能。

通过课程式组织方式，定义每次课程的学习目标，通过“胶片讲解+实际操作+课后练习”相结合的方式培训。

在后续章节里，将对本次培训所涉及所有课程和练习进行详细描述。

4.2.模拟应用项目描述

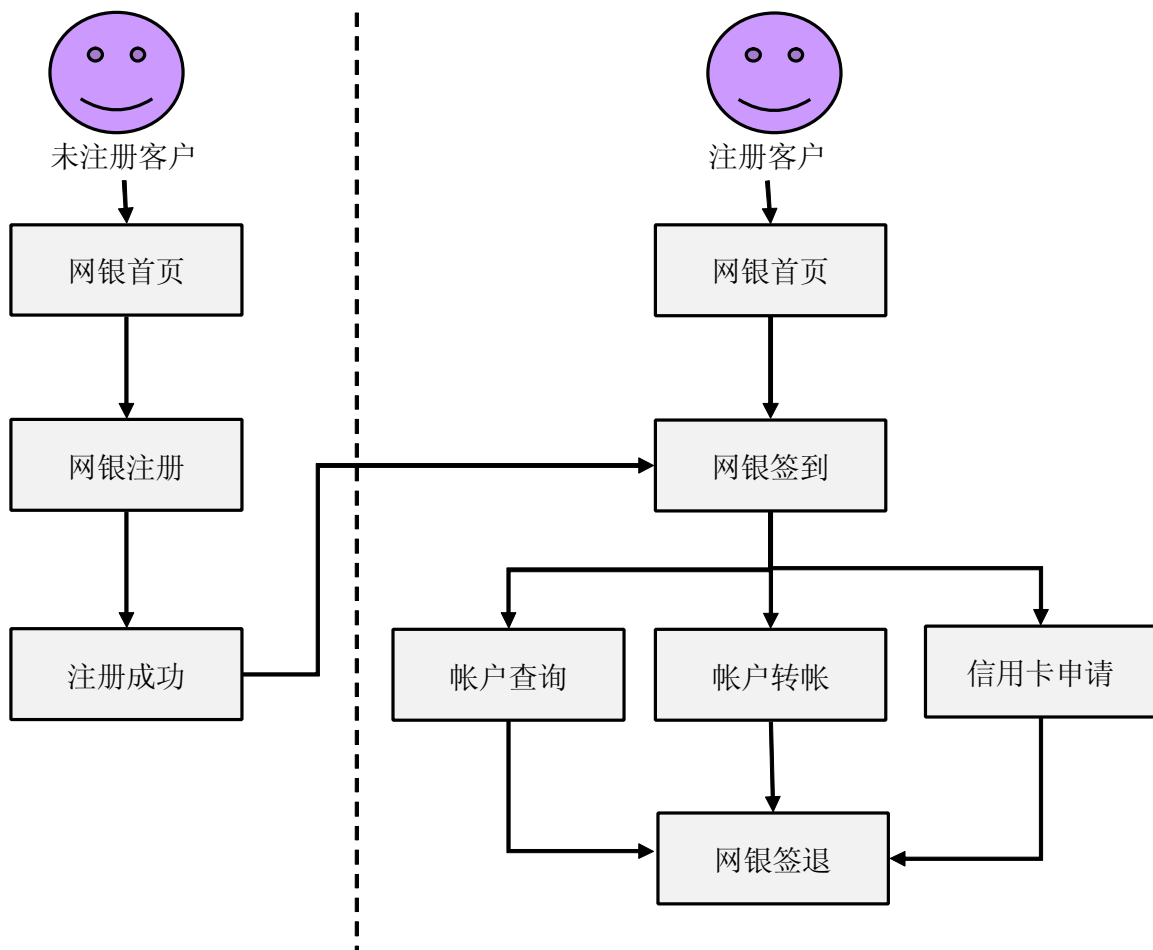
本次培训我们将通过一个网上银行个人银行应用的基础版本的研发来学习EMP平台的开发

使用。

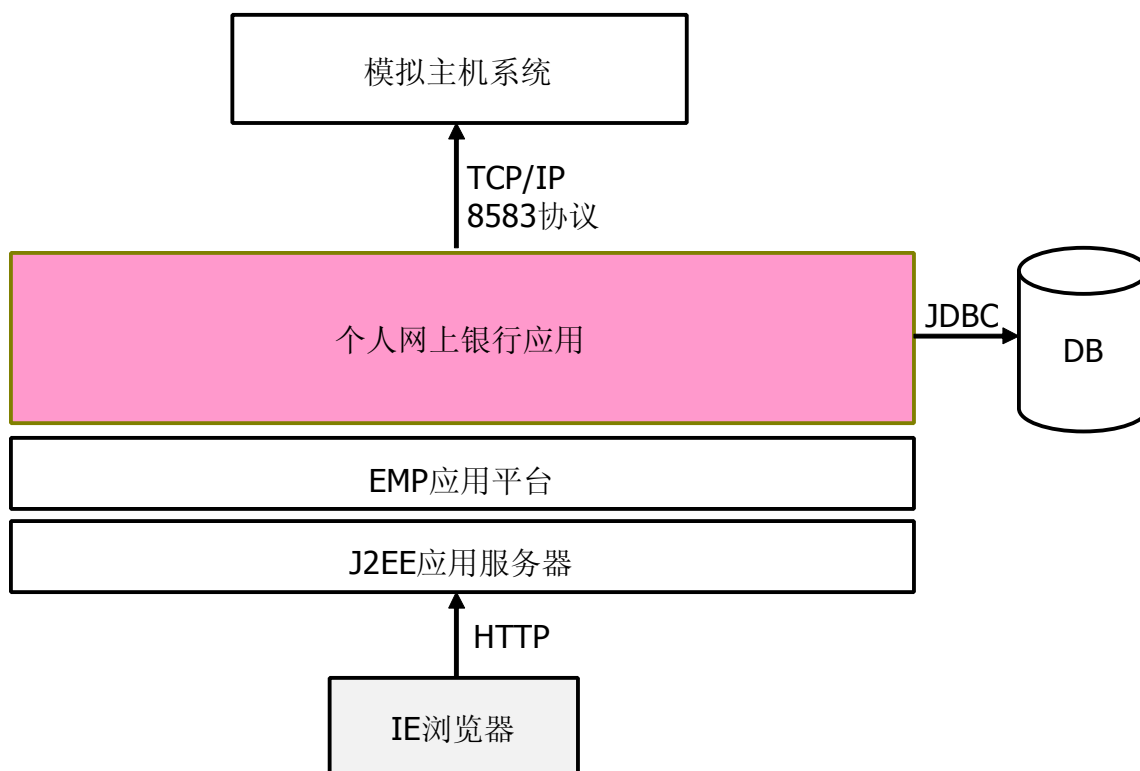
个人网上银行的基本功能有个人客户注册、签到、签退、个人客户帐户查询、个人客户帐户转帐、个人客户信用卡申请等功能。

4.2.1. 应用项目流程

个人网上银行操作的基本流程如下图所示：



4.2.2. 应用项目架构



4.2.3. 交易设计

业务交易描述			
交易码	交易名称	operationName	功能描述
	注册	regist	注册一个新客户
1000	签到	signOn	客户签到网上银行系统
1009	签退	signOff	客户从网上银行签退
0717	添加关联帐户	addAccount	添加一个客户的关联帐户，客户可添加多个关联帐号
0718	查询帐户信息	queryAccount	客户帐户基本信息查询
0720	查询帐户交易明细	tranDetail	客户帐户交易明细记录查询（多条交易记录返回）
0719	转帐	transfer	客户帐户转帐交易处理
0500	申请信用卡	applyCreditcard	网上填写个人资料申请信用卡

注：详细设计请参考附录

4.2.4. 数据库表结构

4.2.4.1. 客户信息表（CUST_INFO）

CUSTOMER					
序号	字段名	内 容	数据类型	长度	备 注
1.	CUST_ID	客户号	INTEGER		PRIMARY KEY NOT NULL 自动产生
2.	CUST_SIGNID	客户注册ID	VARCHER (50)		NOT NULL
3.	NAME	人员名称	VARCHAR (50)	50	NOT NULL
4.	PASSWORD	密码	VARCHAR (200)	200	NOT NULL
5.	EMAIL	电子邮件	VARCHAR (50)	50	
6.	PHONE	电话	VARCHAR (15)	15	

4.2.4.2. 客户帐户信息表（ACCOUNT_INFO）

ACCOUNTINFO					
序号	字段名	内 容	数据类型	长度	备 注
1.	USERID	客户号	INTEGER		PRIMARY KEY NOT NULL 自动产生
2.	ACCOUNTNO	帐号	VARCHAR (22)		PRIMARY KEY NOT NULL
3.	ACCOUNTNAME	帐户名	VARCHAR (20)		NOT NULL
4.	ACCOUNTTYPE	帐户类型	INTEGER		NOT NULL 0: 活期（缺省 值） 1: 定期
5.	ACCOUNTRATE	利率	FLOAT		NOT NULL
6.	CURRENCYCODE	币种	VARCHAR (3)		NOT NULL
7.	OPENDATE	开户日期	VARCHAR (8)		NOT NULL
8.	ACCOUNTBALANCE	帐户余额	DECIMAL (17, 2)		NOT NULL
9.	FREEZEBALANCE	冻结余额	DECIMAL (17, 2)		

4.2.4.3. 用户信息表（USER_INFO）

ACCOUNTINFO					
序号	字段名	内 容	数据类型	长度	备 注
1.	USERID	用户ID	VARCHAR(10)		NOT NULL PRIMAY KEY
2.	PASSWORD	用户密码	VARCHAR(30)		NOT NULL

4.2.5. 主机交易设计

三支主机交易			
序号	交易码	主机格式	交易描述
1.	0718	客户帐户信息查询	查询客户帐户基本信息
2.	0719	帐户转帐	帐户转帐交易
3.	0720	帐户交易明细查询	帐户交易明细查询（多条记录返回）

注：详细设计请参考附录

5. 开发实战课程

5.1. 课程 5：建立应用项目

5.1.1. 课程设计

课程名称	建立应用项目	课程时间	
课程目标	一、了解模拟应用项目基本情况 二、学习创建项目及配置服务器，运行项目 三、了解应用项目结构，掌握配置文件内容和含义		
准备工作	一、Eclipse 3.2 以上版本 二、EMP IDE 插件包 三、TOMCAT5.5 服务器		

5.1.2. 了解个人网银系统基本情况

对个人网银系统的基本情况概括讲解。参照4.2节。

5.1.3. 新建有 EMP 支持的 Web 项目

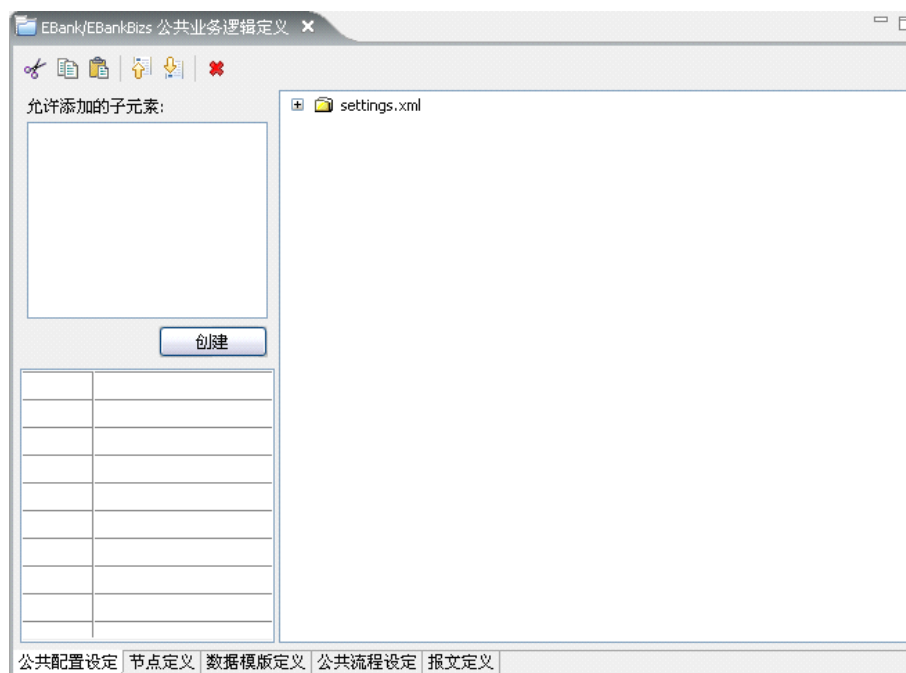
可以新建一个 EMP 支持的 Web 项目，也可以采用在上一课程所建立的 EMP 项目。
在这里我们采用在上一课程中所建立的 EMP 项目。

5.1.4. 定义数据字典

定义数据字典的方法已经在上一课程里已经介绍过来，这里为了方便起见，我们可以将数据字典的文件，直接导入进来。具体方法是将资源文件下的 dataDict.xml 文件拷贝到 designFiles\commons\目录下，覆盖它原有的 dataDict.xml 文件。

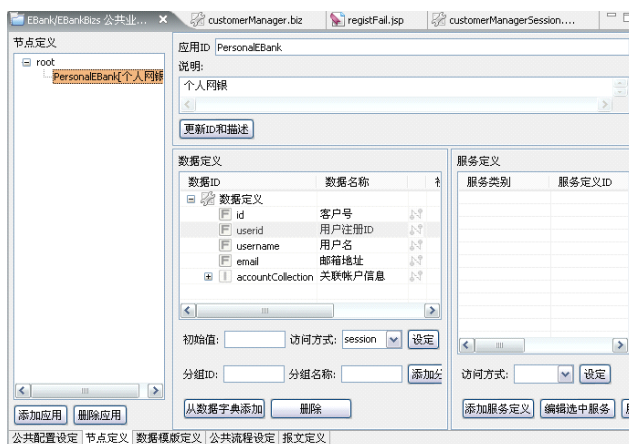
5.1.5. 业务逻辑处理节点 Context 设定

业务逻辑分组的设定：在EBankbizs分组下的公共业务逻辑定义，打开业务逻辑分组设定编辑器，如下图：



点击该编辑器下方的“节点定义”标签，进行应用节点定义。

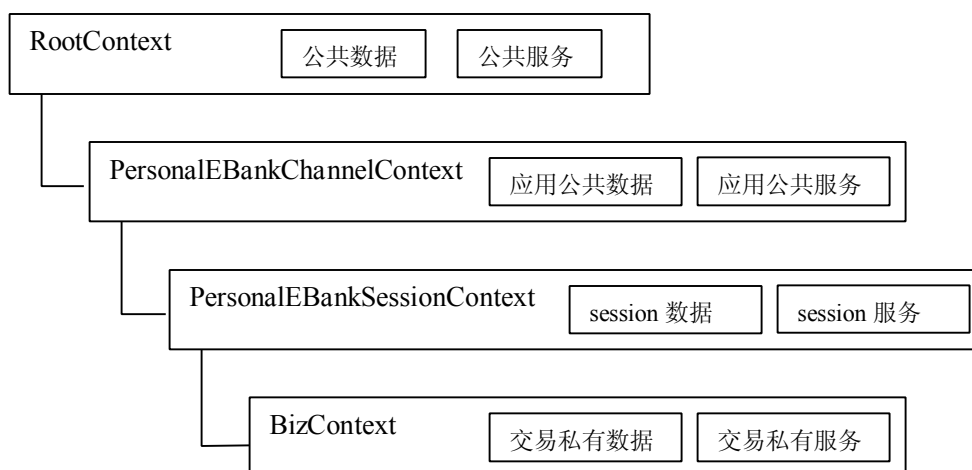
EMP提供内建的多应用支持，使得一次开发的服务端应用可以对应不同应用的前端。在本课程中只需定义一个“个人网银”应用即可：在业务逻辑分组设定编辑器的节点定义分页中，创建一个名为PersonalEBank的应用，如下图：



在定义了应用之后，会自动生成两个级别的交易上下文（Context）。以下是 contexts.xml 中的内容：

```
<contexts.xml>
  <context id="rootCtx" type="root">
    <refKColl refId="rootKColl"/>
  </context>
  <context id="PersonalEBankChannelCtx" parent="rootCtx"
type="channel">
    <refKColl refId="PersonalEBankChannelKColl"/>
  </context>
  <context id="PersonalEBankSessionCtx"
parent="PersonalEBankChannelCtx" type="session">
    <refKColl refId="PersonalEBankSessionKColl"/>
  </context>
</contexts.xml>
```

其中 PersonalEBankChannelCtx 是应用公共 Context，为该应用为所有前端连接会话所共用，其父节点是 rootCtx；而 PersonalEBankSessionCtx 是应用 sessionContext，存放通过该应用访问应用的某会话的 session 级数据和服务，父节点是 PersonalEBankChannelCtx。他们的关系如下图所示：

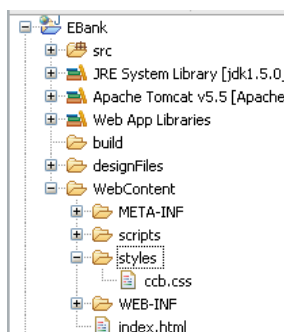


在本课程中，由于用户信息等是在用户登录到网银系统后一直保持的数据，属于会话级数据，因此需要在应用节点中从数据字典引用过来，并且设置其访问方式为session。需要引用的数据如下：

数据ID	数据名称	
数据定义		
id	客户号	
userid	用户注册ID	
username	用户名	
email	邮箱地址	
accountCollection	关联帐户信息	

5.1.6. 手工复制课程相关文件

在后续的课程中会用到一个ccb.css文件。ccb.css需要拷贝到WebContent\styles\目录下。拷贝完成的Package Explorer视图如下：



