HealthConnect开发 -
Business Process Language

InterSystems
Creative data technology

1

1

提纲

- 复习BP的基本概念
- BPL的基本activity- 调用(call)
- 业务协调和流程管理
  - —if/else, foreach, swtich, sycn
  - —其他的activities
- BPL的错误处理

2

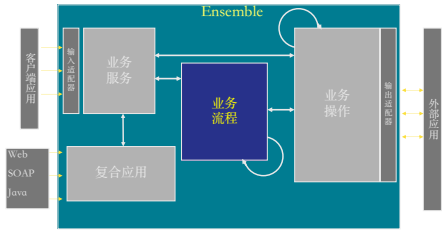
Business Process基本概念

- BP只发送请求消息给BO或者另一个BP, 请求消息和收到的响应消息都在Ensemble内部，不连接外部系统.
  - —请求可以是同步或者异步的，
  - —BP可以要等一个响应消息等很久，等待过程中，BP会被写到磁盘，收到响应才重新调出。

  - —消息校验，转发，分发，路由。。。
  - —数据过滤，校验，转换，汇总
  - —业务协调，术语管理
  - —订阅发布，工作流，
  - —告警管理
  - —。。。。

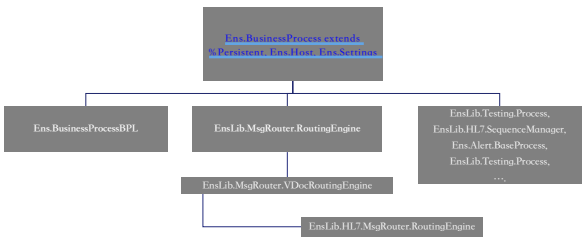
3

BP模型



4

消息路由引擎是已经开发好的BP



5

开发BP的两种方式

BPL.

- —继承Ens.BusinessProcessBPL.
- —图形编辑工具
- —编译后生成XDATA

代码实现(客户定制BP)

- 继承Ens.BusinessProcess.
- 必须实现OnRequest()
- 可以override OnResponse(), OnComplete()

Class Demo.Loan.FindRateDecisionProcessCustom Extends Ens.BusinessProcess

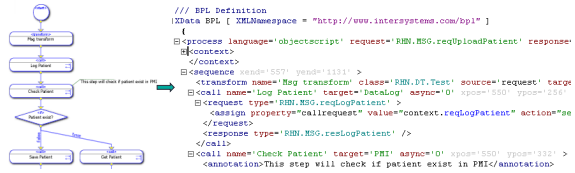
6

BPL基础以及Call

7

业务流程建模

- BPL编译后生成继承Ens.BusinessProcessBPL
- 业务逻辑生成XDATA



8

BPL对象

—请求(request)
—响应(response)
—上下文(context)
—callrequest
—callresponse.

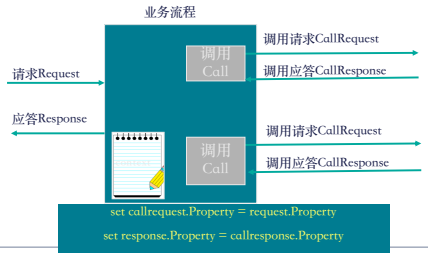
Process.

- 访问BP的对象，或者说， XDATA中的根节点

```
<process language='objectscript' request='HIP.MSG.reqDocumentEvent' response='Ens.Response' height='2000'
width='2000' >
```

9

BPL对象



10

BPL活动-调用(call)

- 同步调用
  —应该设置超时。超时后收到的响应会丢弃

```
Set tSc= ..SendRequestSync("Route and Transform Message", tRequest, .tResponse)
```

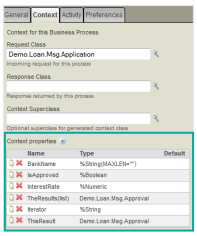
- 异步调用
  —不等待返回消息
  —无法设置超时
  —如果后面用Sync等待返回消息，收不到消息将永远等待

```
Set tSC=..SendRequestAsync("Demo.Loan.FindRateFileOperation",tSendReply,0)
```

