# WEB-SOCKET

Echemos un vistazo a las formas de crear aplicaciones Java con el uso de WebSockets. En este capítulo, primero debemos comprender el protocolo WebSockets, el estilo de comunicación utilizado por el protocolo WebSocket, y luego describir los controladores de puntos finales de WebSocket mediante la API de Java, que se describe en un estándar llamado JSR 356.

Ahora, los controladores de puntos finales de WebSocket, hay dos tipos de controladores, como descubriremos en este capítulo: controladores de clientes y servidores, por lo que debemos estudiar ambos, comprender sus ciclos de vida, observar cómo podrían producir y consumir mensajes. , manejar errores. Y luego hay una parte de este capítulo que trata sobre formas de codificar y decodificar mensajes.

En términos generales, los mensajes utilizados por WebSockets pueden ser cualquier cosa. Pero como ejemplo, estamos usando aquí mensajes JSON. Aunque técnicamente hablando, podría haber usado cualquier otro formato de datos. No tenía que ser JSON en absoluto. Pero bueno, era solo un ejemplo. Supongo que es un ejemplo de uso frecuente, porque en muchas ocasiones los clientes de WebSocket se implementan en JavaScript, que veremos. Pero también estamos buscando clientes Java para WebSockets. Bueno, comencemos.

En primer lugar, ¿qué es el protocolo WebSocket? ¿En qué se diferencia del protocolo HTTP? Verá, en las interacciones HTTP, está produciendo una serie de solicitudes del cliente a las que el servidor debe responder. El protocolo HTTP implica que el originador de la llamada, el originador de la solicitud, es siempre el cliente. Y un servidor está respondiendo a eso.

Entonces, si desea averiguar qué sucedió en un servidor, lo que tiene que hacer desde el cliente es seguir llamando, haciendo una especie de encuesta, invocando repetidamente al servidor y preguntando al servidor, ¿ya llegamos? ¿Que esta pasando? Y luego el servidor puede enviarle varias respuestas, que pueden ser respuestas intermedias, diciendo bueno, todavía estoy trabajando en algo. Y luego, eventualmente, una respuesta final, donde el servidor le dirá al cliente, esto es todo. Hemos concluido algún tipo de actividad en un servidor. Pero el cliente tendrá que seguir interrogando al servidor para averiguarlo. Así es como funciona el HTTP.

¿Qué pasa con WebSocket? El apretón de manos inicial del cliente y un servidor aún se realiza en HTTP. El cliente llama al servidor a través del protocolo HTTP. Pero luego el cliente cambia el protocolo a un protocolo diferente, al protocolo WebSocket. Una vez que se establece la conexión WebSocket, crea un enlace de dúplex completo entre un cliente y un servidor.

Esta conexión, a diferencia de HTTP, no está cerrada. Permanece abierto. Y tanto el cliente como el servidor obtuvieron las mismas capacidades para enviar información a través de él. Por lo tanto, el autor de la llamada a través de la conexión WebSocket no tiene que ser el cliente. Podría ser el servidor. Sin un cliente sondeando y solicitando una respuesta, el servidor podría manejar algún evento, pase lo que pase en un nivel de servidor allí, cualquier cosa, maneja el evento en un servidor y luego invoca al cliente a través de la conexión de socket, diciéndole al cliente lo que acaba de pasar. Y esa es la clave para comprender la estrategia de empujar en lugar de tirar.

Por supuesto, eso significa que para admitir dicha interacción, un servidor tendrá que dedicar un controlador para cada socket abierto. Entonces, en otras palabras, puede resultar ser una técnica de codificación que consume bastantes recursos, porque el servidor tiene que mantener conexiones abiertas, sockets abiertos con los clientes en todo momento. En cuanto a la escalabilidad, los WebSockets presentan un problema.

