Artículo
Ricardo Paiva · 24 mar, 2022 Lectura de 9 min

[InterSystems IRIS por primera vez] Interoperabilidad: Crear componentes (Business Services)

Este artículo es una continuación de esta publicación**.**

En ese artículo, analizamos el desarrollo de *business processes*, que forman parte de los componentes necesarios para la integración del sistema y sirven como coordinadores de la producción.

En este artículo mostraremos la creación de un *business service*, que es la ventana de entrada de información para la producción.

- Producción
- Mensaje
- **Componentes**
  - **Business Services**
  - Business Processes (publicación anterior)
  - Business Operation

Y, finalmente, llegamos al último artículo de "¡Utilicemos la interoperabilidad!"

El *business service* proporciona una ventana de entrada para la información enviada desde fuera de IRIS, con o sin utilizar el adaptador para I/F externo.

Hay tres tipos de business services en la muestra (los enlaces entre paréntesis son enlaces al código de la muestra):

1. *Business services* para archivos que utilizan adaptadores de entrada (Start.FileBS)
2. *Business services* para servicios web que utilizan el adaptador de entrada SOAP (Start.WS.WebServiceBS)
3. *Business services* llamados por procedimientos almacenados o REST sin utilizar adaptadores (Start.NonAdapterBS)

Los diferentes métodos de conexión que se utilizan para introducir la información solo aumentarán el número de *business services*; sin embargo, el procesamiento realizado dentro de un *business service* es

Crear un mensaje de solicitud que se enviará usando información introducida externamente y simplemente llamar al business component

Sin ningún esfuerzo.

Ahora, describiremos cómo crear componentes que utilizan adaptadores de entrada de archivos.

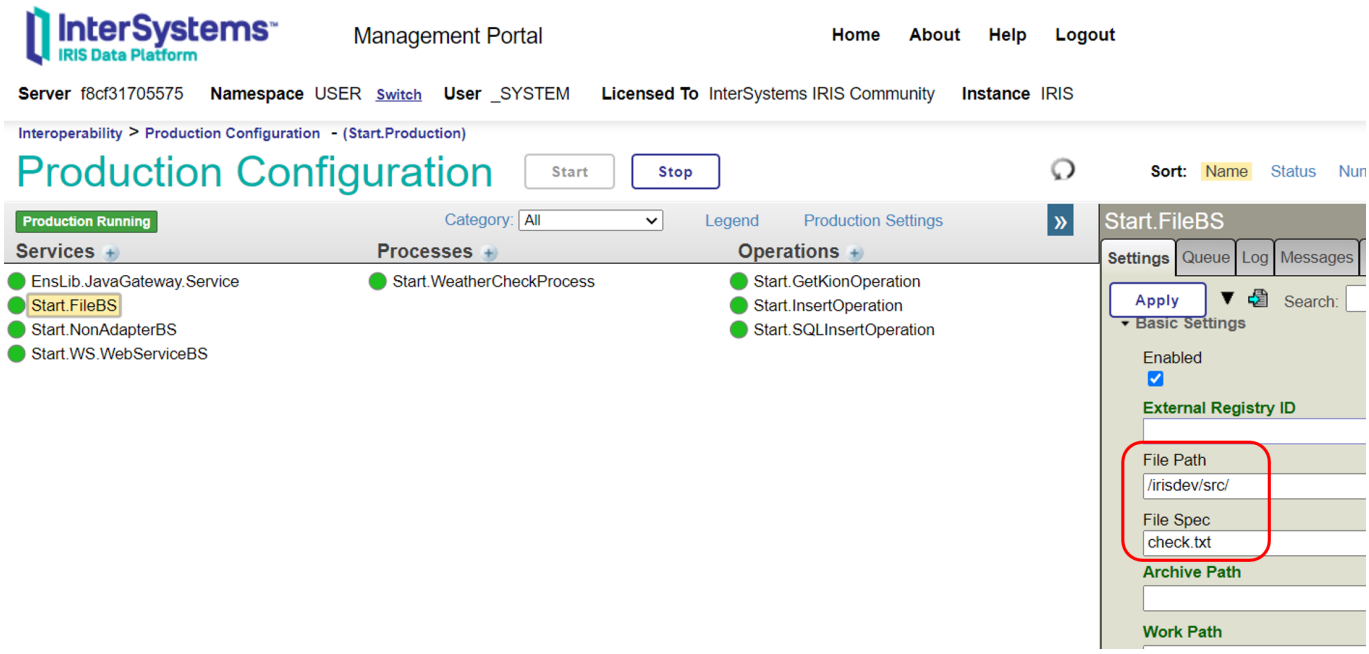
Los business services se escriben en *scripts*, que pueden crearse en VSCode o Studio.

