Artículo
Mathew Lambert · Jun 15, 2020 Lectura de 7 min

REST en trozos

Esta es una guía para principiantes, para el desarrollo de servicios web RESTful con Ensemble.

Contexto

Antes de comenzar a leer esta breve introducción, lee la documentación de Ensemble, prestando especial atención al capítulo sobre creación de servicios y clientes REST con Ensemble ("Creating REST services and clients with Ensemble ")

El enfoque de esa documentación creo que es indiscutiblemente la forma más rápida y simple de crear servicios RESTful. Como principiante, leí la documentación y me quedaron varias dudas. Este breve artículo enumera esas preguntas, junto con mis humildes respuestas

Cuando hago referencia a documentación, me refiero a Ensemble 2016.2 y posteriores.

¿Qué convierte un servicio Ensemble en RESTful?

Todos aprendimos que un servicio Ensemble se convierte en algo específico (como un servicio de archivos) al vincular un adaptador entrante específico a un servicio genérico. Por ejemplo, EnsLib.HL7.Service.Standard en combinación con EnsLib.File.InboundAdapter construye un servicio de archivos HL7. Muy bien. Incluso si esta lógica no aplicaba totalmente a los servicios SOAP, es una forma habitual de construir servicios.

¿Pero funciona este esquema de "servicioestándar+adaptadorespecífico" con servicios RESTful? No del todo. La documentación indica:

Para desarrollar un servicio REST, se debe extender la clase EnsLib.REST.Service. ⋯ Implementar un servicio REST es una extensión de implementar la clase %CSP.REST e implementar un servicio HTTP.

Bien. Es aún más fácil de lo que pensaba. Al generar una sublase de EnsLib.REST.Service lo obtuve todo gratis. Podemos contestar a la pregunta: cualquier subclase de EnsLib.REST.Service es un servicio RESTful de Ensemble. Es así de directo. ¿O no?

Parece así de simple hasta que lees la siguiente nota:

> En la mayoría de los casos, recomentamos usar el puerto CSP de Ensemble y no un puerto especial. Usar el puesto CSP de Ensemble permite utilizar un servidor web completo para ofrecer solidez y seguridad.

Bien. Sigo manteniendo mi recomendación. ¿Pero es la seguridad la única diferencia? Bueno, lee por favor toda la sección posterior a la nota citada.

Al final de la sección encontrarás las instrucciones adecuadas sobre qué hacer exactamente en lugar de crear subclases de EnsLib.REST.Service.

Puedes añadir el servicio empresarial EnsLib.REST.Service directamente a la producción usando el portal de administración, pero solo puedes agregar la subclase %CSP.REST indirectamente. Para añadirla, sigue este procedimiento:

1. Añade a la producción un servicio empresarial personalizado que sea una subclase de Ens.BusinessService. Esta subclase implementa el método OnProcessInput().
2. Llama al método Ens.Director.CreateBusinessService() desde la subclase %CSP.REST. El método CreateBusinessService() habilita el servicio empresarial personalizado y le devuelve un puntero.
3. Usa el puntero al servicio empresarial para llamar al método OnProcessInput().

Así que el servicio RESTful de Ensemble es un único concepto, pero implementado en dos trozos (clases). Por lo tanto, la respuesta a la pregunta es: cualquier servicio genérico puede ser RESTful. O: una única subclase de %CSP.REST puede poblar (casi) cualquier servicio Ensemble como un servicio web RESTful incluso si la clase del servicio no se implementa bajo el concepto de servicios web o RESTful.

¿Cómo se vincula el %CSP.REST con elementos de configuración?

Sin duda, existen un sinfín de formas de vincular la subclase %CSP.REST al nombre del elemento de producción iniciado por el Ens.Director.CreateBusinessService(). La documentación tiene una larga lista de recomendaciones. La podrás encontrar en la sección *Defining the URL Using UrlMap and EnsServicePrefix* (Definición de la URL usando UrlMap y EnsServicePrefix). El resultado final es: la URL de tu servicio RESTful debería tener este aspecto /{{cspapproot}}/{{ensconfigitemname}}//{{restparameters}}. El trozo del medio de la URL vincula la subclase %CSP.REST con el elemento de configuración.

¿Cuántas clases de servicio hay?

Debido a que un servicio en la forma de programar de Ensemble es capaz de procesar un único tipo de mensaje, deberías tener tantas clases de servicio en tu biblioteca de clases como tipos de mensajes distintos tengas. Eso significa que las clases de servicios implementan trozos del procesamiento de mensajes comunes de un servicio web.

¿Cuántas instancias de servicio hay?

¿No es lo mismo que las clases de servicio? No. La misma clase de servicio puede instanciarse en la misma producción bajo distintos nombres, dependiendo de varias condiciones. Como una producción de HL7. Eventos disparadores, orígenes de mensajes y prioridades de mensajes, todos pueden dividirse para identificar los trozos de una producción Ensemble

¿Cómo se construye la solicitud HTTP?

Los servicios RESTful son en definitiva servicios HTTP. HTTP tiene varias formas de llevar trozos de información por toda la red. El cliente crea esos trozos como parte de la generación de la solicitud HTTP. Del lado del servidor, esos trozos se vuelven a juntar para construir la solicitud Ensemble. ¿Cuáles son las opciones?