11

业务流程上下文(Context)

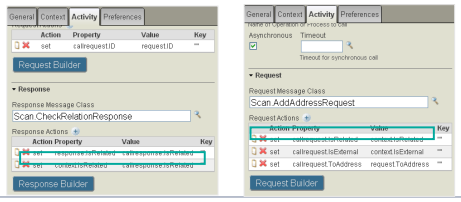
- 保存流程内部信息的对象.
  —Persistent.
- 可以在流程的任何位置访问
- 可以是任何类型
- 可以是单一值，List, 或者Array
- context的生命周期：它只在BP的session未结束时存在



12

使用上下文

Save information to use later. 最常见的场景是保存上一个call的callresponse，给下一个call的callrequest



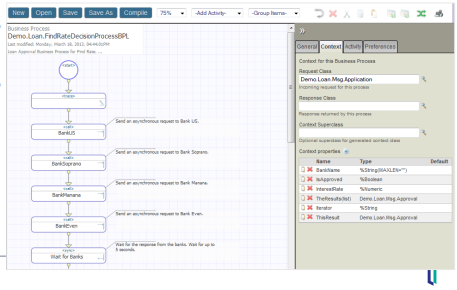
13

业务协调和流程管理

14

BPL示例-Demo.Loan.FindRateDecisionProcessBPL

- https://github.com/OneLastTry/irishealth-ensdemo
- https://github.com/grongierisc/InstallEnsDemoLite

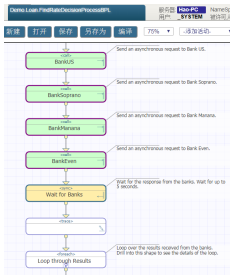


15

BPL活动-Sync（同步）

- 除了同步调用和异步调用，有第三种方式控制调用流程：Sync通常用于等待多个异步调用的结果，使用"类型"和"timeout"控制：
  - 类型设置为"Any":收到第一个返回消息，无论是从哪个调用返回的，程序继续sync后的执行。后面收到的其他返回直接丢弃，如同是超时。
  - 类型设置为"All": 在timeout设置的时间内，接收所有的调用返回消息。timeout设置后收到的返回消息直接丢弃。
  - 如果没有设置timeout, Sync有可能一直等待下去，比如type=all时没有收到所有消息，type=any时没有收到任意返回。

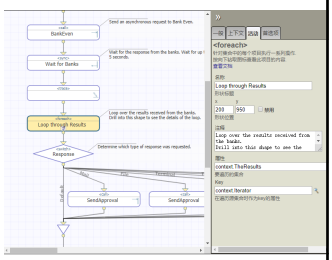
Demo.Loan.FindRateDecisionProcessBPL



16

BPL活动: foreach

- 通常是用于集合类型的对象处理，比如list或者Array: 比如右边例子把4个响应消息append到一个list型的context，由最下方的<for each>活动循环处理。
  - 属性
  - key:可以设置初始值，如果未设置从1开始，哪怕是%String类型。

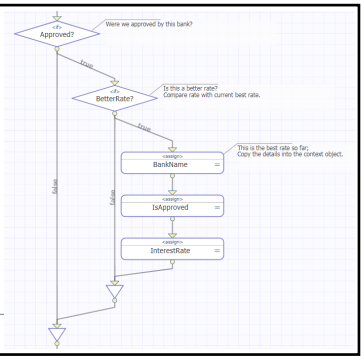


17

BPL活动:if, assign, join

if, assign, join

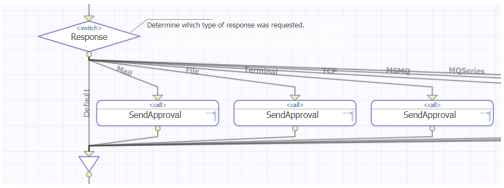
- Demo.Loan.FindRateDecisionProcessBPL



18

BPL活动:switch

Switch, join



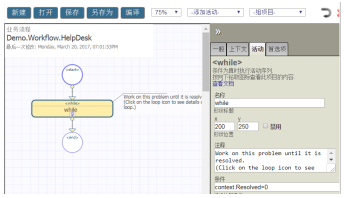
19

其他的activity项目

20

BPL活动: while,until,break,continue

- while定义了以系列的动作，只要条件判断不满足，就一直循环执行下去；
- while内部可以使用break, continue来修改流程
- until用法一样，区别在于while是先判断再执行循环，until是先执行循环再判断



