**OpenID Connect: um padrão para autenticação e identidade sobre OAuth 2.0**

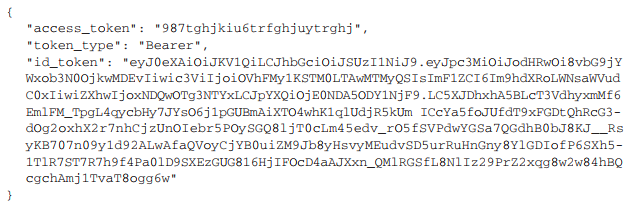
O OpenID Connect é um padrão aberto publicado pela OpenID Foundation em fevereiro de 2014 que define uma maneira interoperável de usar o OAuth 2.0 para realizar a autenticação de usuários. Em essência, é uma "receita de chocolate fudge" amplamente publicada, que foi construída e testada por uma ampla variedade de implementadores. Como um padrão aberto, o OpenID Connect pode ser implementado sem preocupações com licença ou propriedade intelectual. Como o protocolo foi projetado para ser interoperável ("Interoperável" significa que algo é capaz de funcionar ou operar em conjunto com outros sistemas, produtos ou equipamentos, mesmo que de fabricantes diferentes), um aplicativo cliente OpenID pode comunicar um protocolo para vários provedores de identidade, em vez de implementar um protocolo ligeiramente diferente para cada provedor de identidade. O OpenID Connect é construído diretamente no OAuth 2.0 e permanece compatível com ele. Em muitos casos, ele é implantado junto com uma infraestrutura OAuth simples que protege outras APIs. Além do OAuth 2.0, o OpenID Connect usa o conjunto de especificações JSON Object Signing and Encryption (JOSE) (que abordamos no capítulo 11) para transportar informações assinadas e criptografadas em diferentes locais. Uma implantação do OAuth 2.0 com recursos JOSE já está bem avançada no caminho para se tornar um sistema OpenID Connect totalmente compatível, visto que a diferença entre os dois é relativamente pequena. Mas essa diferença faz uma grande diferença, e o OpenID Connect consegue evitar muitas das armadilhas discutidas anteriormente, adicionando vários componentes-chave à base do OAuth 2.0.

**ID tokens**

O token de ID do OpenID Connect é um JSON Web Token (JWT) assinado fornecido ao aplicativo cliente junto com o token de acesso OAuth regular. Ao contrário do token de acesso, o token de ID é direcionado ao RP e deve ser analisado por ele. Assim como os tokens de acesso assinados que criamos no capítulo 11, o token de ID contém um conjunto de declarações sobre a sessão de autenticação, incluindo um identificador para o usuário (sub), o identificador para o provedor de identidade que emitiu o token (iss) e o identificador do cliente para o qual este token foi criado (aud). Além disso, o token de ID contém informações sobre a janela de validade do próprio token (com as declarações exp e iat), bem como quaisquer informações adicionais sobre o contexto de autenticação a serem transmitidas ao cliente. Por exemplo, o token pode informar há quanto tempo o usuário foi apresentado a um mecanismo de autenticação primário (auth\_time) ou que tipo de autenticação primária ele utilizou no IdP (acr). O token de ID também pode conter outras reivindicações dentro dele, tanto reivindicações JWT padrão, como as listadas no capítulo 11, quanto reivindicações estendidas para o protocolo OpenID Connect. As reivindicações necessárias estão em negrito na tabela 13.1.

|  |  |
| --- | --- |
| **Claim Name** | **Claim Description** |
| **iss** | O emissor do token; URL do IdP |
| **sub** | O assunto do token, um identificador estável e único para o usuário no IdP. Geralmente, é uma string legível por máquina e não deve ser usada como nome de usuário. |
| **aud** | O público do token; deve conter o ID do cliente do RP |
| **exp** | O carimbo de data/hora de expiração do token. Todos os tokens de ID expiram, e geralmente bem rápido. |
| **iat** | O registro de data e hora de quando o token foi emitido. |
| auth\_time | O registro de data e hora em que o usuário foi autenticado no IdP |
| nonce | Uma string enviada pelo RP durante a solicitação de autenticação, usada para mitigar ataques de repetição semelhantes ao parâmetro de estado. Deve ser incluída se o RP a enviar. |
| acr | A referência de contexto de autenticação, que indica uma categorização geral da autenticação que o usuário realizou no IdP |
| amr | A referência do método de autenticação, que indica como o usuário se autenticou no  IdP. |
| azp | A parte autorizada para este token; deve conter o ID do cliente do RP, se estiver incluído. |
| at\_hash | Hash criptográfico do token de acesso. |
| c\_hash | Hash criptográfico do código de autorização. |