Pero en términos de la capacidad de un servidor para entregar instantáneamente un mensaje al cliente sin que el cliente lo solicite, bueno, sí, los sockets están haciendo el truco. Son más eficientes en ese sentido que el protocolo HTTP. Ah, también es el hecho de que el socket es, una conexión de socket le permite transportar no necesariamente texto, sino datos binarios, por ejemplo, lo que nuevamente, en comparación con el protocolo HTTP basado en texto, podría ser una mejora. Por lo general, WebSockets sería utilizado por clientes que residen en navegadores web para JavaScript o tal vez aplicaciones móviles, aunque, por supuesto, la persona que llama WebSocket también podría ser una aplicación Java.

Ahora, intentemos comprender lo que sucede con un ciclo de vida de WebSocket. Hay dos lados en la interacción WebSocket. Hay un controlador WebSocket de servidor y cliente. Y son realmente simétricos. Realmente son espejos el uno del otro. Para las operaciones que un controlador de WebSocket, lo que sea, es un controlador de cliente o servidor, no importa para las operaciones que deben admitir: OnOpen, OnClose, OnMessage, OnError.

Los ejemplos en esta página nos muestran por un lado, un punto final del servidor. Un ejemplo está escrito en Java. Ese es ese. Y por otro lado, un extremo del cliente, y en esta página en particular, en realidad está escrito en JavaScript. Pero a pesar de que técnicamente es un lenguaje de programación diferente aquí, todavía ves una sorprendente similitud allí, cómo se reflejan entre sí cuando miras estas funciones onopen, onclose, onmessage y onerror aquí.

Entonces están registrados en ambos lados de esta conversación. Y es precisamente porque el autor de la llamada puede ser el cliente o el servidor. Son una especie de socios iguales en esa conversación. Entonces, por ejemplo, si un cliente del servidor de componentes realiza la llamada, la otra parte recibe la notificación onmessage. Entonces, el método OnMessage se invoca en el otro lado.

Si hay un error, se invoca un mensaje OnError. Cuando el socket está abierto y tal vez desee, no sé, inicializar algunos recursos en ese momento, se invoca el método OnOpen. Una vez que se cierra la conexión del socket, se invocará el método OnClose. Por lo tanto, existen los eventos del ciclo de vida que maneja para los controladores de puntos finales del cliente y del servidor.

Ahora, echemos un vistazo más de cerca a la descripción real del punto final de WebSocket. Puede asignar su controlador WebSocket a una URL. Bueno, ese es este punto final del servidor de anotaciones aquí, que vincula ese controlador a la URL. Y luego, dentro del controlador, anota métodos con anotaciones OnOpen, OnClose, OnMessage y OnError.

¿Cómo se supone que deben ser exactamente estos métodos? Echemos un vistazo a cómo se supone que se ven estos métodos, los métodos que manejan las llamadas OnOpen, OnClose, OnMessage y OnError, tanto para el extremo del cliente como del servidor. Hay ciertas reglas que debemos seguir.

Primero, echemos un vistazo a la operación OnOpen. Opcionalmente, si lo desea, puede aceptar un objeto EndpointConfig como argumento, que es un objeto de configuración. Puede exceptuar el objeto de sesión. Bueno, en realidad todos pueden aceptar el objeto de sesión como argumento, y supongo que cualquier otro parámetro personalizado si desea pasar los parámetros personalizados a dicha operación. Ahí es cuando estamos inicializando el socket OnOpen.

OnClose puede aceptar CloseReason como argumento. De nuevo, es opcional. Podrían ser parámetros personalizados. Podría ser CloseReason. También podría ser el objeto de la sesión, porque está disponible para todos los métodos. OnError debe aceptar un objeto Throwable, que representa el error, y puede aceptar parámetros personalizados adicionales, nuevamente, así como el objeto de sesión, que sigue siendo válido.

Y finalmente, OnMessage, ahora primero que nada, puede ser nulo, pero puede devolver un valor. Así que aquí tienes un par de variaciones. Es posible que desee que el mensaje, cuando lo reciba, responda inmediatamente a la persona que llama, pero no es necesario que lo haga. Por lo tanto, OnMessage puede devolver valor o puede anularse.

En cuanto a los parámetros de entrada, el mensaje puede ser cualquier mensaje binario o de texto. También puede contener parámetros personalizados. Entonces, debe pensar qué tipo de parámetros debería tener el método OnMessage. Puede crear más de un método OnMessage siempre que sea para un tipo diferente de parámetro, por ejemplo, texto, binario.