21

BPL活动(activites)

- Call, Sync
- Rule(在别的章节介绍)
- Transform (在别的章节介绍), XPATH, XSLT
- Code, Assign
- Reply,
- SQL
- Trace, Alert
- Continue, Break, Delay, Empty

22

BPL活动: reply

reply

—默认整个BPL执行结束后才会发送返回消息给调用者， 如果执行中途要返回消息， 可以使用'reply'活动

23

BPL活动: trace

trace

- 当选中组件设置的"在前台运行"，跟踪消息会写到terminal;
- 当选中组件设置的"跟踪消息计入日志",跟踪消息会写到Event Log;
- 会显示在消息跟踪窗口

<trace value=""received application for "_request.Name'/>

24

BPL活动: milestone

milestone

- 类似trace的作用，区别在milestone只在BP运行时存在。BP退出后，milestone产生的消息都被清除
- 经常被用于诊断目的，用于显示一个正确运行的长时间的BP
- ^Ens.MileStone

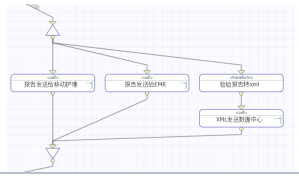
```
<milestone value="The applicant has been notified of the interest rate." />
```

25

DTL活动: flow, sequence

- flow包含一个或者多个sequence, 每一个可以看作一个线程，没有顺序关系
- 根据需要，可以把其中的某个或者某些sequence设置为"disabled"
- 只有所有线程结束，flow才往下执行，比如下例中thread1,thread2都执行完成后才会调用"E"

```
<process>
 <flow>
  <sequence name="thread1">
   <call name="A" />
   <call name="B" />
  </sequence>
  <sequence name="thread2">
   <call name="C" />
   <call name="A" />
  </sequence>
 </flow>
 <call name="E" />
</process>
```



26

DTL活动: 其他活动

branch, label, sql, transform, xPath, xsl….

- http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=EBPLR_elements

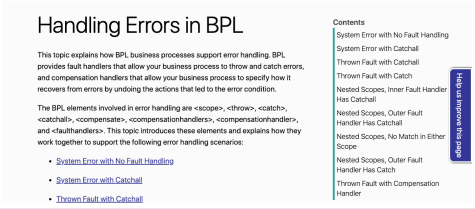
27

BPL错误处理

28

BPL错误处理

—https://docs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=EBPL_doc_errors#EBPL_doc_errors_catchall



Handling Errors in BPL

This topic explains how BPL business processes support error handling. BPL provides fault handlers that allow your business process to throw and catch errors, and compensation handlers that allow your business process to specify how it recovers from errors by undoing the actions that led to the error condition.

The BPL elements involved in error handling are <scope>, <throw>, <catch>, <catchall>, <compensate>, <compensationhandlers>, <compensationhandler>, and <faulthandlers>. This topic introduces these elements and explains how they work together to support the following error handling scenarios:

- System Error with No Fault Handling
- System Error with Catchall
- Thrown Fault with Catchall

Contents

System Error with No Fault Handling
System Error with Catchall
Thrown Fault with Catchall
Thrown Fault with Catch
Nested Scopes, Inner Fault Handler Has Catchall
Nested Scopes, Outer Fault Handler Has Catchall
Nested Scopes, No Match in Either Scope
Nested Scopes, Outer Fault Handler Has Catch
Thrown Fault with Compensation Handler