O token ID é emitido em adição a um token de acesso como o membro id\_token da resposta do endpoint do token, e não em seu lugar. Isso se deve ao fato de que os dois tokens têm diferentes públicos-alvo e usos pretendidos. A abordagem de dois tokens permite que o token de acesso permaneça opaco para o cliente, como no OAuth comum, enquanto permite que o token ID seja analisado. Além disso, os dois tokens também podem ter ciclos de vida diferentes, com o token ID frequentemente expirando mais rapidamente. Embora o token ID represente um único evento de autenticação e nunca seja passado para um serviço externo, o token de acesso pode ser usado para buscar recursos protegidos muito depois que o usuário tiver saído. Embora seja verdade que você ainda possa usar o token de acesso para perguntar quem autorizou o cliente em primeiro lugar, isso não informará nada sobre a presença do usuário, como você viu anteriormente.



Por fim, o próprio token ID é assinado pela chave do provedor de identidade, adicionando outra camada de proteção às declarações contidas nele, além da proteção de transporte TLS que foi usada para obter o token inicialmente. Como o token ID é assinado pelo servidor de autorização, ele também fornece um local para adicionar assinaturas separadas sobre o código de autorização (c\_hash) e o token de acesso (at\_hash). O cliente pode validar esses hashes enquanto mantém o código de autorização e o conteúdo do token de acesso opacos para o cliente, prevenindo toda uma classe de ataques de injeção. Aplicando algumas verificações simples a esse token ID, as mesmas verificações usadas ao processar um JWT assinado, como fizemos no capítulo 11, um cliente pode se proteger de um grande número de ataques comuns:

1. Analise o token de ID para garantir que seja um JWT válido e colete as declarações.

* Divida a string no caractere "."
* Decodifique cada seção em Base64URL.
* Analise as duas primeiras seções (cabeçalho e payload) como JSON.

1. Valide a assinatura do token em relação à chave pública do IdP, publicada em um local detectável.
2. Verifique se o token ID foi emitido por um IdP confiável.
3. Certifique-se de que o identificador de cliente do cliente esteja incluído na lista de público do token ID.
4. Confirme se os valores de carimbo de data/hora de expiração, emitido em e não antes são razoáveis ​​considerando a hora atual.
5. Certifique-se de que o nonce, se presente, corresponda ao enviado.
6. Valide os hashes do código de autorização ou token de acesso, se aplicável.

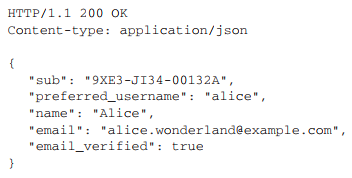
Cada uma dessas etapas é determinística e mecânica, exigindo esforço mínimo de codificação. Alguns modos mais avançados do OpenID Connect permitem que o token de identificação seja criptografado também, o que altera ligeiramente o processo de análise e verificação, mas com os mesmos resultados.

**The UserInfo endpoint**

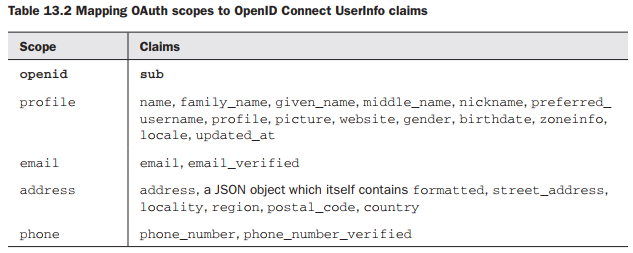
Como o token ID contém todas as informações necessárias para processar o evento de autenticação, os clientes do OpenID Connect não precisam de nada além disso para processar um login bem-sucedido. No entanto, o token de acesso pode ser usado em um recurso protegido padrão que contém informações de perfil sobre o usuário atual, chamado de endpoint UserInfo. As declarações neste endpoint não fazem parte do processo de autenticação discutido anteriormente, mas fornecem atributos de identidade agrupados que tornam o protocolo de autenticação mais valioso para desenvolvedores de aplicativos. Afinal, é preferível dizer "Bom dia, Alice" em vez de "Bom dia, 9XE3-JI34-00132A". A solicitação ao endpoint UserInfo é um simples HTTP GET ou POST, com o token de acesso (não o token ID) enviado como autorização. Não há parâmetros de entrada em uma solicitação normal, embora, como em grande parte do OpenID Connect, haja alguns métodos avançados que podem ser usados ​​aqui. O endpoint UserInfo segue um design de recurso protegido, com o mesmo recurso para todos os usuários do sistema, em oposição a um URI de recurso diferente para cada usuário. O IdP descobre qual usuário está sendo consultado desreferenciando o token de acesso.



