Middleware de servidor da Web para Dart

Shelf torna fácil criar e compor servidores web e partes de servidores web . Como?

- Exponha um pequeno conjunto de tipos simples.
- Mapeie a lógica do servidor em uma função simples: um único argumento para a solicitação, a resposta é o valor de retorno.
- Misture e combine trivialmente o processamento síncrono e assíncrono.
- Flexibilidade para retornar uma string simples ou um fluxo de bytes com o mesmo modelo.

Consulte a documentação do servidor Dart HTTP para obter mais informações. Você também pode querer ver package:shelf_router e package:shelf_static como exemplos de pacotes que se baseiam e estendem `package:shelf`.

Manipuladores e Middleware

Um Handler é qualquer função que manipula um Request e retorna uma Response . Ele pode lidar com a solicitação em si - por exemplo, um servidor de arquivos estático que procura o URI solicitado no sistema de arquivos - ou pode fazer algum processamento e encaminhá-lo para outro manipulador - por exemplo, um logger que imprime informações sobre solicitações e respostas para a linha de comando.

O último tipo de manipulador é chamado de " middleware ", pois fica no meio da pilha do servidor. O middleware pode ser pensado como uma função que pega um manipulador e o envolve em outro manipulador para fornecer funcionalidade adicional. Um aplicativo Shelf geralmente é composto de várias camadas de middleware com um ou mais manipuladores bem no centro; a classe Pipeline facilita a construção desse tipo de aplicativo.

Alguns middlewares também podem usar vários manipuladores e chamar um ou mais deles para cada solicitação. Por exemplo, um middleware de roteamento pode escolher qual manipulador chamar com base no URI ou no método HTTP da solicitação, enquanto um middleware em cascata pode chamar cada um em sequência até que um retorne uma resposta bem-sucedida.

O middleware que roteia solicitações entre manipuladores deve certificar-se de atualizar cada solicitação `handlerPath`e `url`. Isso permite que os manipuladores internos saibam onde estão no aplicativo para que possam fazer seu próprio

roteamento corretamente. Isso pode ser feito facilmente usando `Request.change()`:

Adaptadores

Um adaptador é qualquer código que cria objetos Request , passa-os para um manipulador e lida com o Response resultante . Na maioria das vezes, os adaptadores encaminham solicitações e respostas para um servidor HTTP subjacente; shelf_io.serve é esse tipo de adaptador. Um adaptador também pode sintetizar solicitações HTTP no navegador usando `window.location`e `window.history`, ou pode canalizar solicitações diretamente de um cliente HTTP para um manipulador Shelf.

Requisitos da API

Um adaptador deve manipular todos os erros do manipulador, incluindo o manipulador que retorna uma `null`resposta. Ele deve imprimir cada erro no console, se possível, e agir como se o manipulador retornasse uma resposta 500. O adaptador pode incluir dados do corpo para a resposta 500, mas esses dados do corpo não devem incluir informações sobre o erro ocorrido. Isso garante que erros inesperados não resultem na exposição de informações internas em produção por padrão; se o usuário quiser retornar descrições de erro detalhadas, ele deve incluir explicitamente um middleware para fazer isso.

Um adaptador deve garantir que os erros assíncronos lançados pelo manipulador não causem a falha do aplicativo, mesmo que não sejam relatados pela cadeia futura. Especificamente, esses erros não devem ser passados para o manipulador de erros da zona raiz; no entanto, se o adaptador for executado em outra zona de erro, ele deverá permitir que esses erros sejam transmitidos para essa zona. A função a seguir pode ser usada para capturar apenas erros que, de outra forma, seriam de nível superior:

Um adaptador que conhece sua própria URL deve fornecer uma implementação da `Server`interface.

Solicitar Requisitos

Ao implementar um adaptador, algumas regras devem ser seguidas. O adaptador não deve passar os parâmetros `url`ou `handlerPath`para Request ; deve apenas passar `requestedUri`. Se passar o `context`parâmetro, todas as chaves devem começar com o nome do pacote do adaptador seguido por um ponto. Se vários cabeçalhos com o mesmo nome forem recebidos, o adaptador deverá reduzi-

los em um único cabeçalho separado por vírgulas conforme RFC 2616 seção 4.2 .
Se a solicitação subjacente usar uma codificação de transferência em partes, o adaptador deverá decodificar o corpo antes de passá-lo para Request e remover o `Transfer-Encoding`cabeçalho. Isso garante que os corpos das mensagens sejam agrupados se e somente se os cabeçalhos declararem que são.

Requisitos de resposta

Um adaptador não deve adicionar ou modificar nenhum cabeçalho de entidade para uma resposta.

Se *nenhuma* das condições a seguir for verdadeira, o adaptador deverá aplicar a codificação de transferência em partes ao corpo de uma resposta e definir seu cabeçalho Transfer-Encoding como `chunked`:

- O código de status é menor que 200 ou igual a 204 ou 304.
- Um cabeçalho Content-Length é fornecido.
- O cabeçalho Content-Type indica o tipo MIME `multipart/byteranges`.
- O cabeçalho Transfer-Encoding é definido como qualquer coisa diferente de `identity`.

Os adaptadores podem achar o `addChunkedEncoding()`middleware útil para implementar esse comportamento, se o servidor subjacente não o implementar manualmente.

Ao responder a uma solicitação HEAD, o adaptador não deve emitir um corpo de entidade. Caso contrário, não deve modificar o corpo da entidade de forma alguma.

Um adaptador deve incluir informações sobre si mesmo no cabeçalho Server da resposta por padrão. Se o manipulador retornar uma resposta com o conjunto de cabeçalho do Servidor, isso deverá ter precedência sobre o cabeçalho padrão do adaptador.

Um adaptador deve incluir o cabeçalho Date com a hora em que o manipulador retorna uma resposta. Se o manipulador retornar uma resposta com o conjunto de cabeçalho Date, isso deverá ter precedência.