Hay otro tipo de parámetro que puede manejar OnMessage. Se llama mensaje pong, como en ping pong. Entonces, la otra contraparte podría enviarle un ping, ya sabe, enviar una llamada solo para verificar si el enchufe todavía está, ya sabe, todavía vivo, si la conexión aún es válida. Sí, y el otro lado puede responder con un ping-- sí, así que ping pong aquí, respuesta al ping. Entonces eso es lo que OnMessage puede manejar potencialmente. Estos son los posibles parámetros válidos para ello. Y como digo, opcionalmente, todos pueden aceptar el objeto de sesión WebSocket.

A continuación, la anotación PathParam le permite agregar un parámetro a la URL con la que abre el WebSocket. Cualquier operación dentro del socket puede hacer referencia a ese parámetro. Verá, tiene un nombre de identificación aquí. Para que puedan hacer referencia a ese PathParam usando ese nombre. Cualquier otra operación puede hacerlo.

Sin embargo, pensemos en ello por un momento. ¿Qué representan exactamente estos PathParam? No representa el mensaje que el cliente envía al socket del servidor o viceversa. Este PathParam es una URL con la que abre un socket. Entonces, una vez que se abre, permanece igual durante toda la sesión de socket. Está bien. Es algo constante. Si desea cambiar ese valor de parámetro, debe volver a abrir el zócalo. El cliente todavía tiene abierto otro socket entonces.

No es un mensaje que un cliente envía al servidor. No es un mensaje que un servidor envía al cliente. Estos podrían ser mensajes variables. Es solo una especie de parámetro de inicialización para que el cliente inicie una conexión con el socket. Pero luego todas nuestras otras operaciones en el socket, si desean tener acceso a ese parámetro, pueden hacerlo. Es opcional agregar parámetros personalizados a estas operaciones. Pero si desea que tengan acceso al parámetro de inicialización, simplemente agregue esa anotación PathParam y listo.

Cuando abre un WebSocket, inicializa una sesión entre su cliente y los controladores del servidor. Entonces eso comienza con el método OnOpen, termina con el método OnClose. Esa es la duración de la sesión. Dentro de esa sesión, siempre puede verificar si la sesión aún está abierta. Programáticamente, es el método isOpen. Eso es bastante sencillo.

Puede asociar la sesión con controladores de mensajes; recuerde, para controladores binarios, de texto y pong. Entonces estos son diferentes métodos de OnMessage. Puede proporcionar controladores diferentes separados para cada tipo de método OnMessage si lo desea.

Cuando cierra la sesión, supongo que debe proporcionar información sobre el motivo del cierre. Si no lo hace, se asume que NORMAL\_CLOSURE es el motivo de cierre. Pero puede cerrarlo con alguna otra notificación de por qué lo cerró. Entonces hay una razón ahí.

Y luego, también la sesión le da acceso a la información de seguridad. Por ejemplo, ¿cuál es el usuario actual? Obtenga la entidad de seguridad del usuario. Hablaremos más sobre la seguridad de Java EE en el último capítulo de esta clase. Así que todavía está por llegar. Hay bastante información allí. Pero de todos modos, bueno, así es como puedes acceder a eso desde un socket, una API de sesión.

Una cosa más, que es realmente interesante. Eche un vistazo a esta operación, getOpenSessions. Le permite a su controlador de socket en el servidor obtener acceso no solo a la sesión actual, porque la sesión actual, puede recibirla como parámetro en su método, correcto, sino también para obtener acceso a todas las demás sesiones actualmente activas. Por ejemplo, qué sucede si desea notificar todas estas sesiones al mismo tiempo. ¿Quieres enviar un mensaje a estas sesiones, o algunas de estas sesiones?

Entonces, el servidor quiere transmitir cosas, no solo a un cliente específico sino a otros. Recuerde que en la interacción WebSocket, la llamada no tiene que ser solicitada por el cliente. Por lo tanto, el servidor que ejecuta una línea de tiempo puede decidir que quiere obtener acceso a todas las sesiones del cliente y simplemente distribuir información a estos clientes. Y así es como se hace.