A resposta do endpoint UserInfo é um objeto JSON que contém declarações sobre o usuário. Essas declarações tendem a ser estáveis ​​ao longo do tempo, e é comum armazenar em cache os resultados da chamada do endpoint UserInfo em vez de buscá-los em cada solicitação de autenticação. Usando os recursos avançados do OpenID Connect, também é possível retornar a resposta UserInfo como um JWT assinado ou criptografado.

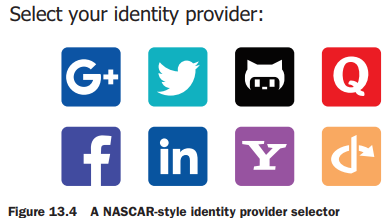


O OpenID Connect usa o valor especial do escopo openid para restringir o acesso ao endpoint UserInfo. O OpenID Connect define um conjunto de escopos OAuth padronizados que mapeiam para subconjuntos desses atributos do usuário (perfil, e-mail, telefone e endereço, mostrados na tabela 13.2), permitindo que transações OAuth simples solicitem tudo o que for necessário para uma autenticação. A especificação do OpenID Connect detalha cada um desses escopos e os atributos aos quais eles mapeiam. O OpenID Connect define um escopo openid especial que controla o acesso geral ao endpoint UserInfo pelo token de acesso. Os escopos do OpenID Connect podem ser usados junto com outros escopos OAuth 2.0 que não sejam do OpenID Connect sem conflito, e o token de acesso emitido pode potencialmente ser direcionado a diversos recursos protegidos, além do endpoint UserInfo. Essa abordagem permite que um sistema de identidade OpenID Connect coexista sem problemas com um sistema de autorização OAuth 2.0.



**Descoberta dinâmica de servidor e registro de cliente**

O OAuth 2.0 foi escrito para permitir uma variedade de implantações diferentes, mas, por design, não especifica como essas implantações são configuradas ou como os componentes se conhecem uns aos outros. Isso é aceitável no mundo OAuth tradicional, no qual um servidor de autorização protege uma API específica e os dois geralmente estão intimamente acoplados. O OpenID Connect define uma API comum que pode ser implantada em uma ampla variedade de clientes e provedores. Não seria escalável que cada cliente tivesse que saber com antecedência sobre cada provedor, nem seria razoável exigir que cada provedor soubesse sobre todos os clientes em potencial. Para neutralizar isso, o OpenID Connect define um protocolo de descoberta5 que permite aos clientes buscar facilmente informações sobre como interagir com um provedor de identidade específico. Esse processo de descoberta ocorre em duas etapas. Primeiro, o cliente precisa descobrir a URL do emissor do IdP. Isso pode ser configurado diretamente, como em um seletor de provedores comum no estilo NASCAR na figura 13.4. Alternativamente, o emissor pode ser descoberto com base no protocolo WebFinger. O WebFinger funciona utilizando um meio comum de identificação de usuários, endereços de e-mail, e fornece um conjunto de regras de transformação determinísticas que utilizam essa entrada amigável e direta para o usuário e geram um URI de descoberta (figura 13.5). Basicamente, você pega a parte do domínio do identificador de endereço de e-mail, anexa https:// ao início e anexa /.wellknow/webfinger ao final para criar um URI. Opcionalmente, você também pode passar informações sobre o que o usuário digitou originalmente, bem como o tipo de informação que você está procurando. No OpenID Connect, esse URI de descoberta pode ser obtido via HTTPS para determinar o emissor do endereço de um usuário específico.