5.2. 课程 6：签到/签退处理

课程名称	数据库访问处理	课程时间	
课程目标	一、学习和掌握数据库连接的核心知识点（JDBC、连接池、数据源等） 二、学习并掌握 tableService,SQLService 和存储过程服务的配置方式 三、学习并掌握在业务逻辑处理步骤中进行数据库访问的配置 四、学习并掌握如何在业务逻辑处理中进行事务管理		
准备工作	完成课程 2		

5.2.1. 客户签到交易 signOn：数据库表（查询）

因为签到签退交易都是 session 交易，而注册是一个无 session 的交易，所以签到和签退不能跟注册一起放在 customerManager.biz 的业务逻辑构件里，所以我们需要新建一个业务逻辑构件名为：customerManagerSession。如下所示：

5.2.1.1. 交易数据引用

签到交易接受从页面端传来的用户注册ID和密码，然后查询数据库得到用户信息和关联帐户信息。由于这些数据基本上都是session数据，所以customerManagerSession只需要唯一的一个私有数据password，如下所示：

基本属性 数据定义 逻辑处理流程 服务定义 数据报文				
业务逻辑构件数据定义：				
数据ID	数据名称	初始值	描述	
数据定义				
errorCode	错误代码			
errorMsg	错误信息			
password	登录密码			

最后在基本属性上将该customerManagerSession.biz的“是一个无session交易”这一项给取消掉，如下所示：

基本属性 数据定义 逻辑处理流程 服务定义 数据报文	
逻辑定义ID：	customerManager
逻辑定义名称：	customerManager
<input checked="" type="checkbox"/> 是一个无Session交易	
描述：	

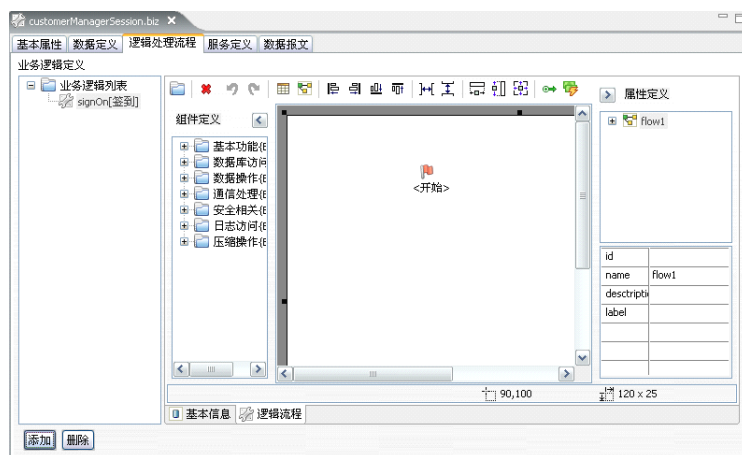
[illegible]

采用SQL语句来访问UserInfo表，传入用户名和密码，检查该用户是否存在且密码是否正确。我们采用EMP提供的SQLExecServiceSQLDefine对象来定义一个SQL语句的查询服务功能。定义sql定义服务的方法前面也介绍过，这里也不再复述，在root节点上添加查询用户名和密码的SQLExecServiceSQLDefine服务名为：sqldef_login,如下图所示：

在这一步中需要注意的是，在选中查询结果的时候，一定要将应用节点引入的数据项对应用的数据库列给勾中。

5.2.1.4. 流程定义

在业务逻辑构件customerManagerSession.biz创建名为signOn[签到]的流程，引用输入数据userid和password。如下图所示：



在引用userid的时候，可能会发现userid不在交易里，这个时候需要打开customerManagerSession的数据定义标签，点击“节点定义引用”按钮将userid先从应用节点上引用过来。再将userid添加到signOn的输入中去

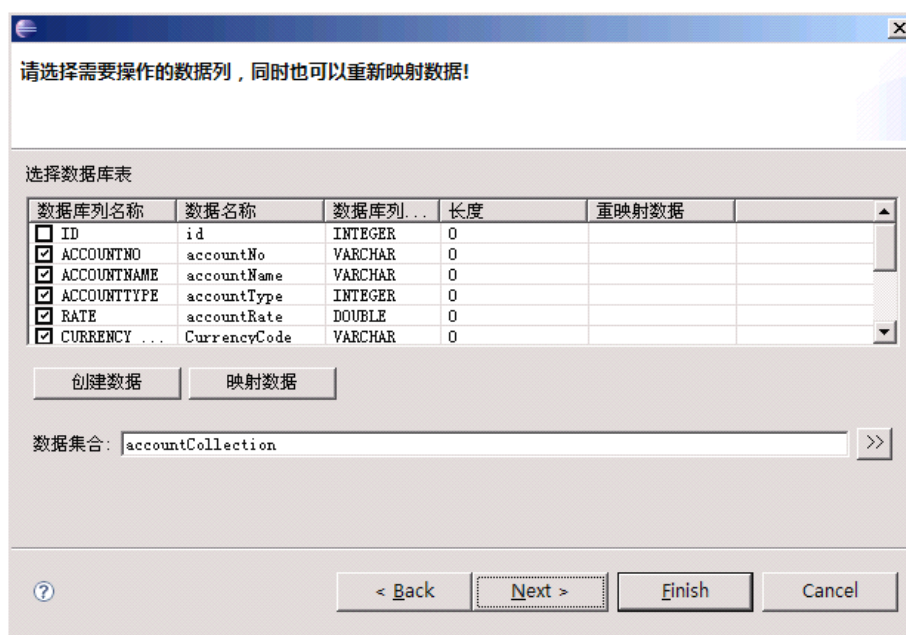
此交易流程的结构和regist基本一致，都是由关键操作引出成功和失败的分支，在失败的分支中设置各种错误信息。本课程的所有交易基本上都采取这样的形式。

- 首先使用“**执行SQL语句**”的action，来校验用户注册ID和密码是否正确，其action的具体设置如下：

描述	验证用户ID和密码
事务类型	应用全局事务
数据源	datasource_derby
执行SQL	sqlService
SQL定义	sqldef_login

- 如果用户名密码校验成功的话，应用还需要去查询该用户ID的对应的帐号。可以通过前面定义的accountinfoTable表定义来查询相关的帐户表，首先给流程添加一个“**访问数据库表**”的action，依次选择服务：tableService，accountinfoTable，

datasource_derby，然后选择访问方式为：查询满足条件的所有记录，接下来选择查询出来的数据映射，和数据集合如下所示：



注：这里的ID必须去掉勾，因为查出来的数据项要和数据集合accountCollection中的数据项一一对应起来。

然后设置查询条件为ACCOUNTINFO.ID = \$id;

注：可能在编辑访问数据库表时找不到accountinfoTable的服务，此时只需要将customerManager.biz关闭，再重开即可以。

SQL条件定义.

SQL条件定义

SQL:

列名称	操作关系	对应数据
ACCOUNTINFO.ID	=	\$id

操作符 列 操作符 数据域/固定值

 ACCOUNTINFO.ID \$id

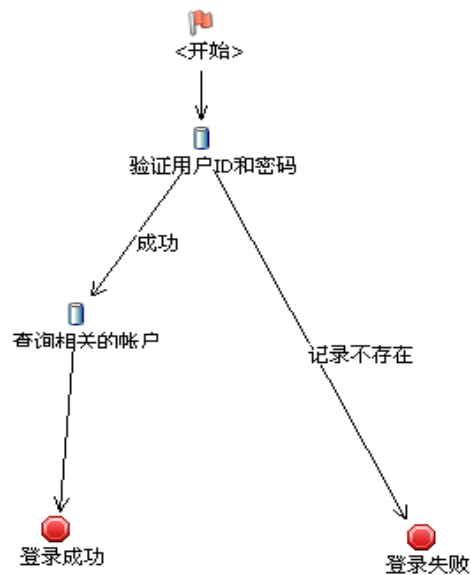
自定义条件:

注：这里没有为查询关联帐户设置出错流转，因为我们假定只要用户存在，查询关联帐户就一定会出错。

在业务逻辑分组编辑器中，定义好 accountinfoTable 服务后。在 ddesignFiles/bizs/Ebankbizs/services.xml 中增添如下内容：

```
<TableDefine id="accountinfoTable"
implClass="com.ecc.emp.jdbc.table.TableDefine" tableName="ACCOUNTINFO"
schema="EBANK">
  <column implClass="com.ecc.emp.jdbc.table.TableColumn" dataName="id"
columnName="ID" columnType="INTEGER"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="accountNo" columnName="ACCOUNTNO" columnType="VARCHAR"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="accountName" columnName="ACCOUNTNAME" columnType="VARCHAR"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="accountType" columnName="ACCOUNTTYPE" columnType="INTEGER"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="accountRate" columnName="ACCONTRATE" columnType="DOUBLE"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="CurrencyCode" columnName="CURRENCYCODE" columnType="VARCHAR"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="openDate" columnName="OPENDATE" columnType="VARCHAR"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="accountBalance" columnName="ACCOUNTBALANCE" columnType="DECIMAL"/>
  <column implClass="com.ecc.emp.jdbc.table.TableColumn"
dataName="freezeBalance" columnName="FREEZEBALANCE" columnType="DECIMAL"/>
</TableDefine>
```

最后具体流程图如下：



signOn的流程定义好后，customerManager.biz中增加如下内容：

```

<operation id="signOn" name="签到">
  <input>
    <field id="userid"/>
    <field id="password"/>
  </input>
  <flow>
    <action id="StartAction0"
implClass="com.ecc.emp.flow.EMPStartAction">
      <transition dest="SQLExecAction0"/>
    </action>
    <action id="EndAction0" result="success"
implClass="com.ecc.emp.flow.EMPEndAction" label="登录成功"/>
    <action id="SQLExecAction0"
implClass="com.ecc.emp.jdbc.sql.SQLExecAction" label="验证用户ID和密码"
sqlService="sqlService" refSQL="sqldef_login" transactionType="TRX_REQUIRED"
dataSource="datasource_derby">
      <transition dest="EndAction1" condition="$retValue='2'"/>
      <transition dest="TableAction2" condition="$retValue='0'"/>
    </action>
    <action id="TableAction2" dataSource="datasource_derby"
transactionType="TRX_REQUIRED" condition="WHERE ACCOUNTINFO.ID=$id; "
columns="ACCOUNTINFO.ACCOUNTNO;ACCOUNTINFO.ACCOUNTNAME;ACCOUNTINFO.ACCOUNTTYPE;
ACCOUNTINFO.ACCOUNTRATE;ACCOUNTINFO.CURRENCYCODE;ACCOUNTINFO.OPENDATE;ACCOUNTIN
FO.ACCOUNTBALANCE;ACCOUNTINFO.FREEZEBALANCE;" tableService="tableService"
op="enquiry" implClass="com.ecc.emp.jdbc.table.TableAction" label="查询相关的帐户"
tableDefine="accountinfoTable" iCollName="accountCollection">
      <transition dest="EndAction0"/>
    </action>
    <action id="EndAction1" result="failed"
implClass="com.ecc.emp.flow.EMPEndAction" label="登录失败"/>
  </flow>
</operation>
  
```


请求页面的表单中包含userid和password的输入框，在表单外还有一个ctp:URL超级链接组件，其href属性为regist.do，表示该链接可以转到regist.do进行注册新用户的操作。

响应页面showCustomerInfo.jsp:

```
...
<%@taglib uri="/WEB-INF/c.tld" prefix="c" %>
<%@taglib uri="/WEB-INF/emp.tld" prefix="emp" %>
...
<table>
  <tr><td colspan="2">客户注册ID:</td><td colspan="2"><ctp:text name="userid"
  dataName="userid"/></td></tr>
  <tr>
    <td>人员名称:</td><td><ctp:text name="username"
    dataName="username"/></td>
    <td>电子邮件:</td><td><ctp:text name="email" dataName="email"/></td>
  </tr>
</table>
<table>
  <tr>
    <td>帐号</td><td>帐号名称</td><td>帐号余额</td>
  </tr>
  <c:forEach var="acctItem" items="${context['accountCollection']}"
  varStatus="status">
    <tr>
      <td align="center">
        <c:set var="accNo" value="${acctItem['accountNo']}"
        scope="request"/>
        <emp:URL contentDivId="div0"
        href="<%= "queryAccount.do?accountNo="+ (String) request.getAttribute("accNo") %>"
        label="<%= (String) request.getAttribute("accNo") %>" />
      </td>
      <td align="center"><c:out value="${acctItem['accountName']}" /></td>
      <td align="center"><c:out
        value="${acctItem['accountBalance']}" /></td>
    </tr>
  </c:forEach>
</table>
.....
```

上半部分使用ctp:text显示客户信息。为了美观起见，可以用HTML表格将它们规划起来。

下半部分显示客户的关联帐户信息，在这里我们的是将JSTL标签库和emp标签库进行混合使用来用表格的形式展现关联的帐户信息。如果页面上需要使用这两个标签库就一定要添加这两个标签库的.tld文件。有关emp标签库的使用，可以查阅文档：EMP平台技术手册。有关JSTL标签库的使用可以查阅sun提供的相关文档和API。

这里需要申明的是，上述的页面源码使用了表达语言（EL）。因为EL是JSP2.0的新特性，所以只能在支持JSP2.0的应用服务器上才能正确编译出来。另外需要说明的是emp标签暂时是不支持EL语言的，所以上面的例子中给出了一个如何往<emp:URL/>标签和EL数据进行交互的方式。

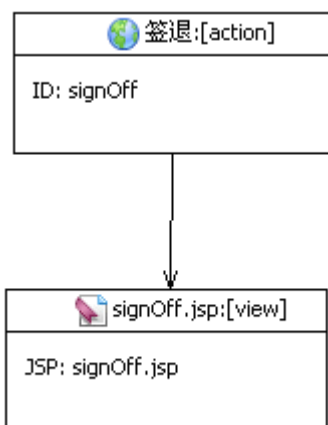
部署项目、启动服务器，查看运行结果。

5.2.2. 签退交易 **signOff: endSession** 控制器

EMP内置的与session相关的控制器一共有两个：session和endSession。session前面已经用到过，用于创建session。而endSession则是用来销毁session，一般用于注销或签退操作。接下来我们使用endSession控制器来制作签退交易。

signOff的业务逻辑流程可以什么都不做。

signOff的mvc模型，控制器类型为endSession：



部署项目、启动服务器，查看运行结果。

将signOff这个action定义在signOn.mvc这个文件中，signOn.mvc增添的内容如下所示：

```
<action id="signOff" label="签退" type="endSession">
    <jspView id="signOff.jsp" url="signOff.jsp"/>
</action>
```

signOff.jsp的内容如下：

```
签退成功，请重新<ctp:URL href="regist.do" label="登录" />
```

然后在showCustomerInfo.jsp中添加一个链接如下所示：

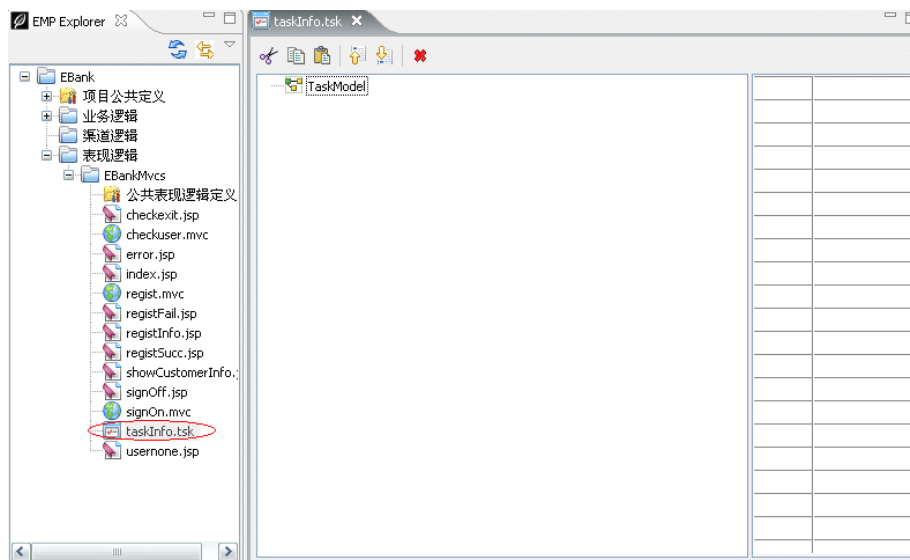
```
<ctp:URL label="签退" href="signOff.do"/>
```

5.2.3. 制作主布局页面和菜单树

随着功能的增多，我们需要一个统一的界面和菜单为每个功能提供入口。EMP提供了Ajax支持的主布局页面和菜单树来实现这一需求。

5.2.3.1. 菜单树的定义

EMP提供了一个用来专门编辑菜单结构的功能，在EMP Explorer视图下，打开表现逻辑分组EBankMvcs下的taskInfo.tsk文件，如下所示：

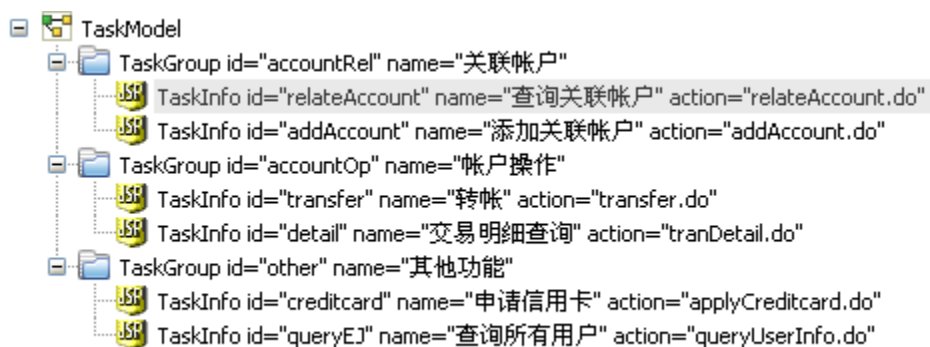


如上图所示，选中TaskModel，点击右键可以选择新建交易分类和交易定义，选中新建的交易分类或者交易定义，通过IDE可以对其属性进行编辑。

交易分类的属性有三个：id是交易分类的唯一标识，name是交易分类的显示名称，dftTaskId是设定当用户选择此菜单项时，工作区默认显示哪个交易定义的action页面。

