Exception handling and recovery for WebSphere Process Server asynchronous service invocation

Question & Answer

Question

How do you handle exceptions during synchronous and asynchronous outbound processi both invocation styles?

Answer

Exception handling and recovery are important disciplines for application developers. This FAQ basics and recommendations for handling runtime exceptions and building a stable recovery scer on making a decision between using a synchronous or asynchronous invocation style - the advan

Overview of invocation patterns and exception types

Before starting with the guide, read the following notes on synchronous and asynchronous invoca

- The synchronous invocation pattern makes a request to the target and receives the response i

- The asynchronous invocation pattern sends the request and receives the response in different operation) or can be received explicitly by the client certificate authority (CA) (deferred resp

Exception handling in WebSphere Process Server and WebSphere Enterprise Service Bus are div

- The business exceptions are declared in a methods signature (for example, in faults or in a Ja the application. These exceptions are passed back to the client and are wrapped by a `Servi`

- More important for further discussion are system exceptions (also called runtime exc the `ServiceRuntimeException` and are returned to the client for further inter
- `ServiceExpirationRuntimeException` marks an asynchronous call as ex messages.
- `ServiceTimeoutRuntimeException` is thrown when an expected response deferred response invocation style.
- `ServiceUnavailableException` is thrown when an asynchronous outbound call (us
- `ServiceUnwiredReferenceRuntimeException` indicates that the service referenc

↑ Back to top

Handling exceptions for synchronous invocations

During a synchronous invocation, both the client and server run in the same thread; the target ser in the case of a one-way operation. The exception can be a business or system exception. When y exception handler, for example:

```
try {
  someService.someOperation(...);
}
catch(ServiceRuntimeException e) {
  // handle runtime exception
```

```
}
```

This catch handler captures the exception (here a runtime exception) and allows you to handle it exception than `ServiceRuntimeException`, use the following code snippet, which shows l

```
try {
  someService.someOperation(...);
}
catch(ServiceUnavailableException e) {
  // service was unavailable
}
```

Calling any service and capturing an exception at run time should include mechanisms to retry th or to generate a fault and handle the exception in a Business Process Execution Language Java code. If it is used in a plain old Java object (POJO), you can write additional code to generate a fault and then use a fault handler and compensation. Use the following code sn exception identifiers:

```
raiseFault(new QName(
  "http://anyNamespace","SomeExceptionIdentifier"));
```


For details on faults and exception handling, visit the Business Process Management Samples Ga depends on the context and can be fulfilled by different techniques:

- Handle the exception in place (where the invocation is executed).
- Generate a BPEL fault and use a fault handler in a BPEL process.
- If the current process or POJO is called by another component, determine if the exception ha components, then send a fault back to them using a response message.
- Change the invocation style to asynchronous and use the failed event manager (read next sec

↑ Back to top

Handling exceptions for asynchronous invocations

This section details the traps and pitfalls of exception handling and lists the advantages of using t

The client and service provider (server) run in different threads. On both threads, exceptions can network connection, or the service provider might fail in case of another runtime exception. In th programming model, the client does not receive the server side exceptions because they are not p client is a business process component, and the target service returns a system exception. This ex process designer to handle system exceptions and to model recovery mechanisms.

Automatic retry mechanism 11.10

SCA uses the service integration bus (SIBus) to transport messages between components. As soo creates a queue on the SIBus for each import artifact component (outbound processing and servic system exception), the SIBus tries to resend the message at least five times. The following article threshold:

Recovering from failed asynchronous SCA service invocations on WebSphere Process Server

Recovery exception destination and database

In WebSphere Process Server and WebSphere Enterprise Service Bus, by default, one sy events (events that reached the retry limit and are archived). The destination has the follo

```
_SYSTEM.Exception.Destination.<nodeName>.<serverName>-
SCA.SYSTEM.<cellName>.Bus
```

In WebSphere Process Server, all failed messages are routed to a recovery exception dest

```
WBI.FailedEvent.WPS.<serverName>
```

Failed event manager subsystem

For each failed message a "failed event" is generated and registered in a recovery database (a set WebSphere Enterprise Service Bus common database tables). The failed event represents the bas manager application is the tool of choice to manage all failed events.