1. Los trozos se codifican en la ruta URL. Los trozos de información son parámetros posicionales de la ruta de la URL. Esta es la forma REST de pasar parámetros. Si fuera SOAP, a este estilo le llamaríamos RPC. Los parámetros del procedimiento se pasan al servicio dentro de la ruta de la URL. Este estilo, sin embargo, sufre de varias limitantes. La más importante para una solicitud compleja es que la sintaxis de la URL es una limitación.
2. Los trozos se codifican en la parte de la consulta URL. Casi igual a la codificación de la ruta de la URL, pero en lugar de tener el trozo de datos en la ruta (antes de "?"), lo ponemos en la consulta (después de "?"). ¿Cuál es la diferencia? Si tienes un trozo de datos obligatorio, puedes ponerlo en la ruta de la URL. Pero si tienes trozos de datos opcionales, es mejor colocarlos en la parte de la consulta. Esto es simplemente debido a que la ruta URL es una vinculación posicional, mientras que la consulta es un par clave-valor. Si tienes un campo vacío en la ruta de la URL, se verá así: “ /some/url/path//with/emty///pieces ” . Mientras que si la consulta guarda los datos, podría ser así “ /some/url/path?qp=&data=& ” .
3. Los trozos se codifican en datos de formulario. Eso es un poco más avanzado que la codificación de parte de la consulta. De hecho, del lado del servidor no hay diferencia.
4. El cuerpo de la solicitud HTTP guarda los trozos. Para un intercambio de datos de tipo documento, esta es la mejor opción. Del lado del servidor se proyecta como un flujo. Cuando hablamos de una interfaz verdaderamente RESTful, debemos esperar el flujo que guarda una cadena JSON.
5. Los trozos se guardan por archivo adjunto. Sin duda, es la única opción cuando tienes varios adjuntos binarios en una única solicitud.
6. Y/O cualquier combinación de las opciones anteriores…

¿Cómo proceso la solicitud HTTP?

Obviamente, como tenemos distintas soluciones para construir la solicitud, tenemos distintas opciones para procesar la solicitud.

- Procesar trozos de la ruta URL. La clase %CSP.REST ofrece el mecanismo para definir el parámetro posicional en la ruta de la URL y mapearlo al argumento de la llamada al método de clase estática. A primera vista, parece como si se pasara por nombre, pero en realidad puedes usar cuando quieras en la firma del método el número variable de sintaxis de argumentos (argv…), lo que significa nuevamente un paso posicional. Como ya sabes, el paso de argumento posicional te ofrece un único vector argumento en lugar de múltiples parámetros. Requiere de atención especial en la implementación del método, pero brinda una gran flexibilidad al programador. En el método OnProcessInput de clase de servicio, el argumento pInput guardará un vector argumento
- Procesar la ruta de la consulta o formar trozos de datos. Esos trozos se pasan dentro de la propiedad multidimensional %request.Data. Consulta la documentación de la clase %CSP.Request para ver cómo acceder a los datos dentro de la propiedad
- Procesar un flujo de cuerpos de solicitudes HTTP. Si el tipo de contenido del cuerpo del mensaje es “ application/json ” , entonces estamos en un excelente posición, porque tan solo necesitamos leer correctamente el flujo %request.Content y hacer un análisis hacia un objeto dinámico
- ¿Adjunto binario? Sí, falta ese trozo…

Como cualquier combinación es válida, podría existir una situación en la que tengas todos estos trozos para construir un objeto de solicitud Ensemble.

¿Cómo pruebo mi servicio?

Tiene dos piezas distintas. ¿Cómo genero mensajes de prueba y cómo monitorizo el tráfico de red? El segundo trozo puede resolverse con facilidad. Como todas las solicitudes pasan por la puerta de enlace CSP estándar, puedes activar HTTP Trace en el Portal de Administración del Sistema. ¿Pero qué pasa con el mensaje? Los ejemplos de la documentación usan "curl". Se trata de una herramienta de línea de comandos disponible para (casi) cualquier plataforma. Incluyendo también Windows 10. Sin duda, para quienes usan una ventana de terminal o una interfaz de línea de comandos a diario, se trata de una opción. Algunas alternativas:

- soapUI si ya sabes usarlo. Funciona con RESTful⋯
- postman
- Tu navegador preferido. Hoy en día los navegadores hacen las delicias de los desarrolladores con herramientas para pruebas, depuración y rastreo.
- Una página HTML estática. Puede hacer un GET para probar fácilmente el pasaje de parámetros codificados en una URL o puede hacer POST para datos de formulario.
- Página HTML con AJAX. Con AJAX puedes hacer GET, POST y PUT, puedes pasar una URL codificada, datos de formulario y hasta el cuerpo de una solicitud HTTP. AJAX es una parte estándar de la mayoría de las librerías de conveniencia populares de JavaScript, como jQuery y AngularJS.

¿Es REST o RESTful?

Tuve que hacer esta pregunta porque a veces es importante hablar en el mismo idioma que usa la industria. Ahora REST es un concepto, mientras que RESTful es una implementación que representa el concepto. Así que creo que la respuesta es RESTful, porque nuestra implementación respeta cada parte del concepto REST.

#API REST #JSON #Ensemble