交易定义的属性有四个：id是交易定义的唯一标识；name为交易定义的显示名称；action是设定点击该交易定义菜单项时，工作区显示的action请求；appendParam是用来给菜单项做前段扩展的，比如说可以在appendParam里写上onclick="alert('***');"

介绍完菜单的定制方式，现在可以按照上面叙述的方式，定制一个如下所示的菜单树：



在designFiles\mvc\Ebankmvc\empServletContext.xml中定义taskInfo.do, 具体内容如下:

```

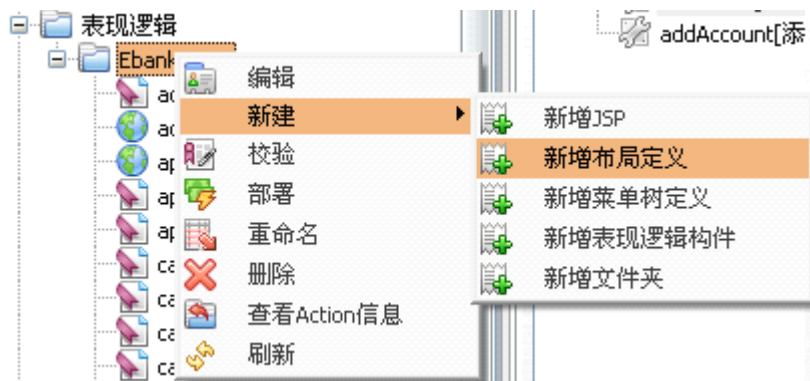
.....
<action id="taskTree" class="com.ecc.emp.web.taskInfo.TaskInfoXMLController" >
    <taskInfoProvider taskInfoFile="WEB-INF/mvc/Ebankmvc/taskInfo.tsk"
class="com.ecc.emp.web.taskInfo.EMPTaskInfoProvider"/>
</action>
</servletContext>
  
```

5.2.3.2. 主布局的使用

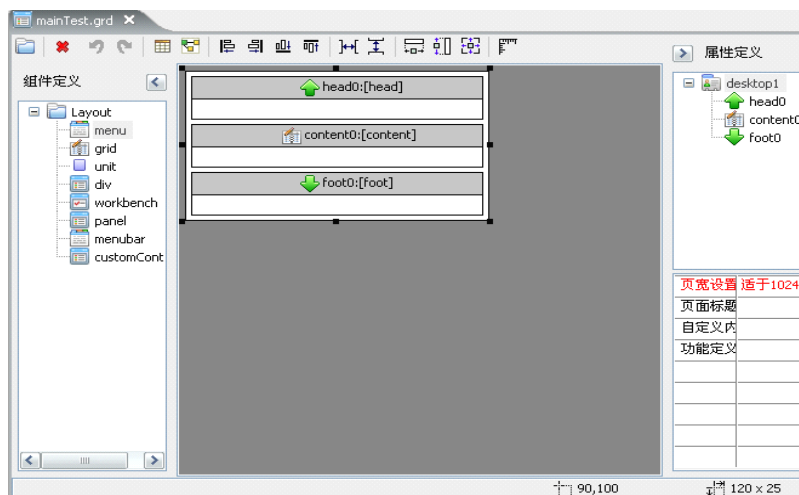
主布局是帮助开发人员设计整个项目操作页面的框架, 让开发工作和页面设计工作有效的分离开。主布局帮助开发人员搭起整个操作页面的大概方案, 开发人员可根据大体的页面展现框架继续进行业务流程的开发设计, 而页面设计人员也可以通过定制CSS样式来灵活的控制整个操作页面的具体展现。

现在来说明一下EMP主布局的具体用法:

- 新建一个主布局的文件, 如下图所示:



主布局定制操作页面如下图所示：

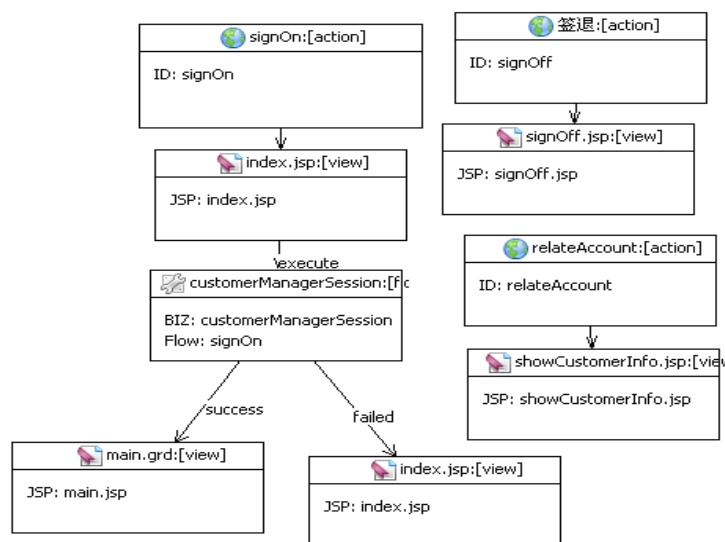


- 主布局主要由四个基本元素组成，即：head0，content0，foot0和menu，其中前三个在操作页面是可见的，而menu是不可见的。另外一个主布局只能添加一个menu进去。当menu添加进去后，主布局定制页面的右边页面会多一个menu0的节点，点中menu0节点，将menu0的“菜单来源”属性配置为：taskTree.do，即刚在empServletContext.xml中定义的action。
- 添加了menu元素在主页面中，并不表示主布局页面能够显示出在上一节中定义的菜单。要让菜单显示还需要往主布局页面中添加menubar，主布局页面是可以添加多个menubar的，并且可以放在head0，content0和foot0中，然后我们点击到menu0节点，可以看到上面有1—5级菜单的属性，因为我们前面只定义了两级的菜单，所以只需要往菜单项添加两个menubar，然后在将的菜单menu0的1，2级菜单分别设置为添加进来的两个menubar，这样菜单就会在主布局设置menubar的位置显示出来。
- 菜单设置好后，菜单控制的操作页面也需要设置，菜单控制的操作页面需要往主布局中添加一个div的元素div0，然后在menu0的“交易显示区域”属性的下拉框中选中div0，则菜单上定义的交易请求都是刷新div0的区域。
- 按照以上的流程可以将整个页面的大体显示方案设计出来，现在我们为这个主布局加一个简单的CSS样式，其添加方法为点击右上角结构树的desktop1节点，并在自定义内容中加入我们添加进来的样式，添加内容如下：

```
<LINK rel="stylesheet" href='<ctp:file fileName="styles/ccb.css"/>' type="text/css"/>
```

5.2.3.3. 修改 mvc 定义

在signOn.mvc中添加一个relatedAccount的action，并把主布局页面main.grd引入。将流程做如下修改。



signOn[action] 修改后内容如下所示：

```

<action id="signOn" sessionContextName="PersonalEBankSessionCtx"
type="session" checkSession="false">
  <jspView id="index.jsp" url="index.jsp"/>
  <refFlow flowId="customerManagerSession" op="signOn">
    <transition dest="index.jsp" condition="$retValue='failed'"/>
    <transition dest="main.jsp" condition="$retValue='success'"/>
  </refFlow>
</action>

```

增加的relateAccount[action] 内容如下所示：

```

<action id="relateAccount" type="normal" checkSession="true">
  <jspView id="showCustomerInfo.jsp" url="showCustomerInfo.jsp"/>
</action>

```

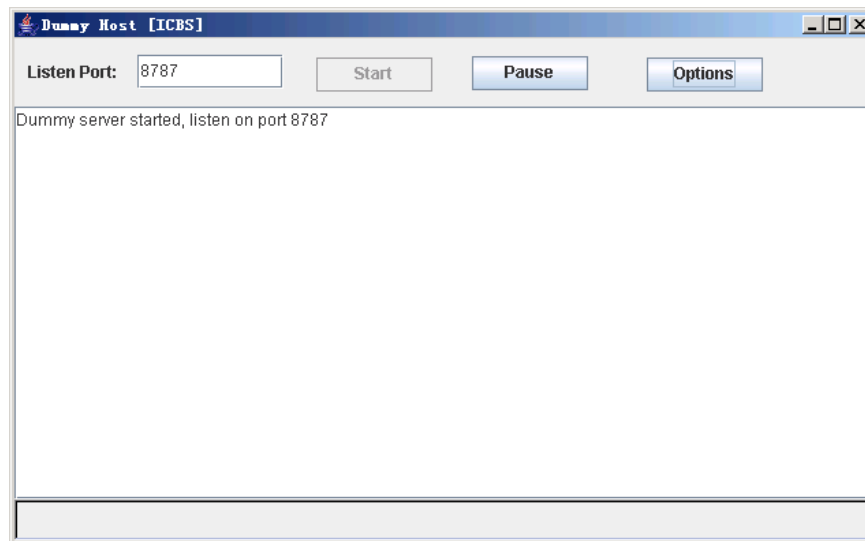
5.3. 课程 7：后台访问处理

课程名称	后台访问处理	课程时间	
课程目标	一、了解后台访问所需要的通信知识（HTTP、TCP/IP、MQ 等等知识点的介绍）		

	二、学习并掌握 TCP/IP 通信协议下的访问配置 三、学习并掌握 8583 协议的数据格式化报文配置 四、了解并掌握其他格式的报文处理配置方式 五、学习并掌握如何处理通信返回后的信息，根据返回的不同内容进行不同逻辑处理
准备工作	完成课程 6

5.3.1. 虚拟主机 Dummy Host 介绍

5.3.1.1. 交易说明



本教程使用的虚拟主机界面如上图所示，只需填写端口号（默认8787）后点击Start就可启动监听。针对本教程的4支主机交易（实际上是3支），分别返回不同的响应报文。

- 查询客户帐户&添加关联帐户（主机交易码0718）：

接受任何19位帐户号，返回一条帐户信息。若帐户号不为19位则通讯出错。

- 转账（主机交易码0719）：

若源帐户号不为19位则通讯出错。第37域作为结果标志域：

目标帐户号不为19位： F

转账金额大于4000： L

转账成功：S

➤ 查询帐户交易明细（主机交易码0720）：

以XML报文形式返回交易明细结果集。若源帐户号不为19位则通讯出错。第37域作为结果标志域：

起始时间格式错：B

结束时间格式错：E

两者皆错：D

查询成功：S

5.3.1.2. 主机报文常识和格式规范

在银行应用系统中，与银行后台主机进行交互是非常重要的操作，银行的应用系统环境几乎均为分布式应用环境，一个系统总是需要与其他系统进行交互。而与后台主机进行帐户处理交互功能是最常见的操作。

访问银行后台主机一般是基于标准的通信协议和报文格式协议基础上来完成交互的。

一般来说，与主机交互的报文总是分为报文头和报文体两部分。

报文头有请求报文头和返回报文头，他们总是规范的，固定长度或者可预先度量长度的，往往是共享的，对所有的交易请求其报文头格式是公共的。主机系统通过报文来识别和放置主机本身所需要的交易处理的公共信息和识别交易，并通过返回报文头来识别交易执行结果和返回的公共信息。

报文体是交易请求本身的数据格式，它是独立的，相互区别的。不同的交易的请求报文和返回报文都是不一样的。

主机通过**报文头+报文体**的方式完整的而且灵活的实现了交易请求交互的高适应性和规范性。

➤ 报文头格式

请求报文头: tranId (4 char) + “#” (5Char)

返回报文头: tranId+ “#” +Status+ “#” (7Char)

➤ 报文体格式

自定义8583格式

5.3.1.3. 定义公共报文头

在IDE Explorer中创建accountManager.biz, 在accountManager.biz中引用serviceId, accountManager.biz中创建请求报文头格式的内容如下:

```
<format id="headReqFmt">
  <document>请求报文头</document>
  <datas packageType="VariousStringFormat">
    <refData len="4" stringType="FixedLenFormat" alignment="right" delimChar="#"
refId="serviceId" padChar="0" nullCheck="true"/>
  </datas>
</format>
```

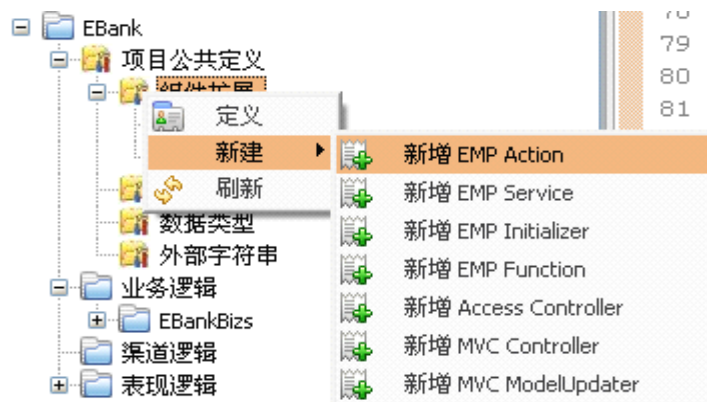
accountManager.biz 中创建返回报文头格式的内容如下:

```
<format id="headRepFmt">
  <document>接收报文头</document>
  <datas packageType="VariousStringFormat">
    <refData len="4" stringType="FixedLenFormat" alignment="right" delimChar="#"
refId="serviceId" padChar="0" nullCheck="true"/>
    <refData len="1" stringType="FixedLenFormat" alignment="none" delimChar="#"
refId="hostResult" nullCheck="true"/>
  </datas>
</format>
```

5.3.2. 创建报文通讯处理的 action

针对报文格式头的特殊要求, 这里需要扩展一个TcpIp通讯访问的action组件, 其扩展步骤如下所示:

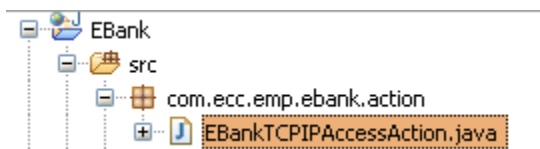
- 选中EBank项目公共定义下的组件扩展, 然后点击右键, 选择新建->新增EMP Action 如下所示:



- 选择新增EMP Action后，弹出Action的信息对话框，按照向导添加相关信息，如下所示：

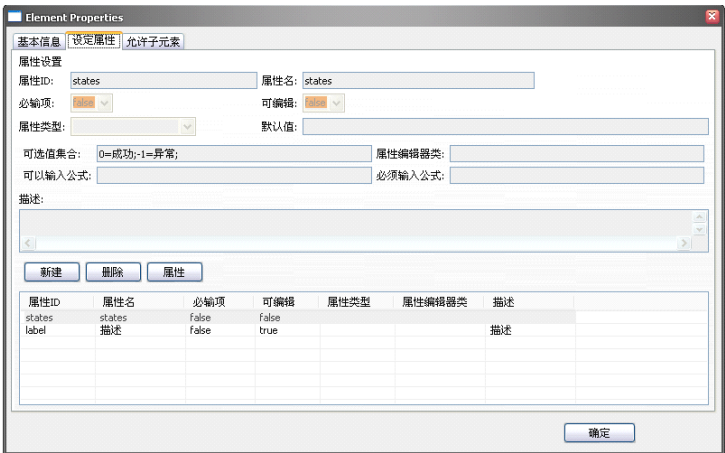


- 点击完成后，该组件顺利被扩展到EMP的IDE中，并在EBank项目中创建了该组件对应的action类，如下所示：



上述的步骤成功成功的在EMPIDE中创建了一个扩展的action，现在需要对该action进行属性的定制，定制action属性的步骤如下所示：

- 打开EBank的项目公共定义的用户扩展定义页签，双击自己新建的action构件，弹出如下对话框：



➤ 点击新建，为该action添加必要的属性，如下所示：

属性ID	属性名	必填项	可编辑	属性类型	属性编辑器类	描述
states	states	false	false	String	com.ecc.ide.edi...	描述
label	描述	false	true	String	com.ecc.ide.edi...	描述
serviceName	通讯服务	true	true	String	com.ecc.ide.edi...	
sendHeadFo...	发送报文头	true	true	String	com.ecc.ide.edi...	
sendFormat...	发送报文体	true	true	String	com.ecc.ide.edi...	
receiveHead...	接收报文头	true	true	String	com.ecc.ide.edi...	
receiveForm...	接收报文体	true	true	String	com.ecc.ide.edi...	
timeOut	延时时间	false	true	int	com.ecc.ide.edi...	