Após a determinação do emissor, o cliente ainda precisa de informações essenciais sobre o servidor, como a localização da autorização e os pontos de extremidade do token. Isso é descoberto anexando /.well-known/openid-configuration ao URI do emissor descoberto na primeira etapa e buscando a URL resultante. Isso retorna um documento JSON contendo todos os atributos do servidor que o cliente precisa para iniciar a transação de autenticação. A seguir, um exemplo adaptado de um servidor de teste disponível publicamente:



Uma vez que o cliente tenha conhecimento do servidor, o servidor precisa ter conhecimento do cliente. Para isso, o OpenID Connect define um protocolo de registro de cliente6 que permite que os clientes sejam apresentados a novos provedores de identidade. A extensão do protocolo OAuth Dynamic Client Registration, discutida no capítulo 12, foi desenvolvida em paralelo à versão do OpenID Connect, e as duas são compatíveis entre si na rede. Aproveitando a descoberta, o registro, uma API de identidade comum e a escolha do usuário final, o OpenID Connect pode funcionar em escala de internet. Mesmo quando nenhuma das partes precisa se conhecer com antecedência, duas instâncias compatíveis do OpenID Connect podem interagir entre si para efetuar um protocolo de autorização além dos limites de segurança.

**Compatibilidade com OAuth 2.0**

Mesmo com toda essa robusta capacidade de autenticação, o OpenID Connect ainda é, por design, compatível com o OAuth 2.0 simples. De fato, se um serviço já utiliza o OAuth 2.0 e as especificações JOSE, incluindo JWT, esse serviço já está a caminho de oferecer suporte total ao OpenID Connect. Para facilitar a construção de bons aplicativos cliente, o grupo de trabalho do OpenID Connect publicou documentos sobre a construção de um cliente OpenID Connect básico7 usando o fluxo de código de autorização, bem como a construção de um cliente OpenID Connect implícito8. Ambos os documentos orientam o desenvolvedor na construção de um cliente OAuth 2.0 básico e na adição de alguns componentes necessários para a funcionalidade do OpenID Connect, muitos dos quais foram descritos aqui.

**Capacidades avançadas**

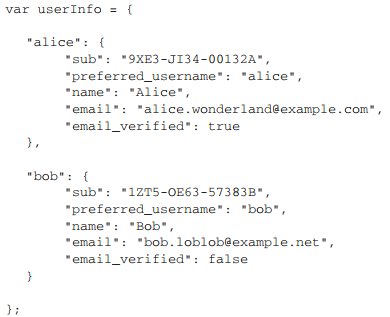
Embora o núcleo da especificação OpenID Connect seja bastante simples, nem todos os casos de uso podem ser adequadamente abordados pelos mecanismos básicos. Para oferecer suporte a muitos casos de uso avançados, o OpenID Connect também define uma série de recursos avançados opcionais além do OAuth padrão. Abordar tudo isso em profundidade poderia facilmente preencher outro livro, mas podemos pelo menos abordar alguns componentes-chave nesta seção. Um cliente OpenID Connect pode, opcionalmente, autenticar usando um JWT assinado em vez do segredo de cliente compartilhado mais tradicional do OAuth. Este JWT pode ser assinado com a chave assimétrica de um cliente, se ele registrar sua chave pública no servidor, ou pode ser assinado simetricamente com o segredo do cliente. Este método fornece um nível mais alto de segurança para os clientes, que evitam o envio de suas senhas pela rede. Da mesma forma, um cliente OpenID Connect pode, opcionalmente, enviar suas solicitações ao endpoint de autorização como um JWT assinado em vez de um conjunto de parâmetros de formulário. Desde que a chave usada para assinar este objeto de solicitação esteja registrada no servidor, o servidor pode validar os parâmetros dentro do objeto de solicitação e ter certeza de que o navegador não os adulterou. Um servidor OpenID Connect pode, opcionalmente, assinar ou criptografar a saída do servidor, incluindo o endpoint UserInfo, como um JWT. O token de ID também pode ser criptografado, além de ser assinado pelo servidor. Essas proteções podem garantir ao cliente que a saída não foi adulterada, além das garantias obtidas com o uso de TLS na conexão. Outros parâmetros foram adicionados como extensões aos endpoints do OAuth 2.0, incluindo dicas para tipos de exibição, comportamento de solicitação e referências de contexto de autenticação. Usando a construção do objeto de solicitação, um cliente OpenID Connect pode fazer solicitações muito mais ajustadas ao servidor de autorização do que suas contrapartes do OAuth 2.0, graças à expressividade inerente do payload JSON do objeto de solicitação. Essas solicitações podem incluir informações refinadas de declarações de usuário, como solicitar que apenas um usuário correspondente a um identificador específico esteja conectado. O OpenID Connect fornece uma maneira para o servidor (ou outro terceiro) iniciar o processo de login. Embora todas as transações canônicas do OAuth 2.0 sejam iniciadas pelo aplicativo cliente, esse recurso opcional oferece ao cliente uma maneira de receber sinais para iniciar o processo de login com um IdP específico. O OpenID Connect também define algumas maneiras diferentes de recuperar tokens, incluindo fluxos híbridos nos quais algumas informações (como o token de ID) são transmitidas no canal frontal e outras informações (como o token de acesso) são transmitidas no canal traseiro. Esses fluxos não devem ser considerados como simples combinações de fluxos OAuth 2.0 existentes, mas sim como novas funcionalidades para diferentes aplicativos. Por fim, o OpenID Connect fornece uma especificação para gerenciar sessões entre o RP e o IdP, ou mesmo entre vários RPs. Como o OAuth 2.0 não considera a presença do usuário além do momento da delegação de autorização, extensões são necessárias para lidar com o ciclo de vida de uma autenticação federada. Se o usuário efetuar logout de uma RP, ele poderá querer efetuar logout de outras também, e a RP precisa ser capaz de sinalizar ao IdP que isso deve acontecer. Outras RPs precisam ser capazes de ouvir um sinal do IdP de que uma saída ocorreu e reagir de acordo. O OpenID Connect oferece todas essas extensões sem comprometer a compatibilidade com o OAuth 2.0.