Sin embargo, en cuanto a la escalabilidad, hay una trampa. Obtiene información sobre todas sus sesiones dentro de un nodo de servidor Java EE determinado. Si está ejecutando el clúster, entonces un controlador de WebSocket que está en un nodo del clúster solo reconoce las sesiones que están abiertas y ese nodo. Pero si se está ejecutando en otro nodo de clúster, es felizmente inconsciente al respecto. Así que tienen conjuntos separados de sesiones. El cliente se conecta a un controlador específico en un servidor específico.

Así que sí, si estás en un escenario de un solo nodo, eso es bastante sencillo. Puedes enviar un mensaje a todo el mundo. Pero si está en un escenario de múltiples nodos, desafortunadamente, la especificación WebSocket realmente no es suficiente. No le dará acceso automáticamente a las sesiones que están en otro nodo del clúster. Los desconoce. Bueno, esa es una limitación que debes tener en cuenta.

Ahora, echemos un vistazo a cómo podemos enviar el mensaje del cliente al servidor o del servidor al cliente. Permítame recordarle que el método OnMessage (ya lo discutimos) puede simplemente devolver un valor del servidor al cliente, por ejemplo, o al revés. Pero esa es una respuesta instantánea. Es cuando recibe un mensaje, lo procesa y luego responde algo a la otra parte en una especie de respuesta inmediata.

Pero, por supuesto, los enchufes no tienen que dar estas respuestas inmediatas. Es posible que solo hagan una llamada porque quieren. Y así es como se hace. La llamada puede ser síncrona o asíncrona. Fuera de la sesión, debe obtener el objeto BasicRemote o el objeto AsyncRemote, por lo que uno de estos: comunicación sincrónica o asincrónica.

Si es BasicRemote, simplemente envía lo que quieras. Sí, entonces pueden ser clientes que envían cosas a un servidor. Puede ser un servidor que envía cosas a un cliente. Si es remoto asíncrono, entonces el método sendObject devuelve su estado de transmisión como un objeto futuro. Básicamente, le permite no esperar a que se produzca la transmisión, pero luego descubrir cómo fue esa transmisión al obtener el estado de ese objeto futuro.

Ese es el ciclo de vida de WebSocket, y la forma en que podríamos usarlo para pasar mensajes entre el servidor y los puntos finales del cliente. Cualquiera de los métodos de WebSocket: OnOpen, OnClose, OnMessage, OnError, no importa, ¿podría usar esa API para realizar la comunicación?

Ahora, echemos un vistazo a la siguiente parte importante del manejo, codificación y decodificación de mensajes de WebSocket. El ejemplo en esta página en particular nos muestra la codificación y decodificación de mensajes JSON. No tiene que ser JSON, pero, por supuesto, lo usan comúnmente otros tipos de componentes del lado del servidor, como los servicios REST. Es comúnmente utilizado por las aplicaciones cliente de JavaScript. Entonces, por lo tanto, es un ejemplo razonable para mostrar.

Aunque técnicamente hablando, el formato de los mensajes que intercambia a través de un socket podría ser cualquier cosa. No tiene que ser JSON. Pero si es JSON, que suele ser el caso, hay una API para ayudarlo a manejar las conversiones JSON, clasificar y desclasificar mensajes JSON. Y se llama API JSON-P, API de procesamiento JSON. Es una API Java estándar para procesar mensajes JSON.

Tiene varias formas de procesar estos mensajes. Son de transmisión, basados ​​en objetos. Y permítanme comparar estos dos primero aquí. Por lo tanto, la capacidad de transmisión de JSON-P implica que se trata de un objeto grande. Y no desea leer todo el objeto o escribir todo el objeto de una sola vez, ¿verdad? Prefieres usar una secuencia para procesarlo. Entonces JsonParser le permite leer datos JSON. Y JsonGenerator le permite generar los datos JSON, así que cree un nuevo objeto JSON y transmítalo a la otra parte a través de la red.