➤ 添加完相关的属性后，点击确定，然后保存EBank的项目公共定义，即该组件的配置基本完成剩下的工作就是扩展EBankTCPIPAccessAction这个类，为该类添加对应的属性，并给上set方法，然后扩展excute这个方法。该类的部分源代码如下所示：

```
...
    private String serviceName;

    private String sendFormatName = "sendHostFormat";
    private String receiveFormatName = "receiveFormat";
    private String sendHeadFormatName = "sendHeadHostFormat";
    private String receiveHeadFormatName = "receiveHeadFormatName";
    private int timeOut = -1;

    //报文唯一标识域
    private String identityField;
    public String execute(Context context)
    {
        // 调用TCP/IP通讯服务
        TCPIPService tcpipService = null;
        long beginTime=System.currentTimeMillis();

        try {
            tcpipService =(TCPIPService) context.getService( serviceName );
        }catch (Exception e){
            EMPLog.log(EMPConstance.EMP_TCPIP, EMPLog.ERROR, 0,
            "TCPIPAccessAction: Fail to get TCPIPService." , e);
            return "2";
        }
    }
}
```

```

// 报文的组装与接收
try {
    FormatElement reqHeadFmt = (FormatElement)
context.getFormat(sendHeadFormatName);
    Object headPkg=reqHeadFmt.format(context);
    FormatElement reqfmt = (FormatElement)
context.getFormat(sendFormatName);
    Object bodyPkg = reqfmt.format(context);

    Object sendPkg = null;
    byte[] sendBytes= null;
    byte[] new_headPkg = null;
    byte[] new_bodyPkg = null;

    if (headPkg instanceof byte[]) {
        String headStr = new String(((byte[]) headPkg));
        new_headPkg = headStr.getBytes();
    }else{
        new_headPkg = ((String) headPkg).getBytes();
    }
    if(bodyPkg instanceof byte[]){
        String bodyStr = new String((byte[])bodyPkg);
        new_bodyPkg = bodyStr.getBytes();
    }else{
        new_bodyPkg = ((String) bodyPkg).getBytes();
    }

    sendBytes = new byte[new_headPkg.length+new_bodyPkg.length];
    System.arraycopy(new_headPkg, 0, sendBytes, 0, new_headPkg.length);
    System.arraycopy(new_bodyPkg, 0, sendBytes, new_headPkg.length,
new_bodyPkg.length);

    byte[] repMsg = null;

    repMsg =tcpipService.sendAndWait(null, sendBytes, timeOut );

    byte[] repHeadPkg = new byte[7];
    byte[] repBodyPkg = new byte[repMsg.length-repHeadPkg.length];

    System.arraycopy(repMsg, 0, repHeadPkg, 0, repHeadPkg.length);
    System.arraycopy(repMsg, repHeadPkg.length, repBodyPkg, 0,
repBodyPkg.length);

    FormatElement repHeadFmt = (FormatElement)
context.getFormat(receiveHeadFormatName);
    FormatElement repFmt = (FormatElement)
context.getFormat(receiveFormatName);

    String testStr = new String((byte[])repHeadPkg);
    repHeadFmt.unFormat(testStr, context);
    repFmt.unFormat(repBodyPkg, context);
} catch (Exception tempE){
    EMPLog.log(EMPConstance.EMP_TCPIP, EMPLog.ERROR, 0,
"TCPIPCp935AccessAction: Fail to sendAnd wait to host.", tempE );
    return "1";
}
return "0";
}
...

```


5.3.3. 查询客户帐户交易 queryAccount

我们将与发送主机有关（即与帐户操作有关）的交易都放在一个新的biz业务逻辑中。在其中引用需要发送主机的数据域。具体请参照项目设计文档。

5.3.3.1. 交易数据引用

在业务逻辑分组设定编辑器中引用帐户相关数据：

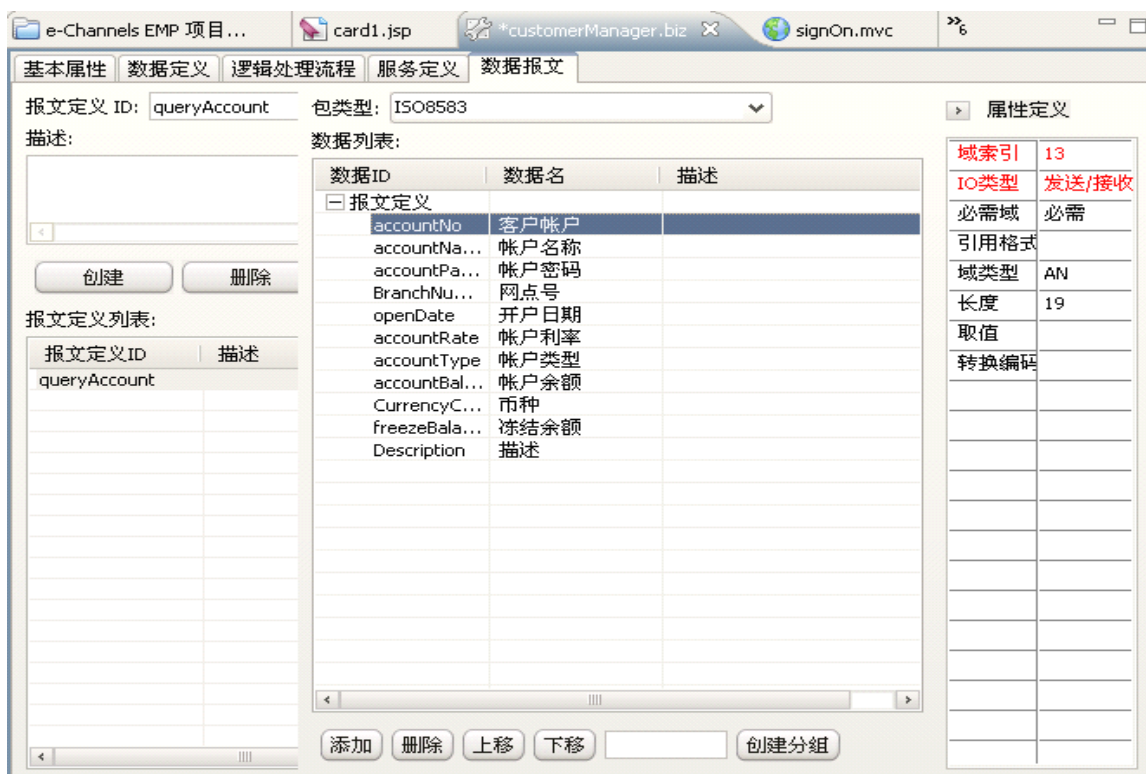
accountNo
BranchNumber
accountPassword
accountName
openDate
accountRate
accountType
accountBalance
freezeBalance
Description

以及日志相关数据：

ejCollection
tranId
tranStatus
firstAccount
secondAccount
comment
CurrencyCode
tranAmount

5.3.3.2. 配置数据报文

在accountManager.biz的数据报文分页中添加一个名为queryAccount的报文，选择包类型为ISO8583。按照下图引用如下数据域（即帐户信息分组下的所有数据域）：



首先选中根节点“报文定义”，进行ISO8583总体设置：编码为ASCII，报文类型设置为查询帐户交易码0718。

然后依次对每一个数据域进行属性设置，具体属性参见下表。

数据域	域索引	IO类型	必需域	域类型	长度	取值
accountNo	13	发送/接收	必需	AN	19	
BranchNumber	20	发送		AN	5	10001
accountPassword	15	发送		AN	6	
accountName	14	接收		AN	20	
openDate	21	接收		AN	8	
accountRate	22	接收		AN	13	
accountType	23	接收		AN	1	
accountBalance	24	接收		AN	20	
CurrencyCode	25	接收		AN	3	
freezeBalance	26	接收		AN	20	
Description	27	接收		AN	128	

最终生成的报文定义XML如下：

```
<fmtDef id="queryAccount" isBin="true">
  <ISO8583PkgFmt msgType="0718" codeSet="ASCII">
    <ISO8583Field dataName="accountNo" fieldType="AN" fieldState="M"
      fieldIdx="13" IOType="IO" fieldLength="19"/>
    <ISO8583Field dataName="BranchNumber" fieldType="AN"
      fieldState="NotM" fieldValue="10001" fieldIdx="20" IOType="O" fieldLength="5"/>
    <ISO8583Field dataName="accountPassword" fieldType="AN"
      fieldState="NotM" fieldIdx="15" IOType="O" fieldLength="6"/>
  </ISO8583PkgFmt>
</fmtDef>
```

```

        <ISO8583Field dataName="accountName" fieldType="AN" fieldIdx="14"
IOType="I" fieldLength="20"/>
        <ISO8583Field dataName="openDate" fieldType="AN" fieldIdx="21"
IOType="I" fieldLength="8"/>
        <ISO8583Field dataName="accountRate" fieldType="AN" fieldIdx="22"
IOType="I" fieldLength="13"/>
        <ISO8583Field dataName="accountType" fieldType="AN" fieldIdx="23"
IOType="I" fieldLength="1"/>
        <ISO8583Field dataName="accountBalance" fieldType="AN" fieldIdx="24"
IOType="I" fieldLength="20"/>
        <ISO8583Field dataName="CurrencyCode" fieldType="AN" fieldIdx="25"
IOType="I" fieldLength="3"/>
        <ISO8583Field dataName="freezeBalance" fieldType="AN" fieldIdx="26"
IOType="I" fieldLength="20"/>
        <ISO8583Field dataName="Description" fieldType="AN" fieldIdx="27"
IOType="O" fieldLength="128"/>
    </ISO8583PkgFmt>
</fmtDef>

```

以上是最终生成的报文格式，而我们在accountManager.biz中定义编辑的报文内容如下所示：

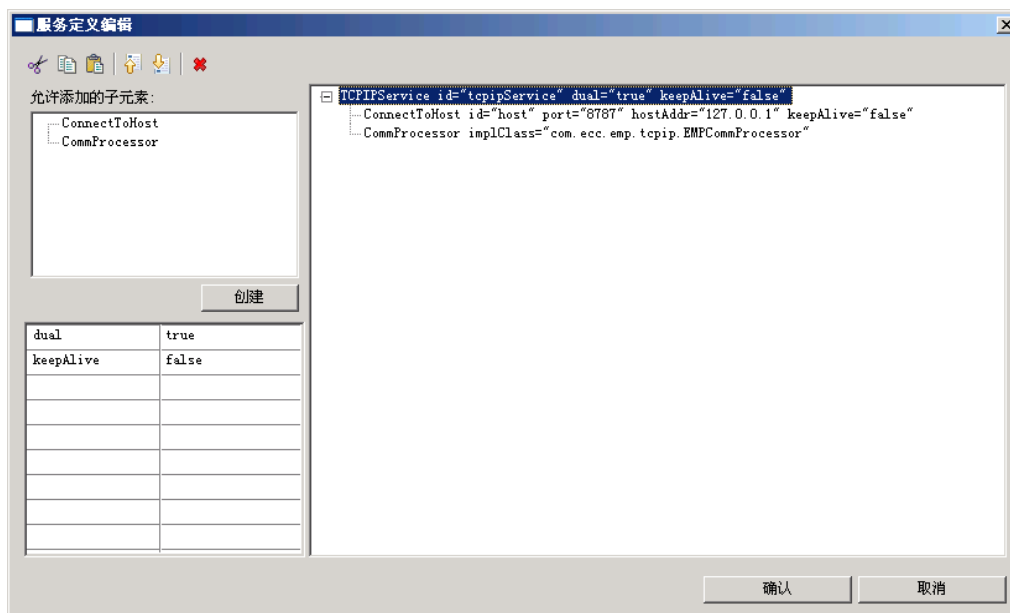
```

<format id="queryAccount">
    <document>查询帐号交易报文</document>
    <datas msgType="0718" codeSet="ASCII" packageType="ISO8583">
        <refData fieldType="AN" refId="accountNo" fieldState="M" fieldIdx="13"
fieldLength="19" IOType="IO"/>
        <refData fieldType="AN" refId="BranchNumber" fieldValue="10001" fieldState="NotM"
fieldIdx="20" fieldLength="5" IOType="O"/>
        <refData fieldType="AN" refId="accountPassword" fieldState="NotM" fieldIdx="15"
fieldLength="6" IOType="O"/>
        <refData fieldType="AN" refId="accountName" fieldIdx="14" fieldLength="20"
IOType="I"/>
        <refData fieldType="AN" refId="openDate" fieldIdx="21" fieldLength="8" IOType="I"/>
        <refData fieldType="AN" refId="accountRate" fieldIdx="22" fieldLength="13"
IOType="I"/>
        <refData fieldType="AN" refId="accountType" fieldIdx="23" fieldLength="1" IOType="I"/>
        <refData fieldType="AN" refId="accountBalance" fieldIdx="24" fieldLength="20"
IOType="I"/>
        <refData fieldType="AN" refId="CurrencyCode" fieldIdx="25" fieldLength="3"
IOType="I"/>
        <refData fieldType="AN" refId="freezeBalance" fieldIdx="26" fieldLength="20"
IOType="I"/>
        <refData fieldType="AN" refId="Description" fieldIdx="27" fieldLength="128" IOType="I"
"/>
    </datas>
</format>

```

5.3.3.3. 配置 TCPIP 通信服务

在accountManager.biz的服务定义分页中创建TCPIP通信的私有服务，id为tcpipService，类型为TCPIPService。点击创建后在弹出的服务定义编辑器中配置服务的参数：

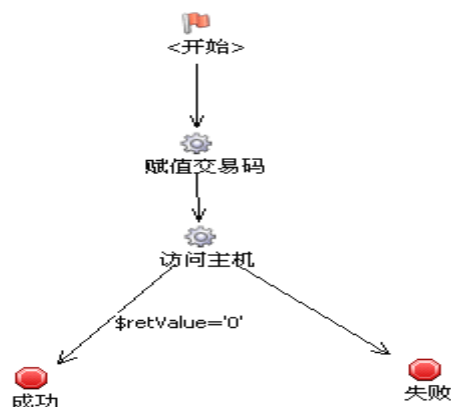


这个编辑器是XML编辑形式，可以为节点添加子节点、设置参数等。这里需要添加ConnectToHost（连接主机）和CommProcessor（通讯协议处理器）两个子节点。参数配置如下。

```
<TCPIPService id="tcpipService" dual="true" keepAlive="false">
  <ConnectToHost id="host" port="8787" hostAddr="127.0.0.1" keepAlive="false">
    <CommProcessor implClass="com.ecc.emp.tcpip.EMPCommProcessor"/>
  </ConnectToHost>
</TCPIPService>
```

5.3.3.4. 流程定义

创建queryAccount交易，引用输入数据accountNo。流程如下：



首先看主要的“访问”TCPIP主机访问步骤：这个步骤没有向导，将扩展的action属性中的

值按要求配进去即可，如下所示：

描述	访问主机
通讯服务	tcpipService
发送报文	headReqFmt
发送报文	queryAccount
接收报文	headRepFmt
接收报文	queryAccount
延时时间	

该交易流程在accountManager.biz中生成的文件内容如下所示：

```
<operation id="queryAccount" name="查询帐户信息">
  <input>
    <field id="accountNo"/>
  </input>
  <flow>
    <action id="StartAction0"
implClass="com.ecc.emp.flow.EMPStartAction">
      <transition dest="SetValueAction0"/>
    </action>
    <action id="EndAction0" result="success"
implClass="com.ecc.emp.flow.EMPEndAction" label="成功"/>
    <action id="EBankTCPIPAccessAction0"
sendHeadFormatName="headReqFmt"
implClass="com.ecc.emp.ebank.action.EBankTCPIPAccessAction" label="访问主机"
sendFormatName="queryAccount" serviceName="tcpipService"
receiveFormatName="queryAccount" receiveHeadFormatName="headRepFmt">
      <transition dest="EndAction0" condition="$retValue='0'"/>
      <transition dest="EndAction1"/>
    </action>
    <action id="EndAction1" result="failed"
implClass="com.ecc.emp.flow.EMPEndAction" label="失败"/>
    <action id="SetValueAction0" dataName="serviceId"
implClass="com.ecc.emp.action.SetValueAction" label="赋值交易码"
dataValue="0718">
      <transition dest="EBankTCPIPAccessAction0"/>
    </action>
  </flow>
</operation>
```

5.3.3.5. mvc 定义和 JSP 页面

首先需要修改之前的 showCustomerInfo.jsp 作为请求页面。为 accountNo 那一列的 ctp:tablecolumn 设置属性 selfAction 为 true，表示该表列是超级链接；formDataName 为 accountNo 表示提交到 accountNo 数据域（虽然同名但也要设置）；action 为 queryAccount.do。这样该表列在浏览器中就会解释为提交该数据到 queryAccount.do 的超级链接。具体内容如下：

```
...
<table>
```

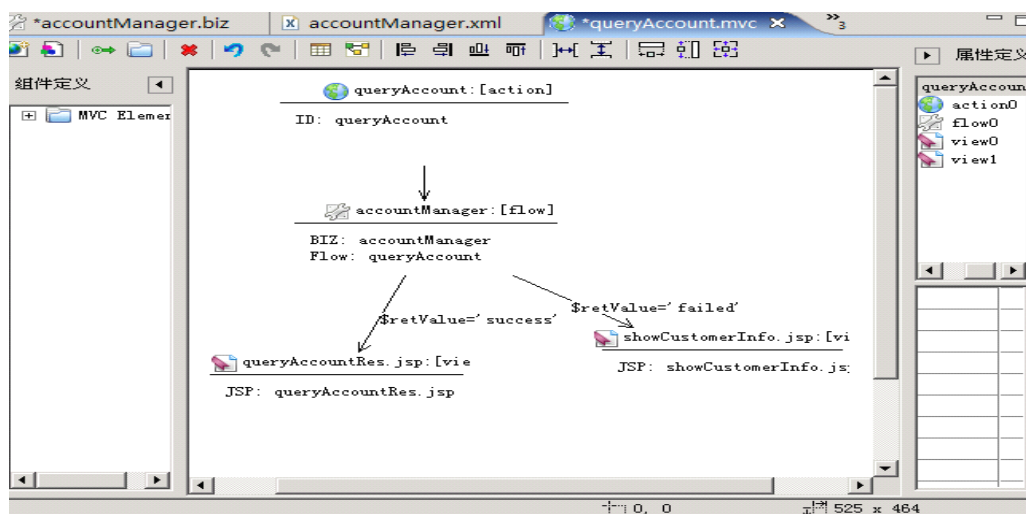
```

<tr><td colspan="2">客户注册ID:</td><td colspan="2"><ctp:text name="userid"
dataName="userid"/></td></tr>
<tr>
<td>人员名称: </td><td><ctp:text name="username"
dataName="username"/></td>
<td>电子邮件: </td><td><ctp:text name="email" dataName="email"/></td>
</tr>
</table>
<ctp:table name="accountInfo" iCollName="accountCollection"
needTableTitle="true" isLayoutContent="true" contentDivId="div0" >
<ctp:tableColumn label="帐号" dataName="accountNo" selfAction="true"
formFieldType="none" formDataName="accountNo" action="queryAccount.do"/>
<ctp:tableColumn label="帐号名称" dataName="accountName"/>
<ctp:tableColumn label="帐号余额" dataName="accountBalance"/>
</ctp:table>
<ctp:URL label="签退" href="signOff.do"/>
...