1. Business services para archivos que utilizan adaptadores de entrada (Start.FileBS)

Si creas una clase en VSCode, deberías crear una clase que hereda de Ens.BusinessService. Para los adaptadores, puedes utilizar el parámetro **ADAPTER** como nombre de la clase ADAPTER (por ejemplo, especificar una clase de adaptador de entrada de archivos).
Si no utilizas el adaptador, no será necesaria ninguna configuración.

```
Class Start.FileBS Extends Ens.BusinessService
{
Parameter ADAPTER = "EnsLib.File.InboundAdapter";
```

En el adaptador de entrada de archivos, puedes especificar el directorio que se supervisará en Settings→File Path para el *business service* de la producción.



Si el archivo localizado en "File Path" coincide con la información especificada en "File Spec", este abre el archivo como un objeto de flujo. Lo define como la primera variable cuando se llama al business service **ProcessInput()**.

Cuando se inicia **ProcessInput()**, se llama automáticamente a **OnProcessInput()**. OnProcessInput() se transmite directamente a **OnProcessInput()** con los parámetros transmitidos a ProcessInput().

En **OnProcessInput()** la sentencia inicial recibe la información del objeto *file stream*, que se transmite como primer parámetro, después crea un mensaje que se dará al siguiente componente, escribe el proceso de llamada al siguiente componente y completa la lógica básica.

【Memo】
En el caso de Studio, inicia el Asistente (Wizard) de Business Services en el menú de New Creation, selecciona el adaptador y haz clic en el botón Finalizar.

La definición del método **OnProcessInput()** es la siguiente:

```
Method OnProcessInput(pInput As %Stream.Object, Output pOutput As %RegisteredObject)
As %Status
```

pInput es proporcionado con una instancia de la clase **%Stream.FileCharacter** para archivos de texto o de la clase

%Stream.FileBinary para archivos binarios.

En el ejemplo, se introducirá un archivo en formato texto, y lo hemos escrito para aceptar solicitudes de varias líneas y una solicitud por línea.

La propiedad AtEnd se establece en 1 cuando se detecta EndOfFile. Puedes utilizar esta propiedad para detener el bucle.

En un bucle, leemos las líneas utilizando el método **ReadLine()**, que nos permite obtener información sobre los contenidos del archivo una línea cada vez (consulta la documentación para más información sobre el adaptador de archivos).

Escribe el mensaje, recuperando información línea a línea. Después, ejecutamos el método **..SendRequestAsync()** , que llama al resto de los componentes.

Cuando se ejecute el método, el primer parámetro debería ser el nombre del componente al que quieres llamar en forma de *string*, y el segundo parámetro debería ser el mensaje de solicitud que hayas creado.

Ten en cuenta que **..SendRequestAsync()** es una llamada asincrónica y no espera recibir una respuesta.

Nota: También existe SendRequestSync() para las llamadas sincronizadas.。

El código de ejemplo es el siguiente:

Referencia: explicación del uso de la función $piece() en el texto de ejemplo anterior

$piece(" string " , " delimiter mark " , " position number ")

La función para establecer/obtener una string con un delimitador, en el ejemplo, para obtener el primer y segundo valor de datos separados por comas, se escribe con la siguiente sintaxis:

```
set request.Product=$piece(record,",",1)
set request.Area=$piece(record,",",2)
```

Ahora, revisemos la función de Start.FileBS tal y como aparecía en la descripción anterior.

En la producción del ejemplo, el "File Path" se estableció en **/irisdev/src**, y el "File Spec" se estableció en **check.txt**. Prepara el mismo directorio o cámbialo a otro y registra los datos de la muestra en el archivo check.txt con el siguiente formato: nombre del producto adquirido, nombre de la ciudad.

※Si estás utilizando el contenedor de muestra, cambia el nombre de [Test-check.txt] en el directorio src en el directorio que se creó con el clon de git.

2. Business services para servicios web que utilizan el adaptador de entrada SOAP (Start.WS.WebServiceBS)

Posteriormente, describiremos la creación de business services para los servicios web.

La clase de Business Service para servicios web actúa como proveedor de servicios web = servidor de servicios web.

En el ejemplo, tenemos dos parámetros en el método del servicio web para esta producción de ejemplo, para tener información que se envía desde el cliente del servicio web. El método web utiliza los datos introducidos en los parámetros para crear una clase de mensaje y llamar a otros componentes.

Cuando se define una clase de servicio web, se crea una pantalla de prueba. Sin embargo, no se muestra de

forma predeterminada.