El lector y escritor, la API JSON basada en objetos, le permite tomar un mensaje JSON y extraerlo de la memoria como un objeto Java, o tomar un objeto Java y convertirlo, ordenarlo, en el mensaje JSON. Por lo tanto, es más adecuado para mensajes más pequeños en los que puede permitirse poner todo el mensaje en la memoria de una sola vez. Entonces, cuanto más grande sea el mensaje, más se inclinará a usar la API de transmisión. Cuanto más pequeño sea el mensaje, más se inclinará a utilizar el objeto-b

Puede realizar la codificación y decodificación de estos mensajes directamente dentro del controlador, el controlador WebSocket, el controlador WebSocket del cliente o el servidor. Eso es cierto, pero también puede codificar y decodificar estos mensajes en una clase de codificador/descodificador separada. Supongo que si tiene varios controladores de punto final que usan los mismos mensajes, tiene sentido dedicar codificadores y decodificadores separados para manejar los mensajes. Entonces compartes el código de manera más eficiente.

Como probablemente haya notado, aquí también hay una referencia a otro curso de capacitación, que es JavaScript y HTML5, curso de desarrollo de aplicaciones web, que brinda un nivel de detalle realmente adecuado sobre cómo funciona JavaScript y qué sucede exactamente en los escenarios de objetos JSON. Para este curso de capacitación en particular, solo usamos JSON, pero en realidad no estamos estudiando JavaScript. Así que ese es solo otro idioma para nosotros.

Bueno, veamos cómo se puede manejar el mensaje WebSocket. En este escenario particular, estamos viendo el método OnMessage. Sucede que acepta alguna cadena, devuelve alguna cadena. Es bastante primitivo. Podríamos haber usado estos objetos remotos, un Basic o Asyncremotes, y leer y escribir mensajes con ellos. Pero lo harán. Porque no nos estamos enfocando en la forma en que iniciamos la conversación aquí. Solo nos estamos enfocando en la forma en que analizamos el paquete de datos.

Así que alguien nos envía ese valor. Y queremos responder con lo que sea la cadena, ¿verdad? Creamos un lector JSON. Lector de JSON, ¿es ese objeto o es la transmisión? Es una API de objetos. ESTÁ BIEN. Sólo comprobando que lo recuerdas. Bien. Entonces lector de cadenas, leyendo ese valor, así que básicamente estamos analizando ese objeto JSON. Y estamos terminando con una representación en Java, ese objeto JSON que acabamos de descubrir.

A partir de ese objeto, podríamos empezar a obtener diferentes bits de información. Como un atributo llamado ID, por ejemplo, presumiblemente Product ID. Y luego usamos eso para encontrar un Producto. Si no se encuentra Product, bueno, lanzaremos una excepción. Si se encuentra Producto y todo está bien, creamos un generador de objetos. Y construimos un objeto JSON aquí para que contenga la identificación, el nombre y el precio.

ESTÁ BIEN. Vayamos una página hacia atrás por un segundo. Esto es precisamente lo que estás construyendo. Básicamente, es un mensaje de texto que contiene este conjunto de atributos: pares de nombre/valor. Y eso es lo que estarás transmitiendo a través de la red. Y de la red, recibirá el mensaje que presumiblemente solo contenía la identificación, la identificación del producto. Porque mira el código aquí, eso es lo que obtienes de ese paquete. Obtiene solo ese ID de producto entero. Y en respuesta, está construyendo un Producto que tiene ID, nombre y precio.

Observe también este tipo de notación de invocación de método encadenado. Así que tienes un punto para agregar, un punto para agregar aquí. El objeto se construye añadiéndole más componentes. Bueno, una vez que se construye el objeto, está listo para escribirlo en la transmisión. Entonces puede usar JsonWriter y escribirlo en el flujo de salida. Ahí lo tienes, y devuelve el valor. Si usa una API transmitida, supongo que puede usar una transmisión directamente aquí. Pero en este caso particular, estamos usando, estamos construyendo el objeto primero.

Hablando de manejar errores con sockets, bueno, eso se hace en la operación OnError. La operación OnError debe aceptar un objeto Throwable como argumento que indica qué error acaba de ocurrir. Puede aceptar sesión. Puede aceptar parámetros adicionales, si lo desea. Y bueno, en este caso particular, la lógica es así. Si tenemos un problema, hay un error.