```

接下来创建queryAccount.mvc。将action直接指向flow（因为此交易的请求页面是showCustomerInfo.jsp），若成功则转向响应页面queryAccountRes.jsp，若失败则返回showCustomerInfo.jsp显示错误信息。

注：在对showCustomerInfo.jsp的修改设计中对ctp:table的isLayoutContent属性一定要设置为false，否则帐户号码这一列产生的连接会无法跳转，步骤是先选中ctp:table组件，然后再属性框中将toBeForm属性设为true，这时isLayoutContent属性会显示到属性框里，将其设为false，然后再将toBeForm属性设为false。



queryAccount.mvc 文件的内容如下所示：

```

<action id="queryAccount" type="normal" checkSession="false">
<refFlow flowId="accountManager" op="queryAccount">
<transition dest="queryAccountRes.jsp"
condition="$retValue='success'"/>
<transition dest="showCustomerInfo.jsp"
condition="$retValue='failed'"/>
</refFlow>

```

```
</action>
```

queryAccountRes.jsp的内容如下:

```
<TABLE height="330" width="255" cellspacing="1" cellpadding="1" border="1">
  <TR>
    <TD height="30" width="174"><ctp:label name="Label10"
dataName="accountNo" text="客户帐户"/></TD>
    <TD width="80"><ctp:text name="accountNo" dataName="accountNo"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label13" dataName="accountName" text="帐
户名称"/></TD>
    <TD><ctp:text name="accountName" dataName="accountName"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label14" dataName="openDate" text="开户日
期"/></TD>
    <TD><ctp:text name="openDate" dataName="openDate"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label15" dataName="accountRate" text="帐
户利率"/></TD>
    <TD><ctp:text name="accountRate" dataName="accountRate"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label16" dataName="accountType" text="帐
户类型"/></TD>
    <TD><ctp:text name="accountType" dataName="accountType"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label18" dataName="CurrencyCode" text="
币种"/></TD>
    <TD><ctp:text name="CurrencyCode" dataName="CurrencyCode"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label17" dataName="accountBalance"
text="帐户余额"/></TD>
    <TD><ctp:text name="accountBalance" dataName="accountBalance"/></TD>
  </TR>
  <TR>
    <TD height="30"><ctp:label name="Label19" dataName="freezeBalance" text="
冻结余额"/></TD>
    <TD><ctp:text name="freezeBalance" dataName="freezeBalance"/></TD>
  </TR>
</TABLE>
```

部署、启动服务器和虚拟主机，查看运行结果。

5.4. 课程 8：交易实战练习

5.4.1. 练习一：添加客户账户

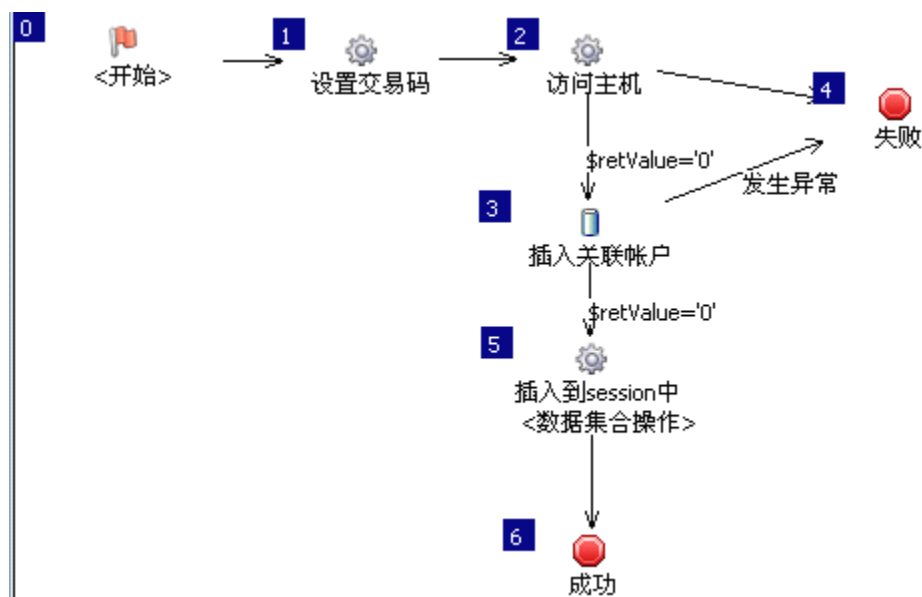
交易功能：添加客户账户是将已注册客户的帐户信息添加到客户信息中，已方便客户汇总

其帐户信息，迅速掌握其资产情况。

本交易用到的数据、报文和服务都和queryAccount一样，所以不需要再次定义。只是在accountManager.biz中还需从“应用节点”引用一个accountCollection。

5.4.1.1. 流程定义

创建addAccount的流程，引用输入数据accountNo和accountPassword。



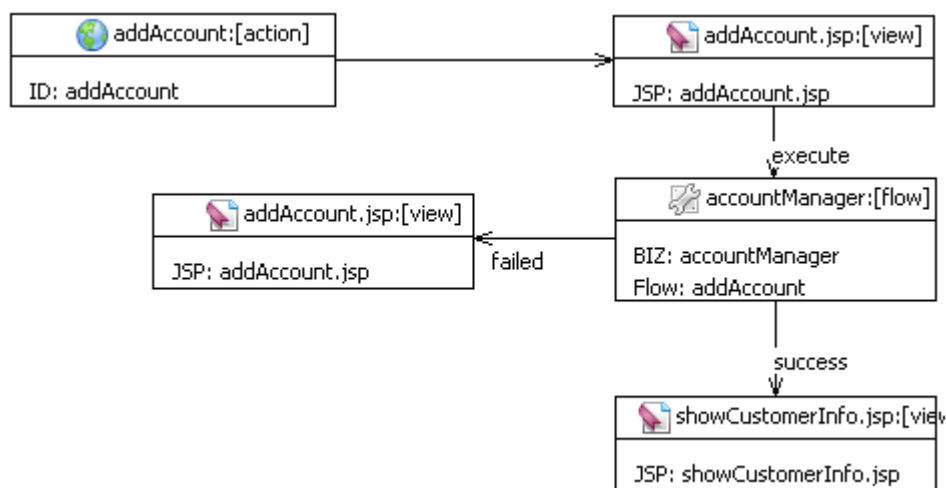
[2] 访问主机步骤和查询帐户交易完全一样。

[3] 将主机返回的帐户信息插入ACCOUNTINFO表。若插入失败则说明已存在该帐户，转到[4]。

[5] 数据集合操作步骤将帐户信息插入session中的accountCollection。

5.4.1.2. mvc 定义

addAccount.mvc如下图，请求页面和失败响应页面为addAccount.jsp，成功响应页面为showCustomerInfo.jsp，引用addAccount流程，使用normal控制器：



5.4.1.3. JSP 页面

addAccount.jsp 的内容如下:

```

<ctp:form name="form" isLayoutContent="true" action="addAccount.do"
method="post">
  <table>
    <tr>
      <td>帐户: </td>
      <td><ctp:input dataName="accountNo" type="text" /></td>
    </tr>
    <tr>
      <td>帐户密码</td>
      <td><ctp:input dataName="accountPassword" type="text" /></td>
    </tr>
  </table>
  <ctp:input type="submit" value="添加" />
</ctp:form>

```

5.4.2. 练习二：转账交易 transfer

5.4.2.1. 交易数据引用

在accountManager.bize 的编辑器中引用转账相关数据:

```

destAccount
destAccountName
transferAmonunt
transferDescription

```

5.4.2.2. 配置数据报文

按照下表配置名为transfer的ISO8583转账报文。

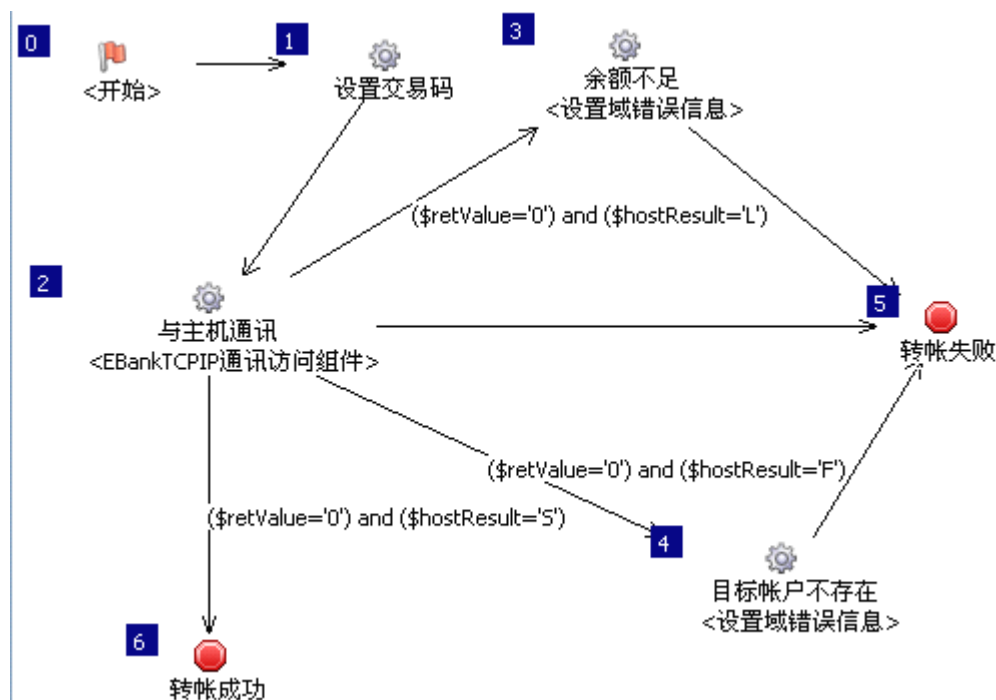
数据域	域索引	IO类型	必需域	域类型	长度	取值
accountNo	13	发送/接收	必需	AN	19	
accountPassword	15	发送		AN	6	
BranchNumber	20	发送		AN	5	10001
destAccount	30	发送		AN	19	
destAccountName	32	发送		AN	20	
transferAmount	35	发送		AN	20	
transferDescription	36	发送		LLVAR		
hostResult	37	接收		AN	1	

在accountManager.biz中报文定义好后的内容如下所示：

```
<format id="transfer">
  <document>转账报文</document>
  <datas msgType="0719" codeSet="ASCII" packageType="ISO8583">
    <refData fieldType="AN" refId="accountNo" fieldState="M"
fieldIdx="13" fieldLength="19" IOType="IO"/>
    <refData fieldType="AN" refId="accountPassword" fieldState="M"
fieldIdx="15" fieldLength="6" IOType="O"/>
    <refData fieldType="AN" refId="BranchNumber" fieldValue="10001"
fieldIdx="20" fieldLength="5" IOType="O"/>
    <refData fieldType="AN" refId="destAccount" fieldIdx="30"
fieldLength="19" IOType="O"/>
    <refData fieldType="AN" refId="destAccountName" fieldIdx="32"
fieldLength="20" IOType="O"/>
    <refData fieldType="AN" refId="transferAmount" fieldIdx="35"
fieldLength="20" IOType="O"/>
    <refData fieldType="LLVAR" refId="transferDescription"
fieldIdx="36" IOType="O"/>
    <refData fieldType="AN" refId="hostResult" fieldIdx="37"
fieldLength="1" IOType="I"/>
  </datas>
</format>
```

5.4.2.3. 流程定义

定义名为transfer的交易流程，输入数据为accountNo, accountPassword, destAccount, destAccountName, transferAmount, transferDescription。



[2]为TCPIP主机访问步骤，使用transfer报文。根据主机响应的不同有以下几种情况：在报文中有一个hostResult数据域，虚拟主机将处理结果标志写在这个数据域中，S代表成功，L代表余额不足，F代表目标帐户不存在。要对它们分别设置流转和不同的出错信息。

如果返回值为0且\$hostResult='S'（成功），则转到[6]成功分支；

如果返回值为0且\$hostResult='F'（帐户不存在），则转到[4]设置域错误信息；

如果返回值为0且\$hostResult='L'（余额不足），则转到[3]设置域错误信息；

否则直接转向[5]失败分支。

[5]失败结束步骤设置错误信息为“主机交易失败!”。

这里提到了设置域错误信息的操作，需填写显示的数据域名，和显示的错误信息，余额不足的域错误信息设置，如下图：

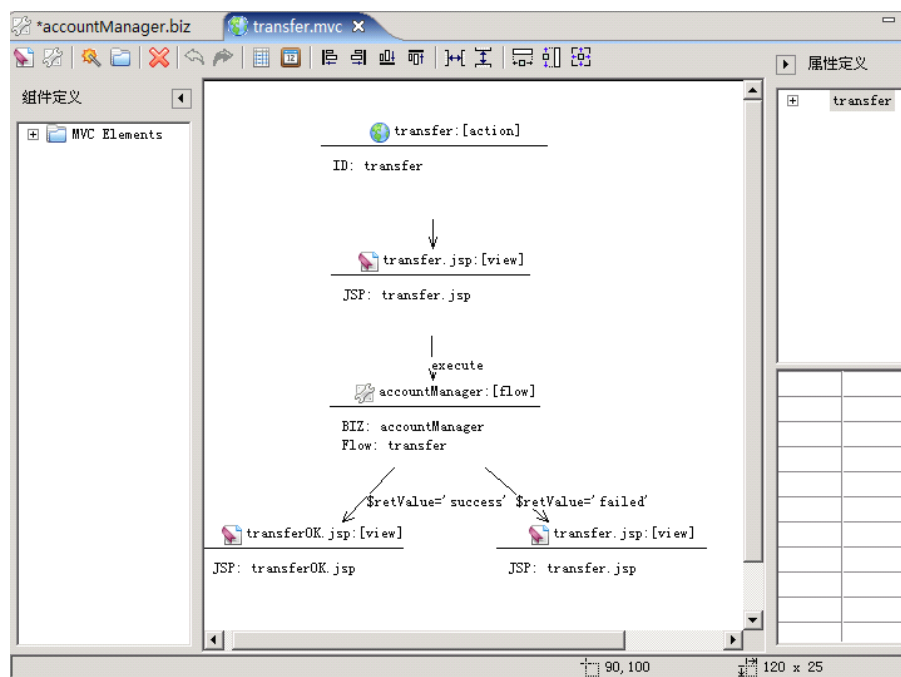
描述	
数据域列	transferAmount
错误信息	余额不足
清空	true

交易流程定义好后，accountManager.biz增添如下内容：

```
<operation id="transfer" name="交易流程">
  <document/>
  <flow name="flow1" x="10" height="400" width="500" y="10">
    <StartAction name="StartAction0" x="8" height="28" width="84" y="22">
      <transition name="transition0" dest="SetValueAction0"/>
    </StartAction>
    <SetValueAction name="SetValueAction0" dataName="serviceId" height="32" x="137"
label="设置交易码" width="99" dataValue="0719" y="21">
      <transition name="transition1" dest="EBankTCPIPAccessAction0"/>
    </SetValueAction>
    <EBankTCPIPAccessAction name="EBankTCPIPAccessAction0" height="50" x="14" label="
与主机通讯" width="168" y="142">
      <transition name="transition2" dest="SetFieldErrorMsgAction0"
condition="($retValue='0') and ($hostResult='L')"/>
      <transition name="transition3" dest="SetFieldErrorMsgAction1"
condition="($retValue='0') and ($hostResult='F')"/>
      <transition name="transition4" dest="EndAction0"/>
      <transition name="transition7" dest="EndAction1" condition="($retValue='0')
and ($hostResult='S')"/>
    </EBankTCPIPAccessAction>
    <SetFieldErrorMsgAction name="SetFieldErrorMsgAction1" width="121"
needClear="true" errorMsgs="目标帐户不存在" y="272" label="目标帐户不存在" x="327" height="54"
fieldNames="destAccount">
      <transition name="transition6" dest="EndAction0"/>
    </SetFieldErrorMsgAction>
    <SetFieldErrorMsgAction name="SetFieldErrorMsgAction0" width="113"
needClear="true" errorMsgs="余额不足" y="16" label="余额不足" x="250" height="52"
fieldNames="transferAmount">
      <transition name="transition5" dest="EndAction0"/>
    </SetFieldErrorMsgAction>
    <EndAction name="EndAction0" result="failed" x="432" height="32" label="转帐失败"
width="65" y="152"/>
    <EndAction name="EndAction1" result="success" height="32" x="62" label="转帐成功"
width="75" y="321"/>
  </flow>
  <input/>
  <output/>
</operation>
```

5.4.2.4. mvc 定义

transfer.mvc 如下图，请求页面和失败响应页面为 transfer.jsp，成功响应页面为 transferOK.jsp，引用transfer流程，使用normal控制器：



文件transfer.mvc定义好的内容显示如下：

```
<EMPMVC name="transfer" x="0" height="464" width="525" y="0">
  <action id="transfer" name="action0" x="110" height="80" width="180"
  type="normal" checkSession="false" y="10">
    <reference name="ref0" dest="view0"/>
  </action>
  <view id="transfer.jsp" name="view0" width="160" jspType="html"
  encoding="UTF-8" toBeLayout="true" fileName="transfer.jsp"
  jspFile="transfer.jsp" y="116" x="121" height="70">
    <reference name="ref1" dest="flow0" condition="execute"/>
  </view>
  <flow id="accountManager" name="flow0" height="70" x="115" width="160"
  refId="transferAccount" fileName="accountManager.biz" y="215">
    <transition name="transition0" dest="view2" retValue="success"/>
    <transition name="transition1" dest="view1" retValue="failed"/>
  </flow>
  <view id="transfer.jsp" name="view1" x="220" height="70"
  jspFile="transfer.jsp" width="160" y="345"/>
  <view id="transferOK.jsp" name="view2" width="160" jspType="html"
  encoding="UTF-8" toBeLayout="true" fileName="transferOK.jsp"
  jspFile="transferOK.jsp" y="337" x="13" height="70"/>
</EMPMVC>
```