Inicia sesión en IRIS (o inicia un terminal), ve al *namespace* donde se encuentra la producción y haz lo siguiente:

Para tu referencia：Acceso al catálogo y a las páginas de prueba

Esta es una configuración de código de ejemplo en el ajuste, donde el contenedor se inició con **docker-compose up -d** (ejecutar en el namespace %SYS)

```
set $namespace="%SYS"
set ^SYS("Security","CSP","AllowClass","/csp/user/","%SOAP.WebServiceInfo")=1
set ^SYS("Security","CSP","AllowClass","/csp/user/","%SOAP.WebServiceInvoke")=1
```

【Atención】 Ten en cuenta que la frase distingue entre mayúsculas y minúsculas y debe escribirse con cuidado. Además, según el namespace en el que se utilice el producto, cambia el script especificado. La oración del ejemplo fue escrita considerando que el ejemplo se importa en el namespace USER.
Si importas el código del ejemplo en el namespace ABC, el cuarto subíndice debería ser "/csp/abc/".

Cuando se haya completado la configuración, ve a la siguiente URL:

http://localhost:52773/csp/user/Start.WS.WebServiceBS.cls

Si quieres ofrecer el WSDL a tu cliente de servicios web, especifica WSDL=1 al final de la siguiente URL

http://localhost:52773/csp/user/Start.WS.WebServiceBS.cls?WSDL

3. Business services llamados por procedimientos almacenados o REST sin utilizar adaptadores（Start.NonAdapterBS）

A continuación, introduciremos el *Business Service* sin adaptadores (Start.NonAdapterBS).

Para los *business services* que utilizan adaptadores, el adaptador llama al método ProcessInput() del business service para detectar la información.

Si no utilizas adaptadores, puedes seguir llamando al método ProcessInput(), pero este método no es público. Por lo tanto, si implementas un business service que no utiliza adaptadores, tendrás que considerar ProcessInput().

La muestra utiliza los dos métodos siguientes:

- Procedimientos almacenados（Start.Utils）
- Clase dispatch para REST（Start.REST）→Este es el servicio que ejecutamos en este artículo.

Este es un ejemplo del procedimiento almacenado.

Después de añadir un business service (Start.NonAdapterBS) que no utiliza adaptadores a la producción (estado incorporado en la muestra), ejecuta el siguiente procedimiento almacenado

call Start.Utils_CallProduction('piroshki','Russia')

Un rastreo de los resultados de la ejecución es el siguiente:

A continuación, se muestra un ejemplo de creación de una clase dispatch para REST:

El XML descrito en el mapa de la URL de XData define qué métodos se llaman como respuesta a la URL en el momento que se realiza la llamada de REST.
En el ejemplo se describe una definición que llama al método **WeatherCheck()** cuando se proporcionan las URL's

del **/weather/first parameter (purchased product name)/ second parameter (name of the city)** en la **solicitud GET**.

```
<Route Url="/weather/:product/:arecode" Method="GET" Call="WeatherCheck"/>
```

A continuación, define la URL base para la URL anterior en la pantalla de Configuración de la ruta de acceso a la aplicación web del Portal de administración, y estará completo.

Consulta este artículo para más detalles sobre la configuración.

Cuando esté listo, intenta ejecutar la información usando un *business service* que te permita enviar la información REST.

Ejemplo) http://localhost:52773/start/weather/Takoyaki/Osaka

Si no utilizas un adaptador, como ProcessInput() no se puede llamar directamente desde fuera, hemos creado un objeto para el business service en la lógica que se ejecuta por medio de REST o procedimientos almacenados (utilizando el método CreateBusinessService() de la clase Ens.Director) y llamado a ProcessInput()

Si utilizas un adaptador, este detecta la entrada y almacena la información en un objeto único y la transmite al business service. En cambio, si no utilizas un adaptador, el resto es prácticamente igual, la diferencia solo se encuentra en la parte del proceso mencionada anteriormente.

El business service está diseñado simplemente para utilizar la información que se introduce fuera de IRIS para crear mensajes de solicitud y llamar a los business components.

Durante la producción de la muestra, pudimos ver lo siguiente:

Los distintos componentes desempeñan diferentes funciones en la ejecución de una producción (business services, business processes, business operations).

Para transmitir información entre componentes, utiliza el mensaje.

Los mensajes se almacenan en la base de datos a menos que se borren y, por ello, se pueden rastrear en cualquier momento.

Algunos adaptadores facilitan el proceso que rodea a la conexión.

Estas son las operaciones básicas sobre cómo utilizar la interoperabilidad en IRIS.

También hay mapas de registros y herramientas de conversión de datos que son útiles para la entrada y salida de archivos CSV y otros archivos con formato específico.

Además, IRIS for Health también es compatible con las transmisiones FHIR y HL7 (incluyendo SS-MIX2).

Estaré encantado de explicarlo en otra publicación. Si tienes algo interesante que compartir, ¡deja un comentario!

Por último, también hay cursos de formación para aprender a utilizar la Interoperabilidad.

#API REST #Interoperabilidad #Principiante #Servicio empresarial #InterSystems IRIS #InterSystems IRIS for Health

componentes-business-services