Luego tomamos un objeto remoto y notificamos a la otra parte sobre lo que acaba de suceder, ya sabes, enviamos un mensaje, cualquier otra cosa que quieras hacer, no sé, escribir archivos de registro, así que cualquier otro manejo que quieras para aplicar al error en ese punto. Se puede hacer en un servidor. Se puede hacer en un cliente. Este código se reflejaría completamente en el otro nivel de todos modos.

Ahora, puede decidir que desea proporcionar clases separadas de codificador y decodificador. No codifique el algoritmo de codificación o decodificación de sus mensajes en el controlador, pero hágalo en una clase separada. Primero, debe registrar estos codificadores y decodificadores. Entonces registramos eso con su punto final. Podría ser servidor. Podría ser el punto final del cliente. No importa, sí, la misma idea. Entonces, obviamente, necesitas escribir estas dos clases.

Ahora, aquí está la cosa. ¿Ve que este método particular de OnMessage utiliza el objeto Producto como valor de retorno y el objeto Producto como parámetro? Sin codificador y decodificador, no tienes ninguna posibilidad de que funcione. Porque, ¿cómo convertiría un objeto Java arbitrario, solo su objeto personalizado, a JSON? Tienes que tener un poco de código que haga la conversión.

El método OnMessage simplemente fallará. Dirá, No tengo idea de qué hacer con el Producto. En ausencia de codificador o decodificador, no podrá comunicarse con el otro nivel, porque no sabría cómo transmitir este objeto Java al otro controlador. Ahí vas.

Bueno, echemos un vistazo a cómo se escriben los codificadores y decodificadores. Aquí está el ejemplo de codificador. Podría ser binario o de texto, bueno, dependiendo del formato que desee enviar. En este caso particular, estamos convirtiendo Producto a JSON. Lo que significa el codificador toma un objeto Java y conviértelo al formato que deseas enviar a través de la red.

Método de codificación, eso es lo que necesitamos anular, aceptar el objeto Java como argumento, devolver cualquier tipo que desee enviar a través de la red. ¿Y si es texto? Supongo que una cadena aquí, pero podría ser binario. Entonces podría ser de tipo binario o de texto. De hecho, es texto y es JSON, por lo que estamos creando aquí. Este código es exactamente igual que un ejemplo anterior. Creamos un nuevo objeto JSON que representa Producto con ID nombre y precio. Estamos construyendo eso y lo estamos escribiendo.

También los codificadores y decodificadores tienen algunos métodos de ciclo de vida, como estos dos: iniciar y destruir. Bueno, si necesita inicializar recursos adicionales, los limpiaré cuando el codificador o decodificador se inicialice o se destruya. Y el siguiente ejemplo nos muestra el decodificador. Así que es un espejo del ejemplo anterior.

En este caso, estamos convirtiendo el mensaje de cadena en Producto. De nuevo, es un codificador de texto, lo siento, decodificador, por supuesto, decodificador de texto. Y estamos usando el objeto JSON aquí para analizar el mensaje entrante aquí, extraer bits de él y formar un nuevo objeto Java a partir de estos bits, que es lo que devolvemos con este método.

Además de los métodos habituales del ciclo de vida, como init y destroy, aquí hay un método más que quizás desee utilizar y se llama willDecode. La idea del mensaje o método willDecode es verificar si el mensaje entrante es de su agrado. Tal vez no te guste. Tal vez hay algo mal con eso. Devuelve falso de este método y no continúa con un intento de decodificarlo. Eso le permite eliminar todo tipo de problemas relacionados con mensajes corruptos o erróneos que podría estar recibiendo. Por lo tanto, willDecode si devuelve falso básicamente evita que se invoque la decodificación. Si devuelve verdadero, obviamente, el método de decodificación progresa como de costumbre.