5.4.2.5. JSP 页面

transfer.jsp 内容如下:

```
<ctp:form name="transfer" isLayoutContent="true" action="transfer.do"
method="POST" >
<TABLE>
  <TR>
    <TD><ctp:label name="Label0" dataName="accountNo" text="客户帐户"/></TD>
    <TD><ctp:combobox iCollName="accountCollection" dataName="accountNo"
descName="accountNo" optionSrc="0" valueName="accountNo" /></TD>
  </TR>
  <TR>
    <TD><ctp:label name="Label1" dataName="accountPassword" text="帐户密码"
"/></TD>
    <TD><ctp:input name="accountPassword" dataName="accountPassword"
type="password"/></TD>
  </TR>
  <TR>
    <TD><ctp:label name="Label2" dataName="destAccount" text="目标帐号"
"/></TD>
    <TD><ctp:input name="destAccount" dataName="destAccount"
type="text"/><ctp:fieldError dataName="destAccount"/></TD>
  </TR>
  <TR>
    <TD><ctp:label name="Label3" dataName="destAccountName" text="目标帐号名"
"/></TD>
    <TD>
      <ctp:input name="destAccountName" dataName="destAccountName"
type="text" />
    </TD>
  </TR>
  <TR>
    <TD><ctp:label name="Label4" dataName="transferAmount" text="转帐金额"
"/></TD>
    <TD><ctp:input name="transferAmount" dataName="transferAmount"
dataType="Currency" type="text"/><ctp:fieldError
dataName="transferAmount"/></TD>
  </TR>
  <TR>
    <TD><ctp:label name="Label5" dataName="transferDescription" text="转帐说
明"/></TD>
    <TD><ctp:input name="transferDescription"
dataName="transferDescription"/></TD>
  </TR>
</TABLE>
<ctp:input type="submit" value="转帐"/>
```

上面红色字体的就是显示错误信息的域。

transferOK.jsp 则写个成功提示信息: 转帐成功。即可

5.4.2.6. 金额数据类型的校验

在实际应用中，往往金额的数据的输入都有限制，例如精度，最大值，最小值，非法字符的校验，在 EMP 中我们可以使用 `Currency DataType` 来对金额进行校验。金额数据类型 `Currency` 已经在 EMP 自带了，我们可以通过定制 `Currency` 的属性来满足不同的校验要求。

`Currency` 类型一共有 8 个属性，其中：`min`、`max` 是用于设置金额的最大值与最小值，`precision` 是金额的精度，即金额数据的最大长度，`scale` 是金额数据小数点后的位数。这两个属性合起来也可以设定金额数据的最大值与最小值，它们与 `min`、`max` 属性合起来使用时，取其中的最小范围。如最大值为 1000，而 `precision` 为 7，`scale` 为 2，而最大值也只能是 1000.00，若最大值为 100000，而 `precision` 为 5，`scale` 为 2，而最大值为 999.99。

`showDot` 属性定义的是在客户端显示的字符是否带有小数点，如 1234.00 数据，若是不带小数点，则显示为 123400；若带有小数点，则显示：1,234.00。

`keepDot` 属性定义的是后台的数据是否带有小数点，如：1234.00 数据，若是不带小数点，则后台存放的数据为：123400；若是带有小数点，则后台存放的数据为：1234.00。当然其中小数点的位数为 2。如果小数点的位数为 3，则若是不带小数点，则存放的数据为：1234000；若是带小数点，则存放的数据为：1234.000。如果小数点的位数为 1，则该数据不合法。

`keepStringValue` 属性定义的是后台的数据的格式是否是字符串，如果是，则以字符串的形式存放金额数据，如果不是，则是以 `BigDecimal` 格式存放金额数据。无论是否是字符串，存放的数据都不带有各种修饰符，即字符串也是由 `BigDecimal` 直接转换成的。

`showSymbol` 属性定义的是前端显示的字符串是否带有金额符号，即是否带有“¥”符号。

现在介绍一下数据校验的使用方法，数据校验分为前台校验和后台校验两种方式：前台校验只需要将页面的输入框的标签中加上 `Currency` 的数据类型校验，如给 `transfer.jsp` 页面的转帐金额进行前台校验。对页面做如下修改：

```
<ctp:input name="transferAmount" dataName="transferAmount" dataType="Currency"/>
```

后台校验是指数据提交到后台服务器上后对其进行校验，该校验过程是由 `Currency` 的 `implement Class` 的 `JAVA` 类来实现的。其具体的使用方法是在交易逻辑构件的交易流程下，为该交易的的数据类型添加类型校验。例如给转帐交易的转帐金额做后台交易，则在 `accountManager.biz` 的 `transfer` 流程做如下修改：

```
<input>
.....
<field id="transferAmount" dataType="Currency"/>
.....
</input>
```

5.4.3. 练习三：查询交易明细交易 tranDetail

5.4.3.1. 交易数据引用

在业务逻辑分组设定编辑器中引用交易明细相关数据：

```
tranDetailCollection
beginDate
endDate
```

5.4.3.2. 配置数据报文

ISO8583报文的域可以引用其它报文定义，EMP Format模块会自动对其进行格式化/反格式化，即格式化时将被引用报文格式化后作为这个域的内容，反格式化时将这个域的内容按照被引用报文格式进行反格式化（到数据域中）。

按照下表配置名为tranDetail和tranDetailXML的交易明细报文。其中tranDetail的30域引用tranDetailXML报文。

tranDetail:

数据域	域索引	IO类型	必需域	域类型	长度	取值
accountNo	13	发送/接收	必需	AN	19	
BranchNumber	20	发送		AN	5	10001
beginDate	16	发送		AN	8	
endDate	17	发送		AN	8	
Description	30	接收		LLVAR		
hostResult	37	接收		AN	1	

定义好tranDetail报文后，在accountManager.biz中增添如下内容：

```
<format id="tranDetail">
  <document>查询交易</document>
  <datas msgType="0720" codeSet="ASCII" packageType="ISO8583">
    <refData fieldType="AN" refId="accountNo" fieldState="M"
fieldIdx="13" fieldLength="19" IOType="IO"/>
    <refData fieldType="AN" refId="BranchNumber" fieldValue="10001"
fieldIdx="20" fieldLength="5" IOType="O"/>
    <refData fieldType="AN" refId="beginDate" fieldIdx="16"
fieldLength="8" IOType="O"/>
    <refData fieldType="AN" refId="endDate" fieldIdx="17"
fieldLength="8" IOType="O"/>
    <refData fieldType="LLVAR" refFormat="tranDetailXML"
refId="Description" fieldIdx="30" IOType="I"/>
    <refData fieldType="AN" refId="hostResult" fieldIdx="37"
fieldLength="1" IOType="I"/>
  </datas>
</format>
```



```

    </datas>
</format>

```

tranDetailXML:

采用XML方式对iColl进行打包，首先把tranDetailCollection引入报文编辑器，将其下的4个数据域如下配置：

数据域	标签名	数据域名来源	完整标签
tranDate	tranDate	dataName	true
tranType	tranType	dataName	true
tranAmount	tranAmount	dataName	true
tranDesc	tranDesc	dataName	true

然后还需选中这4个数据域，在右键菜单中选择“组包选定的数据”。由于EMP的iColl下的数据必须定义在kColl内，所以需要进行组包操作，数据分组相当于kColl一层，也可以为其定义xml报文标签。

定义好tranDetailXML后，accountManager.biz中增添了如下内容：

```

<format id="tranDetailXML">
    <document>查询交易XML</document>
    <datas version="1.0" packageType="WrapXMLFormat" encoding="gb2312">
        <refColl refId="tranDetailCollection" xmlTag="tranDetailCollection">
            <dataGroup>
                <refData dataNameType="0" refId="tranDate" fullTag="true"
xmlTag="tranDate"/>
                <refData dataNameType="0" refId="tranType" fullTag="true"
xmlTag="tranType"/>
                <refData dataNameType="0" refId="tranAmount" fullTag="true"
xmlTag="tranAmount"/>
                <refData dataNameType="0" refId="tranDesc" fullTag="true"
xmlTag="tranDesc"/>
            </dataGroup>
        </refColl>
    </datas>
</format>

```

而以上内容会被EMP编译，最终生成的报文定义如下：

```

<fmtDef id="tranDetailXML">
    <xmlWrap>
        <xmlHead version="1.0" encoding="gb2312"/>
        <xmlIColl dataName="tranDetailCollection"
            tagName="tranDetailCollection">
            <xmlWrap>
                <xmlFullTag dataName="tranDate" dataNameType="0"
                    tagName="tranDate" desc="交易日期"/>
                <xmlFullTag dataName="tranType" dataNameType="0"
                    tagName="tranType" desc="交易种类"/>
                <xmlFullTag dataName="tranAmount" dataNameType="0"
                    tagName="tranAmount" desc="交易金额"/>
                <xmlFullTag dataName="tranDesc" dataNameType="0"
                    tagName="tranDesc" desc="交易描述"/>
            </xmlWrap>
        </xmlIColl>
    </xmlWrap>
</fmtDef>

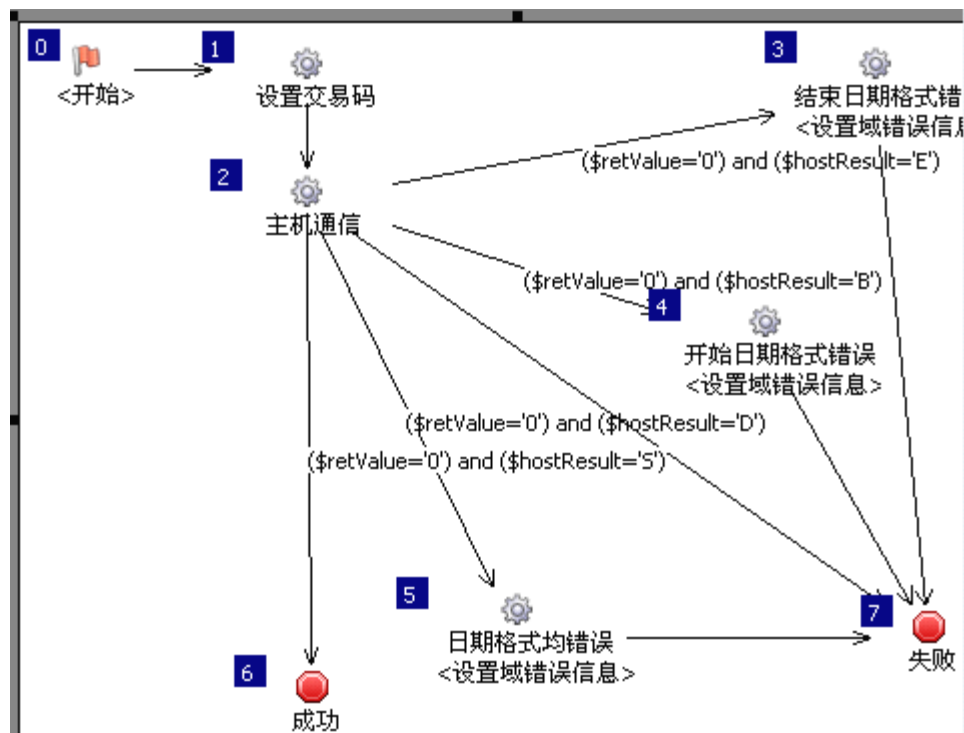
```

注：数据分组的标签名和数据域名可以不填写任何的内容，所以tranDetailCollection的

标签名依然要设置为tranDetailCollection。

5.4.3.3. 流程定义

本交易流程tranDetail的结构和转账交易基本相同，不再详述。输入数据为accountNo，beginDate，endDate。



注：流程图上的跳转条件的括号和大小写都要注意。

交易流程定义好后，accountManager.biz增添的内容如下所示：

```

<operation id="tranDetail" name="查询交易">
  <document>查询交易详细条目</document>
  <flow name="flow1" x="10" height="400" width="500" y="10">
    <SetValueAction name="SetValueAction0" x="97" height="31"
dataName="serviceId" label="设置交易码" width="94" dataValue="0720" y="10">
      <transition name="transition1" dest="EBankTCPIPAccessAction0"/>
    </SetValueAction>
    <EBankTCPIPAccessAction name="EBankTCPIPAccessAction0" width="86"
sendHeadFormatName="headReqFmt" sendFormatName="tranDetail"
serviceName="tcpipService" y="74" label="主机通信" x="101"
receiveFormatName="tranDetail" height="31" receiveHeadFormatName="headRepFmt">
      <transition name="transition2" dest="SetFieldErrorMsgAction0"
condition="($retValue='0') and ($hostResult='E')"/>
      <transition name="transition3" dest="SetFieldErrorMsgAction1"
condition="($retValue='0') and ($hostResult='B')"/>
      <transition name="transition4" dest="SetFieldErrorMsgAction2" co
  
```

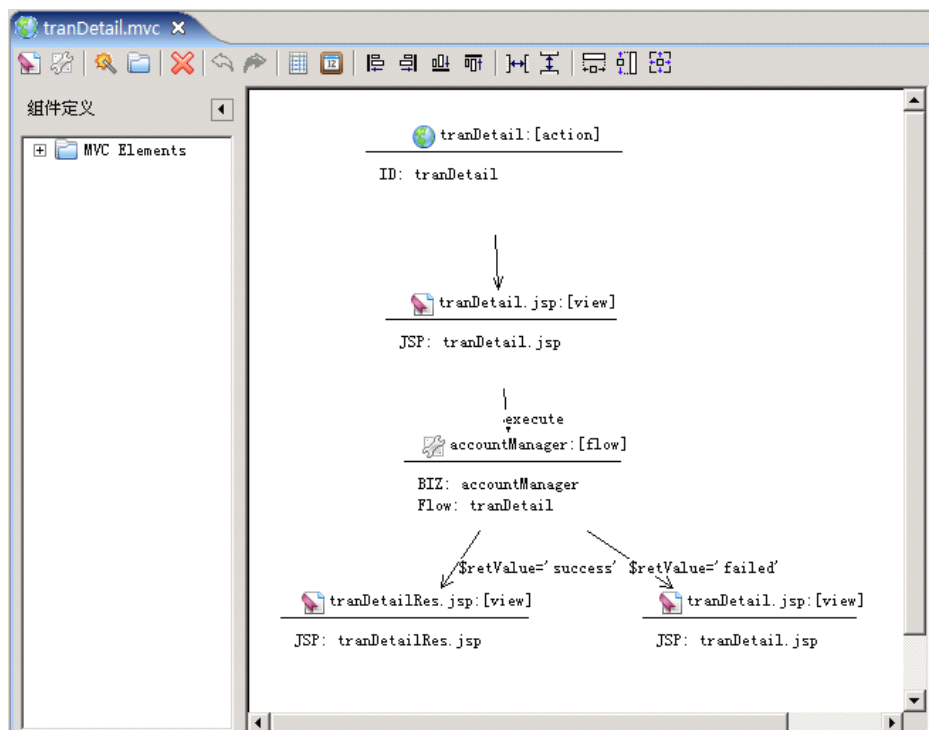
```

ndition="($retValue='0') and ($hostResult='D')"/>
    <transition name="transition5" dest="EndAction1"/>
    <transition name="transition9" dest="EndAction0"
condition="($retValue='0') and ($hostResult='S')"/>
    </EBankTCPIPAccessAction>
    <SetFieldErrorMsgAction name="SetFieldErrorMsgAction0" width="100"
needClear="true" errorMsgs="出错" y="10" label="结束日期格式错误" x="378"
height="52" fieldNames="endDate">
    <transition name="transition6" dest="EndAction1"/>
    </SetFieldErrorMsgAction>
    <SetFieldErrorMsgAction name="SetFieldErrorMsgAction1" width="106"
needClear="true" errorMsgs="出错" y="139" label="开始日期格式错误" x="320"
height="46" fieldNames="beginDate">
    <transition name="transition7" dest="EndAction1"/>
    </SetFieldErrorMsgAction>
    <SetFieldErrorMsgAction name="SetFieldErrorMsgAction2" width="110"
needClear="true" errorMsgs="出错|出错" y="283" label="日期格式均错误" x="194"
height="50" fieldNames="endDate|beginDate">
    <transition name="transition8" dest="EndAction1"/>
    </SetFieldErrorMsgAction>
    <StartAction name="StartAction0" x="10" height="31" width="48"
y="9">
    <transition name="transition0" dest="SetValueAction0"/>
    </StartAction>
    <EndAction name="EndAction0" x="113" result="success" height="31"
label="成功" width="68" y="322"/>
    <EndAction name="EndAction1" x="426" result="failed" height="31"
label="失败" width="58" y="292"/>
</flow>
<input>
    <refData refId="accountNo"/>
    <refData refId="beginDate"/>
    <refData refId="endDate"/>
</input>
<output/>
</operation>

```

5.4.3.4. mvc 定义

tranDetail.mvc如下图，请求页面和失败响应页面为tranDetail.jsp，成功响应页面为tranDetailRes.jsp，引用tranDetail流程，使用normal控制器：



tranDetail.mvc定义好后，其文件的具体内容如下所示：

```

<EMPMVC name="tranDetail" x="0" height="464" width="525" y="0">
  <action id="tranDetail" name="action0" x="110" height="80" width="180" type="normal"
  checkSession="false" y="10">
    <reference name="ref0" dest="view0"/>
  </action>
  <view id="tranDetail.jsp" name="view0" width="160" jspType="html" encoding="UTF-8"
  toBeLayout="true" fileName="tranDetail.jsp" jspFile="tranDetail.jsp" y="108" x="122"
  height="70">
    <reference name="ref1" dest="flow0" condition="execute"/>
  </view>
  <flow id="accountManager" name="flow0" height="70" x="123" width="160"
  refId="tranDetail" fileName="accountManager.biz" y="206">
    <transition name="transition0" dest="view1" retValue="success"/>
    <transition name="transition1" dest="view2" retValue="failed"/>
  </flow>
  <view id="tranDetailRes.jsp" name="view1" width="160" jspType="html" encoding="UTF-8"
  toBeLayout="true" fileName="tranDetailRes.jsp" jspFile="tranDetailRes.jsp" y="328" x="13"
  height="70">
  </view>
  <view id="tranDetail.jsp" name="view2" x="210" height="70" jspFile="tranDetail.jsp"
  width="160" y="325"/>
</EMPMVC>