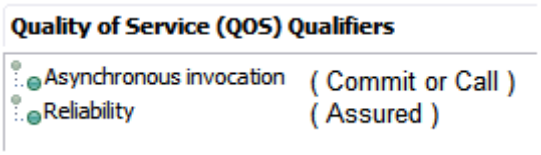
No runtime data is lost during the messaging and failed event process because the entire data flo This includes the request and response message handling as well as exception handling by the fai subsystem, it is unnecessary to implement specific recovery and exception handling mechanisms

Enabling asynchronous invocation and failed event management

To enable a service for asynchronous invocation, adjust the component artifacts interface and ref

1. In WebSphere Integration Developer, in the assembly diagram, select the component.
2. On the **Properties** tab, in the **Details** section, select the reference interface of the target service
3.

On the **Quality of Service (QOS) Qualifiers** tab, apply the following settings to the QoS para



The component that invokes the reference partner must also ensure that the service is invoked as Java code in a BPEL snippet or POJO, the method invocation must be adjusted as well:

```
Service someService = (Service)
  ServiceManager.INSTANCE.locateService(
    "<interfacePartnerName");

someService.invokeAsync("<someOperation>", <payloadBO>);
```

Manage and resubmit failed events

All failed events that are registered in the recovery database are managed by the failed ev Process Server and WebSphere Enterprise Service Bus products. Visit the information ce

Managing WebSphere Process Server failed events

↑ Back to top

Choosing the right invocation

There is no simple guideline that allows you to choose the right invocation style for every situati recovery must be treated as a unit in a complex application architecture that allows for different a help you make a more informed choice.

Asynchronous invocation

Choose asynchronous invocation in the following situations:

- You are calling a long-running BPEL process. In this situation, do not invoke an operation s the target process will finish and send the response. Invoking an operation synchronously in exception.
- You do not want to handle any system exceptions or use the failed event manager capability,
- You want to use a deferred response or callback mechanism to retrieve the response from th

- You invoke message-related SCA import components in general (JMS, MQ, or SOAP-JMS

Synchronous invocation

Choose synchronous invocation in the following situations:

- You expect a response message in a short and specific time frame and cannot handle failed e
- You want to handle exceptions explicitly using your own code or BPEL process design that
- You are using event sequencing on the service target. If an asynchronous invocation fails, th arrives in order. If the message is held on the system exception destination queue, you must Further processing of the target service depends on manual interaction.

Tips for recovery using the failed event manager

- No failed events are generated for synchronous invocations or any two-way business
- You must manually resubmit failed events to finish the invocation.
- You can adjust the retry threshold of the SIBus queue that is used for resending an ev
- The retries are performed immediately one after another, in general within seconds. I higher threshold does not solve the basic problem.
- Using a very high threshold value (one hundred and more) generates a lot of traffic o

↑ Back to top

Exception handling and recovery resources

The following resources on developerWorks provide additional information regarding invocatior recovery:

- Exception handling in WebSphere Process Server and WebSphere Enterprise Service Bus
- Recovery from failed asynchronous SCA service invocations on WebSphere Process Server

↑ Back to top

Related Information

Business Process Choreographer Samples Gallery

WebSphere Process Server Support

Cross-reference information

Product	Component	Platform	Version	Editi
WebSphere Enterprise Service Bus				

Product Synonym

WPS

Document Information

More support for:
WebSphere Process Server

Software version:
7.0, 6.2, 6.1.2, 6.1

Operating system(s):
AIX, HP-UX, Linux, Solaris, Windows, z/OS

Document number:
95453

Modified date:
15 June 2018

Contact and feedback