Ahora, hemos mencionado la otra API, la capacidad de transmisión JSON. Entonces, cuando está haciendo la codificación, esta es su elección entre el objeto y el tipo de enfoque de transmisión aquí. ESTÁ BIEN. Entonces, el primero que creamos en JsonGenerator y lo apuntamos directamente al paradero de flujo de destino que queremos generar. Y luego comenzaremos a escribir en la secuencia.

Mira, la cuestión es que no estamos creando un objeto intermedio en memoria para conservar ese conjunto de valores. Seguimos escribiéndolos a medida que surgen. Y si obtienes un objeto realmente grande para formar, eso ahorrará algo de memoria.

Bueno, el otro enfoque es crear ese objeto intermedio con el generador de objetos JSON. Eso significa que mantenemos todos los valores al mismo tiempo en la RAM, y luego escribimos ese objeto donde queremos que escriba, por lo que podría ser JsonWriter apuntando y nuevamente a algún flujo. Así que realmente, esa es la diferencia. ¿Quiere tener un objeto intermedio para mantener sus valores en la memoria? ¿O eres feliz simplemente escribiendo cosas directamente en la secuencia? Eso depende completamente de ti.

Y al revés, esta es la opción de analizar los mensajes entrantes. Nuevamente, básicamente puede usar un analizador que funcione con una secuencia. Sin embargo, eso es un poco complicado, como puedes ver. Entonces, sí, está conectando el analizador al flujo entrante. Pero luego tienes que manejar las cosas que surgen en esa corriente a medida que surgen.

Así que miras el flujo de contenido que está llegando y estás diciendo, ¿es una clave JSON, como recordar, ID, nombre, precio aquí? ¿El siguiente es un valor JSON? Y tienes que generar el... el analizador generará los eventos para cada elemento que ocurre en una secuencia y lo que tienes que hacer es reaccionar a estos eventos generados por el analizador y obtener los valores relevantes.

Sacas estos valores, ¿dónde los pones? Depende completamente de ti. Si comienza a poner todos estos valores en la memoria, también podría crear un objeto JSON en primer lugar. Pero la idea es que el analizador de secuencias, JsonParser, le permite no necesariamente poner todos estos valores en la memoria de una sola vez, y manejarlos a medida que emergen de la secuencia.

O si está feliz de analizar el objeto y el flujo de entrada, y convertirlo en el objeto de una sola vez, simplemente use el ejemplo de JsonReader, en lugar de JsonParser. Y eso solo crea un objeto en línea recta. Es mucho más conveniente, por supuesto. Y si tienes un objeto que no es demasiado grande, supongo que el lector es mucho más fácil de usar que el analizador. Entonces, para objetos grandes, es posible que tenga que ir a un nivel de codificación un poco más bajo aquí, en un nivel más elemental, procesar el mensaje entrante.

Bueno, ¿qué tal llamar a nuestro servidor WebSocket desde el cliente? Habrá un ejemplo de cliente JavaScript y habrá un ejemplo de cliente Java. Comencemos con JavaScript. Nuevamente, para una conversación más detallada sobre cómo funciona JavaScript con respecto a los sockets, se trata en esta clase de desarrollo de aplicaciones web de JavaScript y HTML5.

Pero en este curso, todavía necesitamos tener una comprensión al menos vaga de lo que está pasando, porque en realidad lo hará en su ejercicio. Entonces, lo que debemos hacer es crear un nuevo objeto WebSocket. Así que ahí está. Ese es este nuevo objeto WebSocket, apunta a la URL de WebSocket relevante. Y luego registre las funciones onmessage, onerror, onopen, onclose como mejor le parezca.

Supongo que onmessage y onerror son los must, porque si no, ¿cómo vas a tener una conversación? En cuanto a abrir y cerrar, tal vez, pero eso es algo opcional aquí si necesita estos controladores. De todos modos, comenzará a comunicarse. Puede analizar los objetos JSON que está enviando a través de la red. Puede crear nuevos objetos JSON, transmitir diferentes valores. Ese es el código de comunicación real. Es bastante sencillo.

Ahora este es el cliente de Java. Es muy similar al punto final del servidor. Pero la diferencia real es que está utilizando la anotación del punto final del cliente en lugar de la anotación del punto final del servidor, ¿verdad? Así que este es tu controlador. Usted define OnOpen, OnClose, OnError, OnMessage, los métodos que necesite.