```

5.4.3.5. JSP 页面

tranDetail.jsp内容如下：

```

<ctp:form name="tranDetail" isLayoutContent="true" action="tranDetail.do"
method="POST">
  <TABLE>
    <TR>
      <TD><ctp:label name="Label0" dataName="accountNo" text="客户帐户"
"/></TD>

```

```

        <TD><ctp:combobox iCollName="accountCollection" dataName="accountNo"
descName="accountNo" optionSrc="0" valueName="accountNo"/></TD>
    </TR>
    <TR>
        <TD><ctp:label name="Label1" dataName="beginDate" text="起始日期
"/></TD>
        <TD><ctp:input name="beginDate"
dataName="beginDate"/><ctp:fieldError dataName="beginDate"/></TD>
    </TR>
    <TR>
        <TD><ctp:label name="Label2" dataName="endDate" text="结束日期
"/></TD>
        <TD><ctp:input name="endDate" dataName="endDate"/><ctp:fieldError
dataName="endDate"/></TD>
    </TR>
</TABLE>
    <ctp:input type="submit" value="查找"/>
</ctp:form>

```

注：客户帐户的下拉列表框中，iCollName的属性值为：accountCollection，且valueName的属性值为：accountNo。

tranDetailRes.jsp内容如下：

```

...
<ctp:label text="以下是帐户["/>
<ctp:text name="accountNo" dataName="accountNo"/>
<ctp:label text="]"在["/>
<ctp:text name="accountNo" dataName="accountNo"/>
<ctp:label text="]"和["/>
<ctp:text name="accountNo" dataName="accountNo"/>
<ctp:label text="]"期间的交易明细："/>
<ctp:table needTableTitle="true" iCollName="tranDetailCollection" border="1"
isLayoutContent="true">
    <ctp:tableColumn dataName="tranDate" label="交易日期" formFieldType="none"/>
    <ctp:tableColumn dataName="tranType" label="类型" formFieldType="none"/>
    <ctp:tableColumn dataName="tranAmount" label="金额" formFieldType="none"/>
    <ctp:tableColumn dataName="tranDesc" label="描述" formFieldType="none"/>
</ctp:table>
...

```

5.4.3.6. 日期数据类型的校验

EMP 提供了各种数据类型的校验，前面我们介绍过了金额数据类型的校验，这里我们在介绍一下日期数据类型校验的使用方法。

我们可以在项目设定编辑器中的数据类型中通过改变 Date 的属性来定制一个符合实际要求的日期数据类型来进行校验。

Date 类型一共有 6 个属性：其中 keepStringValue 属性与其它类型一样，如果 keepStringValue 为 false，则后台存放的数据格式就为 Date 格式。

inputFormat 属性定义的是客户端日期显示的格式，如 2006 年 3 月 1 号，若 inputFormat 为 “yyyy ”，则在客户端显示的字符串为：2006/03/01。其中表示日期的字符必须是 y、M、d、H、h、K、k、a、m、s（具体代表的含义与 JAVA 程序中表示的含义一样）。如果是其它字符则看成是修饰符号，如“年”、“月”、“日”等。

valueFormat 属性定义的是后台的数据格式，这与 inputFormat 设置方法一样。inputAmPm 属性定义的前提是 inputFormat 属性中定义中有字符 a。该属性的作用是显示 am/pm 的字符，例如设定为“上午|下午”，如果显示格式中有要求显示 am/pm（即显示格式中有字符 a），则显示的字符串会在相应的位置显示上午或者是下午。如果设定的是“am|pm”，则显示的则是 am 或者是 pm。如果在显示格式 inputFormat 中没有定义字符 a，则该属性不起作用。valueAmPm 与 inputAmPm 属性类似，只是作用的是后台的数据。

Type 属性定义的是后台存放的数据是：单纯的时间格式、单纯的日期格式或者是同时包括日期与时间格式。该属性受到 valueFormat 和 keepStringValue 属性的影响。首先后台的数据必须是存放为字符串格式，Type 属性才会起作用。另外，Type 属性必须符合 inputFormat 属性，即 inputFormat 属性如果没有日期，则 Type 属性就不能选择日期或日期与时间，否则则抛出异常，因此，Type 属性必须与 inputFormat 合在一起使用。另外，Type 属性与 valueFormat 属性也有一定的关系，如果 valueFormat 定义为空，且存储的数据为字符串，则存储的格式就由 Type 属性决定。

日期数据的校验类型也分为前台和后台。其方法和前面讲到的金额数据类型基本一致。例如现在在项目设定编辑器中定义一个叫 Date 的数据类型，其在 designFiles/commons/dataTypeDef.xml 中生成的内容如下所示：

```
<Date id="Date" name=" 日期 " inputFormat="yyyy 年 MM 月 dd 日 " type="DateType"
keepStringValue="true" valueFormat="yyyyMMdd"/>
```

如果需要使用页面校验则在页面的输入框里加上 dataType="Date"。例如需要给 tranDetail.jsp 页面的开始日期和结束日期加上日期校验，需做如下修改：

```
.....
<TD><ctp:inputname="beginDate" dataName="beginDate" dataType="Date"/></TD>
.....
<ctp:input name="endDate" dataName="endDate" dataType="Date"/></TD>
.....
```

如果需要对对其进行后台的数据校验，可以在 accountManager.biz 的 tranDetail 流程做如下修改：

```
<input>
.....
<field id=" beginDate " dataType="Date"/>
<field id=" endDate " dataType="Date"/>
.....
</input>
```

当然也可以在数据字典中为其添加 dataType，这样整个项目中该数据的输入都会已定好的方式对其进行数据校验。

5.5. 课程 9：其他 MVC 控制器

课程名称	MVC 控制器	课程时间	
------	---------	------	--

课程目标	一、学习并掌握图表产生的 MVC 控制器配置方法 二、学习并掌握多步骤操作的 MVC 控制器配置方法
准备工作	完成课程 4

5.5.1. 制作信用卡申请交易

5.5.1.1. 交易数据引用

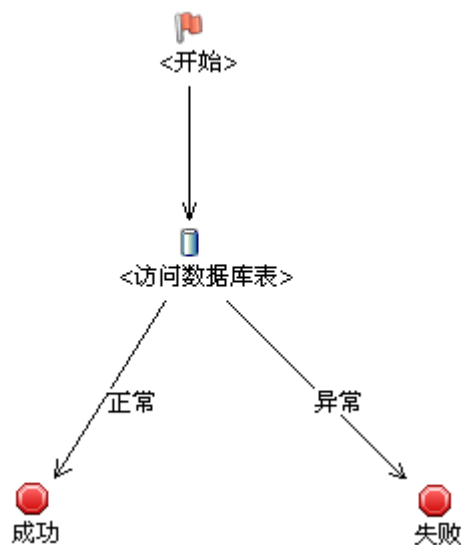
在customerManagerSession.biz中引用信用卡相关数据(信用卡分组下的所有数据)

5.5.1.2. 服务定义

定义名为cardTable的访问数据库表服务。注意此处CHINESENAME没有找到映射，需要手工添加映射到name数据域。

5.5.1.3. 流程定义

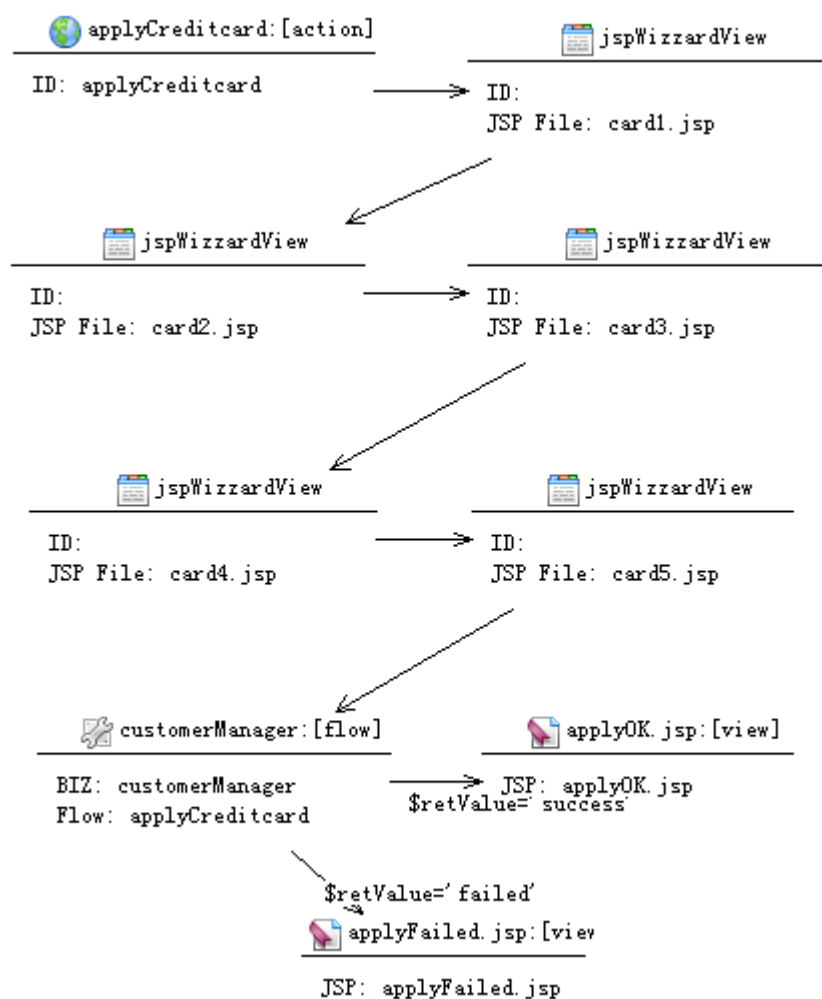
交易流程applyCreditcard的输入数据：信用卡分组下的所有数据；id, userid, name, phoneNo, email, address, postcode, certType, certNo, mobile（共49个）



5.5.1.4. mvc 定义: Wizard 控制器

由于申请信用卡需要填写的数据非常多，都放在一个页面中显然不合适，因此采用向导式的页面提交方式。EMP提供的Wizard Controller可以实现这一需求。此controller可以把每一步提交的数据放在一个临时的context中，最后一并提交给flow进行处理。

下图是applyCreditcard.mvc的模型:



其中 `applyCreditcard` 的 action 控制器选为 Wizard 类型，由 action 指向一条由若干 `jspWizardView` 串成的页面链，最后转向 flow。这样在运行时，这些页面就会以向导的方式逐步出现，并且还能够实现返回“上一步”操作。

5.5.1.5. 向导页面定义

若想让一个页面变成向导页面，需要在表单中加入两个隐藏的数据域，分别为command和pageIndex。其中command指定表单提交后的操作为next（下一步）、last（上一步）还是submit（提交）。而pageIndex指定该页面在向导中处于第几页。

从第二页开始页面中需要出现“上一步”的按钮（链接），但表单中不能出现重复的数据域，因此需要使用javascript等方法进行提交。这里采用一个折衷的方法，使用一个ctp:URL直接将command和pageIndex参数提交到applyCreditcard.do。如下所示。

```
.....
<emp:form>
.....
    <input type="hidden" name="command" value="next" />
    <input type="hidden" name="pageIndex" value="0" />
    <emp:URL href="applyCreditcard.do?pageIndex=1&command=last" contentDivId="div0"
    label="上一步"/>
    <emp:input value="下一步" type="submit" />
</emp:form>
```

href为applyCreditcard.do?pageIndex=1&command=last。

用此方法制作好所有页面。最后一页的command为submit。

部署、启动服务器，查看运行结果。

注：第一个页面的pageIndex的value值为0，后面的页面依次递增，command的value值默认为next，如果pageIndex的值未从0开始，则页面控制会出错。

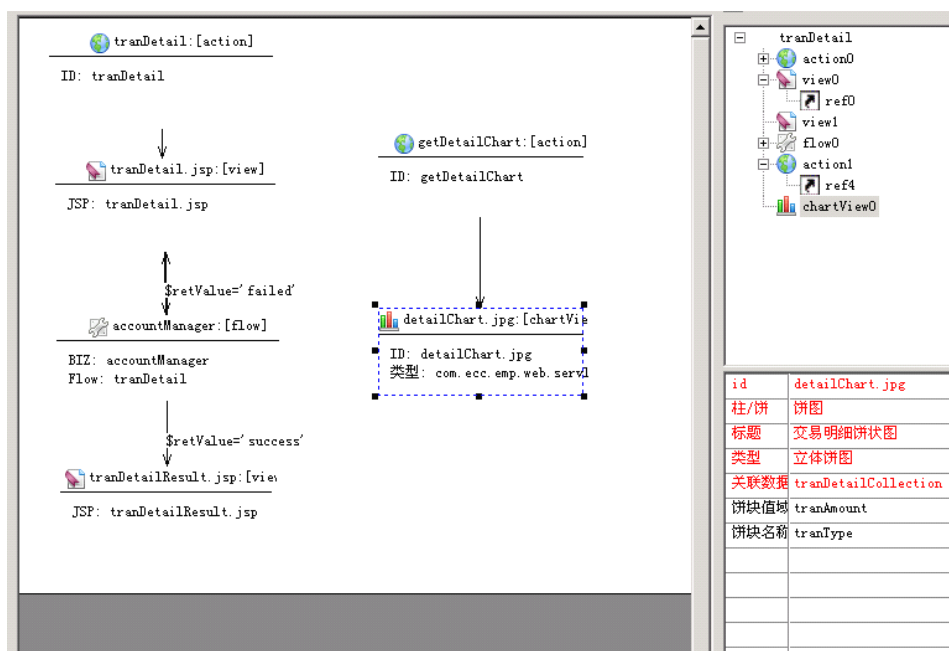
5.5.2. 显示图表

5.5.2.1. chartView

EMP提供了借助JFreeChart实现的图表绘制功能，能够把数据集合中的数据显示为柱状或饼状图。只需要简单的配置就可以使用这个功能。我们来为查询交易明细添加一张收入支出比的饼状图。

在tranDetail.mvc中添加一个名为getDetailChart的action，指向一个chartView。

chartView的设置如下图：关联数据设为tranDetailCollection，饼块值域设为tranAmount，饼块名称（分类的依据）设为tranType。



这样在执行 getDetailChart.do 后就会得到一个饼状图的 jpg 图片，只需要在 tranDetailResult.jsp 页面显示图片的地方添加一个 ctp:img，设置其 src 为 getDetailChart.do 即可。

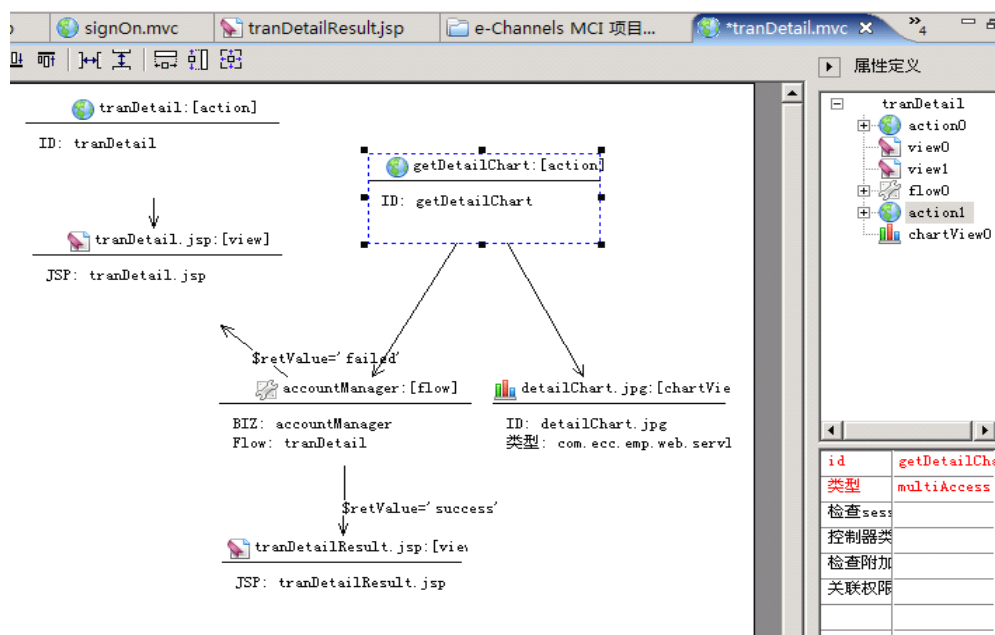
```
<ctp:label text="以下是帐户["/>
<ctp:text name="accountNo" dataName="accountNo"/>
<ctp:label text="]在["/>
<ctp:text name="accountNo" dataName="accountNo"/>
<ctp:label text="]和["/>
<ctp:text name="accountNo" dataName="accountNo"/>
<ctp:label text="]期间的交易明细："/>
<ctp:table needTableTitle="true" iCollName="tranDetailCollection" border="1"
isLayoutContent="true">
  <ctp:tableColumn dataName="tranDate" label="交易日期" formFieldType="none"/>
  <ctp:tableColumn dataName="tranType" label="类型" formFieldType="none"/>
  <ctp:tableColumn dataName="tranAmount" label="金额" formFieldType="none"/>
  <ctp:tableColumn dataName="tranDesc" label="描述" formFieldType="none"/>
</ctp:table>
<emp:img name="detailChart" src="getDetailChart.do?viewId=detailChart.jpg" />
```