Help us improve this page

29

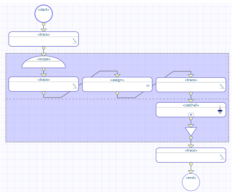
BPL错误处理

- 系统本身的错误处理
  - 出错时退出BP, 系统产生错误消息写入Event Log, 产生Errror类型的Event条目。

30

BPL错误处理-<catchall>系统错误

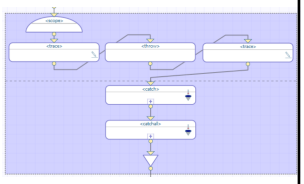
- 定义<Scope>(范围)和<catchall>捕获系统错误
  - —<Scope>定义<catchall>作用的范围，其中任何步骤出现的系统错误会被<catchall>捕获。
  - —捕获的系统错误存在%context.%LastError
  - —不会再向Event Log中产生错误事件日志
  - —在BPL的xml代码中， <catchall>的上层节点为<faulthandler>，它通常包含<catchall>和<catch>等内容，但<faulthandler>的名字不出现在BPL图形中。



31

BPL错误处理-throw

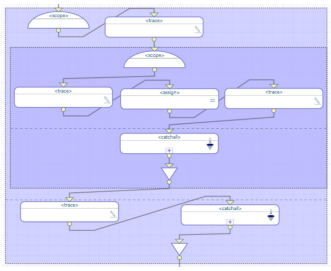
- 定义<Scope>(范围)和<catchall>或者<catch>捕获<throw>抛出的自定义程序错误。
  - —throw抛出的错误是一个小于255字节的字符串
  - —无论是<catch>还是<catchall>只工作在<scope>中。<scope>定义它们的作用的范围
  - —<catch>只捕获<throw>抛出的错误，<catchall>捕获<throw>错误和系统错误
  - —<catch>和<catchall>可以单独出现在scope中，但如果两个都使用，<catchall>一定在所有<catch>后面。
  - —不会再向Event Log中产生错误事件日志



32

DTL错误处理-嵌套

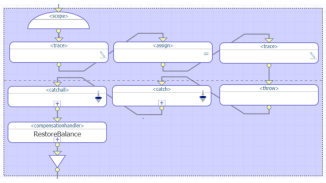
<scope>可以嵌套



33

DTL错误处理-Compensation

- <compensate>在<catch>和<catchall>调用<compensationhandler>
- <compensationhandlers>可以包含多个<compensationhandler>, 其中每一个可执行一个系列的回滚动作



34

DTL错误处理-Compensation

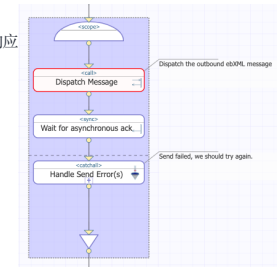
```
<scope>
 <assign property='context.MyBalance' value='context.MyBalance-1'/>
 <throw fault='"BuyersRegret"'/>
 <compensationhandlers><compensationhandler name="RestoreBalance">
 <trace value='"Restoring Balance"'/><assign property='context.MyBalance'  value='context.MyBalance+1'/>
 </compensationhandler> </compensationhandlers>
 <faulthandlers>
  <catch fault='"BuyersRegret"'><compensate target="RestoreBalance"/></catch>
  <catchall><trace value='"%LastFault "_..%Context.%LastFault'/></catchall>
 </faulthandlers>
</scope>
```

35

实际使用例子

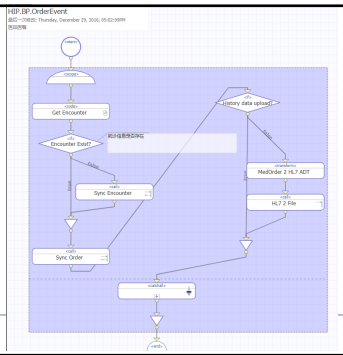
- 在<scope>中使用异步调用，用<sync>等待响应消息， 如果超时会出现系统错误， 用<catchall>捕获后设置BP的context.Ack=$$$NULLOREF
- 忽略<call>的红框

EnsLib.ebXML.process.MessageSender



36

实际使用例子


HIP.BP.OrderEvent
Get Encounter
Encounter Exist?
Sync Encounter
Sync Order
MedOrder 2 HL7 ADT
HL7 2 File

37


InterSystems
Creative data technology
Thank you

38