Luego, debe crear un objeto contenedor WebSocket. Con ese objeto contenedor WebSocket, tome el URI que apunta a ese socket. Y use el contenedor para establecer una conexión con ese URI. El otro parámetro del método connectToServe es una referencia a su controlador de punto final de cliente, a este. Básicamente, esa es la clase que manejará los eventos onopen, onclose, onerror, onmessage, etcétera.

Eso es todo. Estás listo para manejar la comunicación. Si desea enviar algo al servidor, obtenga Basic o AsynchRemote y comience a enviar. En este caso particular, estamos enviando Producto, por lo que presumiblemente, tenemos codificadores y decodificadores para eso, obviamente aquí. Cuando termine de comunicarse con un servidor, cierre la sesión. Puede proporcionar una razón de cierre aquí. Ahí tienes ESTÁ BIEN.

Eso es todo por el capítulo WebSockets. Entonces, lo que hemos aprendido hasta ahora es el estilo de comunicación WebSocket, que es bastante diferente de HTTP porque nos permite realizar comunicaciones de estilo push, en lugar de solo pull. Hemos analizado las formas en que podemos escribir controladores de punto final y el ciclo de vida del WebSocket, tanto para el cliente como para los controladores del servidor. Cómo producir y consumir mensajes de socket, cómo manejar errores y todo tipo de variaciones de cómo puede codificar y decodificar mensajes JSON.

Ahora, si puedo llamar su atención sobre un par de cosas con respecto a WebSockets. Desafortunadamente, la API de WebSocket no convierte automáticamente los mensajes JSON o XML a Java y viceversa. Tienes que escribir codificadores y decodificadores.

¿Por qué menciono eso? Porque en un capítulo anterior, cuando usó la API REST, eso funcionó para usted. Estaba enviando objetos a través de la API REST sin tener que escribir sus convertidores de JSON a Java y viceversa. Simplemente funcionó milagrosamente porque en la API REST, esa es una pieza ya escrita de funcionalidad estándar. Pero con WebSockets, usted es responsable de proporcionar codificadores y decodificadores. eso es una cosa

La otra cosa, una vez más, quiero reiterar la aplicabilidad de WebSocket. Está creando una gran tensión en su servidor, porque está asignando una conexión con estado, una conexión dúplex completa, una sesión con estado con cada cliente. Entonces eso obviamente es una limitación de escalabilidad. No use WebSockets, porque introducen estado, si puede evitar el uso de estado. Si su aplicación lo necesita, está bien. Entonces, por supuesto, optaría por la conectividad con estado. Pero la conectividad sin estado, como HTTP, definitivamente escala mejor.

Hablando de esto, cuando está viendo el ejercicio, un ejemplo que se le pide que haga y el ejercicio crea un servidor de chat WebSocket. Suponga que desea que sus usuarios implementen algún tipo de algoritmo de chat e intercambien mensajes. Esa naturaleza de un WebSocket de aprovisionar esa comunicación instantánea, de transmitir instantáneamente el mensaje a través del servidor a todas las sesiones, es bastante importante. Esa sería una buena justificación de por qué usaría un WebSocket.

Otras buenas justificaciones podrían ser, por ejemplo, los juegos de computadora, donde tienes juegos para múltiples jugadores. Necesitan intercambiar información sobre eventos que ocurren dentro del juego instantáneamente. Entonces, nuevamente, esa es una consideración bastante importante que un WebSocket le permitiría satisfacer. Pero de lo contrario, si esa no es una alta prioridad para una comunicación tan instantánea, entonces tal vez el uso de una buena extracción del servidor a la antigua escalará mejor.

Entonces, si se trata de alguna funcionalidad de una aplicación comercial, como manejar estos productos aquí, es posible que realmente no necesite un WebSocket para eso, porque no hay una demanda tan alta de comunicación instantánea en este escenario. Y luego, por supuesto, en el ejercicio, está creando clientes JavaScript y Java para interactuar con un servidor o un servidor de chat.