部署、启动服务器，查看结果，会发现图片并不能显示，而且控制台中报告了找不到数据的异常。原因在于（可以先自己思考一下为什么？），在得到 tranDetailResult.jsp 响应页面时，流程已经执行结束，其交易私有 Context 也已经销毁，此时再去向 getDetailChart.do 发送请求，就会报告找不到数据的错误。如果图表要显示的数据是 session 数据，则这么做是没有问题的。

5.5.2.2. MultiAccess 控制器

若想使用私有数据产生图表，就需要用到MultiAccess控制器。MultiAccess控制器的作用是，直到访问其它交易之前，都会暂时保存着交易数据，以便进行多次请求。这和Wizard控制器有一些相像。

修改tranDetail.mvc，将tranDetail.jsp到flow的箭头删去，改由getDetailChart.do来调用；并将getDetailChart.do的控制器类型改为MultiAccess。如下图。



然后修改tranDetail.jsp，把表单的请求改为getDetailChart.do。

最后还要修改tranDetailRes.jsp中ctp:img的src属性为：

getDetailChart.do?viewId=detailChart.jpg (加了viewId参数)

这样当tranDetail.jsp的表单提交请求到getDetailChart.do时（第一次请求），会执行flow得到tranDetailCollection的数据，然后转到响应页面tranDetailRes.jsp。在页面中的img再次调用getDetailChart.do（第二次请求），请求得到id为detailChart.jpg的view。由于用了MultiAccess控制器，仍然保存着之前的数据，所以这次可以正常得到图表图片。

5.6. 课程 9：应用监控

EMP提供了基于JMX框架下的J2EE应用监控平台。EMP应用针对这个监控平台提供了内嵌的支持。所有的EMP应用均可以无编码的纳入到EMP Monitor的监控应用中。

EMP监控平台采用JMX MBean的方式对应用对象进行监控和管理。EMP平台所提供的所有的应用组件均可以在开发完成后，很方便地通过配置文件的定义将应用组件导出为一个JMX MBean对象，该应用组件可根据自身管理的需要开放自己的属性和方法。监控平台可以通过JMX机制拿到被监控组件所开放的属性和方法，从而完成对应用组件的信息察看和方法调用控制。

5.6.1. 将 EMP 应用加入到监控平台

启动EMP Monitor项目，通过http://127.0.0.1:8080/EMP_Monitor/signOn.do在浏览器中访问。

点击菜单树的应用节点管理，可以看到应用节点列表。如下图：

应用节点图：

点击上图中的【添加应用节点】按钮，进入添加应用节点信息页面，输入应用节点基本信息。如下图：新增应用节点中文命名为：“示例应用”。

应用节点新增图：

点击【保存】按钮完成应用节点的新增，返回到应用节点列表页面。

注意：添加应用节点信息时，端口号必须是数字类型，长度不能超过5位；应用节点名称不能和已有的应用节点名称重复。

基本被监控应用信息如下：

IPAddress: 127.0.0.1; Port: 10330; 服务名: jmxServer

添加完成后，可以看到在左边的树节点上已经有了新的应用节点：示例应用。

5.6.2. 启动对应用的监控

点击菜单树的应用节点，在应用管理页面的应用节点状态栏可以看到，当前应用节点是否启动。

点击【启动】按钮，启动应用节点监控，菜单树刷新显示应用节点信息，此时应用节点状态改变为“已启动”。如下图：

点击【停止】按钮，停止应用节点监控，菜单树不显示应用节点信息，应用节点状态改变为“已停止”。如下图：

应用节点新增图：

在该显示页面上可以对被监控应用的所有已开放的MBean对象进行属性察看和方法调用。通过MBean的操作可以很清晰的了解到被监控应用的当前状态和应用运行情况。还可以通过监控平台进行应用的信息采集和报警设置。

6. 附录

6.1. 公共定义

6.1.1. session 数据

id	客户号	dataElement
userid	客户注册 ID	dataElement
name	客户名称	dataElement
email	电子邮件	dataElement
accountCollection	关联帐户信息	dataCollection
accountNo		
accountName		
openDate		
accountRate		

accountType		
accountBalance		
CurrencyCode		
freezeBalance		

6.1.2. 公共服务

transactionManager	DataSourceTransactionManager
dataSource	JDBCDataSource
accountinfoTable	JDBCTableService

6.2.customerManager.biz

6.2.1. biz 配置

6.2.1.1. 数据定义

errorCode	错误代码	dataElement
errorMsg	错误信息	dataElement
hostResult	主机返回结果	dataElement
password	登录密码	dataElement
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一账户	dataElement
secondAccount	第二账户	dataElement
CurrencyCode	币种	dataElement
tranAmount	交易金额	dataElement
comment	备注	dataElement
ejCollection	日志信息集合	dataCollection
cardType	卡种	dataElement
autoChange	自动降档申请	dataElement
creditLine	额度	dataElement
sex	性别	dataElement
englishName	英文姓名（拼音）	dataElement
birthday	出生日期	dataElement
nation	国籍	dataElement
marriage	婚姻状况	dataElement

education	教育程度	dataElement
resAddress	户籍地址	dataElement
resZipcode	户籍邮编	dataElement
houseType	房产类别	dataElement
houseLoanPay	月均还款	dataElement
houseLoanEndDate	还贷结束年	dataElement
houseRentFee	月租	dataElement
carType	汽车品牌	dataElement
carBuyDate	购买日期	dataElement
carId	车牌号码	dataElement
carLoanPay	车贷还款	dataElement
otherCardNo	其他银行信用卡号	dataElement
otherCardCreditLine	其他卡信用额度	dataElement
companyName	单位全称	dataElement
industry	行业类别	dataElement
department	任职部门	dataElement
duty	职务/岗位	dataElement
serviceLen	工龄	dataElement
income	年收入	dataElement
directContactName	联系人姓名	dataElement
directContactRel	联系人关系	dataElement
directTel	联系人电话	dataElement
directMobile	联系人手机	dataElement
otherContactName	第二联系人姓名	dataElement
otherContactRel	第二联系人关系	dataElement
otherTel	第二联系人电话	dataElement
otherMobile	第二联系人手机	dataElement
addressType	帐单地址类别	dataElement
debitcardNo	人民币卡号	dataElement
debitcardUSD	美元卡号	dataElement
payType	还款方式	dataElement

6.2.1.2. 服务定义

userinfoTable	JDBCTableService
cardTable	JDBCTableService

6.2.2. 注册

6.2.2.1. 所需数据

id	客户号	dataElement
userid	客户注册 ID	dataElement
name	客户名称	dataElement
password	登录密码	dataElement
email	电子邮件	dataElement

6.2.2.2. 所需服务

userinfoTable JDBCTableService private

6.2.2.3. 输入

userid
name
password
email

6.2.3. 签到

6.2.3.1. 所需数据

id	客户号	dataElement
userid	客户注册 ID	dataElement
name	客户名称	dataElement
password	登录密码	dataElement
email	电子邮件	dataElement
accountCollection	关联帐户信息	dataCollection
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一账户	dataElement
secondAccount	第二账户	dataElement
tranAmount	交易金额	dataElement
CurrencyCode	币种	dataElement
comment	备注	dataElement

6.2.3.2. 所需服务

userinfoTable

accountinfoTable

JDBCTableService

JDBCTableService

private

common

6.2.3.3. 输入

userid

password

6.2.4. 签退

6.2.4.1. 所需数据

id	客户号	dataElement
userid	客户注册 ID	dataElement
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一账户	dataElement
secondAccount	第二账户	dataElement
tranAmount	交易金额	dataElement
CurrencyCode	币种	dataElement
comment	备注	dataElement

6.2.4.2. 所需服务

无

6.2.4.3. 输入

无

6.2.5. 查询日志

6.2.5.1. 所需数据

无

6.2.5.2. 所需服务

无

6.2.5.3. 输入

无

6.2.6. 申请信用卡

6.2.6.1. 所需数据

id	客户号	dataElement
userid	客户注册 ID	dataElement
name	客户名称	dataElement
phoneNo	电话号码	dataElement
email	电子邮件	dataElement
address	地址	dataElement
postcode	邮政编码	dataElement
certType	证件类型	dataElement
certNo	证件号码	dataElement
mobile	手机号码	dataElement
cardType	卡种	dataElement
autoChange	自动降档申请	dataElement
creditLine	额度	dataElement
sex	性别	dataElement
englishName	英文姓名（拼音）	dataElement
birthday	出生日期	dataElement
nation	国籍	dataElement
marriage	婚姻状况	dataElement

education	教育程度	dataElement
resAddress	户籍地址	dataElement
resZipcode	户籍邮编	dataElement
houseType	房产类别	dataElement
houseLoanPay	月均还款	dataElement
houseLoanEndDate	还贷结束年	dataElement
houseRentFee	月租	dataElement
carType	汽车品牌	dataElement
carBuyDate	购买日期	dataElement
carId	车牌号码	dataElement
carLoanPay	车贷还款	dataElement
otherCardNo	其他银行信用卡号	dataElement
otherCardCreditLine	其他卡信用额度	dataElement
companyName	单位全称	dataElement
industry	行业类别	dataElement
department	任职部门	dataElement
duty	职务/岗位	dataElement
serviceLen	工龄	dataElement
income	年收入	dataElement
directContactName	联系人姓名	dataElement
directContactRel	联系人关系	dataElement
directTel	联系人电话	dataElement
directMobile	联系人手机	dataElement
otherContactName	第二联系人姓名	dataElement
otherContactRel	第二联系人关系	dataElement
otherTel	第二联系人电话	dataElement
otherMobile	第二联系人手机	dataElement
addressType	帐单地址类别	dataElement
debitcardNo	人民币卡号	dataElement
debitcardUSD	美元卡号	dataElement
payType	还款方式	dataElement

6.2.6.2. 所需服务

cardTable

JDBCTableService

private

6.2.6.3. 输入

id
userid
name
phoneNo

email
address
postcode
certType
certNo
mobile
cardType
autoChange
creditLine
sex
englishName
birthday
nation
marriage
education
resAddress
resZipcode
houseType
houseLoanPay
houseLoanEndDate
houseRentFee
carType
carBuyDate
carId
carLoanPay
otherCardNo
otherCardCreditLine
companyName
industry
department
duty
serviceLen
income
directContactName
directContactRel
directTel
directMobile
otherContactName
otherContactRel
otherTel
otherMobile
addressType
debitcardNo
debitcardUSD
payType

6.3.accountManager.biz

6.3.1. biz 配置

6.3.1.1. 数据定义

errorCode	错误代码	dataElement
errorMsg	错误信息	dataElement
hostResult	主机返回结果	dataElement
accountNo	客户帐户	dataElement
BranchNumber	网点号	dataElement
accountPassword	帐户密码	dataElement
accountName	帐户名称	dataElement
openDate	开户日期	dataElement
accountRate	帐户利率	dataElement
accountType	帐户类型	dataElement
accountBalance	帐户余额	dataElement
CurrencyCode	币种	dataElement
freezeBalance	冻结余额	dataElement
Description	描述	dataElement
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一账户	dataElement
secondAccount	第二账户	dataElement
tranAmount	交易金额	dataElement
comment	备注	dataElement
destAccount	帐户利率	dataElement
destAccountName	帐户类型	dataElement
transferAmount	帐户余额	dataElement
transferDescription	币种	dataElement
beginDate	起始日期	dataElement
endDate	结束日期	dataElement
tranDetailCollection	交易明细集合	dataCollection

6.3.1.2. 服务定义

tcpipService

TCPIPService

6.3.1.3. 报文定义

queryAccount: ISO8583 交易码0718

数据域	域索引	I0类型	必需域	域类型	长度	取值
accountNo	13	发送/接收	必需	AN	19	
BranchNumber	20	发送		AN	5	10001
accountPassword	15	发送		AN	6	
accountName	14	接收		AN	20	
openDate	21	接收		AN	8	
accountRate	22	接收		AN	13	
accountType	23	接收		AN	1	
accountBalance	24	接收		AN	20	
CurrencyCode	25	接收		AN	3	

transfer: ISO8583 交易码0719

数据域	域索引	I0类型	必需域	域类型	长度	取值
accountNo	13	发送/接收	必需	AN	19	
accountPassword	15	发送		AN	6	
BranchNumber	20	发送		AN	5	10001
destAccount	30	发送		AN	19	
destAccountName	32	发送		AN	20	
transferAmount	35	发送		AN	20	
transferDescription	36	发送		LLVAR		
hostResult	37	接收		AN	1	

tranDetail: ISO8583 交易码 0720 30域引用tranDetailXML

数据域	域索引	I0类型	必需域	域类型	长度	取值
accountNo	13	发送/接收	必需	AN	19	
BranchNumber	20	发送		AN	5	10001
beginDate	16	发送		AN	8	
endDate	17	发送		AN	8	
Description	30	接收		LLVAR		
hostResult	37	接收		AN	1	

tranDetailXML: WrapXML

数据域	标签名	数据域名来源	完整标签
tranDetailCollection	tranDetailCollection		
tranDate	tranDate	dataName	true
tranType	tranType	dataName	true
tranAmount	tranAmount	dataName	true
tranDesc	tranDesc	dataName	true

6.3.2. 添加关联帐户

本交易将用户提交的帐户号和密码发送虚拟主机，得到帐户资料后将其插入关联帐户表和 session 中的 accountCollection，并返回 success。若主机操作失败则进入错误分支；若插入关联帐户表失败则设置 accountNo 的域错误信息为“账户已关联”后进入错误分支。在错误分支中设置错误信息为“主机交易失败”，返回 failed。成功与失败都需要记录日志。

6.3.2.1. 所需数据

accountNo	客户帐户	dataElement
BranchNumber	网点号	dataElement
accountPassword	帐户密码	dataElement
accountName	帐户名称	dataElement
openDate	开户日期	dataElement
accountRate	帐户利率	dataElement
accountType	帐户类型	dataElement
accountBalance	帐户余额	dataElement
CurrencyCode	币种	dataElement
freezeBalance	冻结余额	dataElement
Description	描述	dataElement
id	客户号	dataElement
userid	客户注册 ID	dataElement
accountCollection	关联帐户信息	dataCollection
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一账户	dataElement
secondAccount	第二账户	dataElement
tranAmount	交易金额	dataElement
comment	备注	dataElement

6.3.2.2. 所需服务

tcpipService	TCPIPSERVICE	private
accountinfoTable	JDBCTableService	common

6.3.2.3. 所需报文

queryAccount

6.3.2.4. 输入

accountNo
accountPassword

6.3.3. 查询帐户信息

本交易将用户提交的帐户号发送虚拟主机，得到帐户资料并返回 success。若主机操作失败则设置错误信息为“主机交易失败”，返回 failed。成功与失败都需要记录日志。

6.3.3.1. 所需数据

accountNo	客户帐户	dataElement
BranchNumber	网点号	dataElement
accountPassword	帐户密码	dataElement
accountName	帐户名称	dataElement
openDate	开户日期	dataElement
accountRate	帐户利率	dataElement
accountType	帐户类型	dataElement
accountBalance	帐户余额	dataElement
CurrencyCode	币种	dataElement
freezeBalance	冻结余额	dataElement
Description	描述	dataElement
id	客户号	dataElement
userid	客户注册 ID	dataElement
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一帐户	dataElement
secondAccount	第二帐户	dataElement
tranAmount	交易金额	dataElement
comment	备注	dataElement

6.3.3.2. 所需服务

tcpipService

TCPIPService

private

6.3.3.3. 所需报文

queryAccount

6.3.3.4. 输入

accountNo

6.3.4. 转帐

本交易将用户提交的转账信息发送虚拟主机，得到响应结果。若结果标志为 S 则返回 success。若主机操作失败或结果标志不为 S 则设置相应错误信息并返回 failed。成功与失败都需要记录日志。

6.3.4.1. 所需数据

accountNo	客户帐户	dataElement
BranchNumber	网点号	dataElement
accountPassword	帐户密码	dataElement
destAccount	帐户利率	dataElement
destAccountName	帐户类型	dataElement
transferAmount	帐户余额	dataElement
transferDescription	币种	dataElement
hostResult	主机返回结果	dataElement
id	客户号	dataElement
userid	客户注册 ID	dataElement
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一帐户	dataElement
secondAccount	第二帐户	dataElement
CurrencyCode	币种	dataElement
tranAmount	交易金额	dataElement
comment	备注	dataElement

6.3.4.2. 所需服务

tcpipService

TCPIPService

private

6.3.4.3. 所需报文

transfer

6.3.4.4. 输入

accountNo
accountPassword
destAccount
destAccountName
transferAmount
transferDescription

6.3.5. 查询交易明细

本交易将用户提交的帐户号、起始结束时间发送虚拟主机，得到交易明细结果。若结果标志为 S 则返回 success。若主机操作失败或结果标志不为 S 则设置相应错误信息并返回 failed。成功与失败都需要记录日志。

6.3.5.1. 所需数据

accountNo	客户帐户	dataElement
beginDate	起始日期	dataElement
endDate	结束日期	dataElement
tranDetailCollection	交易明细集合	dataCollection
hostResult	主机返回结果	dataElement
id	客户号	dataElement
userid	客户注册 ID	dataElement
tranId	交易代码	dataElement
tranStatus	交易状态	dataElement
firstAccount	第一账户	dataElement
secondAccount	第二账户	dataElement
CurrencyCode	币种	dataElement
tranAmount	交易金额	dataElement
comment	备注	dataElement

6.3.5.2. 所需服务

tcipService

TCIPService

private

6.3.5.3. 所需报文

tranDetail
tranDetailXML

6.3.5.4. 输入

accountNo
beginDate
endDate