**Construindo um sistema OpenID Connect simples**

Abra o capítulo 13 do exemplo 1 para encontrar um sistema OAuth 2.0 totalmente funcional. Agora, vamos construir um sistema OpenID Connect simples sobre nossa infraestrutura OAuth 2.0 existente. Embora um livro inteiro pudesse ser dedicado à implementação de todos os recursos do OpenID Connect, abordaremos o básico aqui neste exercício. Adicionaremos suporte para a emissão do token de ID ao fluxo de código de autorização em nosso servidor de autorização. Também criaremos um endpoint UserInfo em nosso recurso protegido com um banco de dados compartilhado, pois este é um padrão de implantação comum. Observe que, embora nosso servidor de autorização e o endpoint UserInfo estejam sendo executados em processos separados, da perspectiva da RP, eles estão funcionando como um único IdP. Também transformaremos nosso cliente OAuth 2.0 genérico em uma RP OpenID Connect, analisando e validando o token de ID e buscando o UserInfo para exibição. Em todos esses exercícios, deixamos de fora um componente-chave: a autenticação do usuário. Em vez disso, estamos usando novamente uma simples seleção suspensa na página de autorização para determinar qual usuário está "logado" no IdP, como fizemos no capítulo 3. Em um sistema de produção, o mecanismo de autenticação primário usado no IdP é de extrema importância, pois a identidade federada emitida pelo servidor depende disso. Existem muitas boas bibliotecas de autenticação primária, e incorporá-las à nossa estrutura fica como um exercício para o leitor. Mas, ainda assim, caso precise ser dito: por favor, não use uma simples caixa suspensa como mecanismo de autenticação em seu sistema de produção.

Generating the ID token

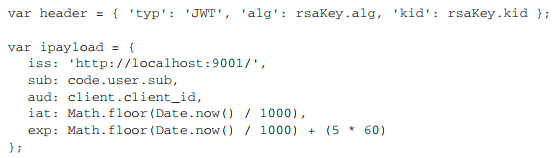
Primeiro, precisamos gerar um token de ID e distribuí-lo junto com nosso token de acesso. Usaremos as mesmas bibliotecas e técnicas que usamos no capítulo 11, já que um token de ID é na verdade apenas um JWT especial. Se quiser detalhes sobre JWTs, volte ao capítulo 11 para mais informações. Abra authorizationServer.js em um editor. Próximo ao topo do arquivo, fornecemos informações de usuário para dois usuários no sistema, Alice e Bob. Precisaremos disso para criar o token de ID e a resposta UserInfo. Para simplificar, optamos por uma variável simples na memória indexada pelo nome de usuário selecionável no menu suspenso na página de autorização. Em um sistema de produção, isso provavelmente estaria vinculado a um banco de dados, serviço de diretório ou outro armazenamento persistente.



Em seguida, criaremos o token de ID após já termos criado nosso token de acesso. Primeiro, precisamos determinar se devemos ou não criar um token de ID. Queremos gerar um token de ID somente se o usuário autorizou o escopo openid e se tivermos um usuário para falar.



Em seguida, criaremos um cabeçalho para o nosso token de ID e adicionaremos todos os campos necessários para o payload. Primeiro, definimos nosso servidor de autorização como emissor e adicionamos o assunto. Identificador do usuário. Lembre-se de que esses dois campos, juntos, fornecem um identificador global exclusivo para o usuário. Em seguida, definiremos o ID do cliente solicitante para o público do token. Por fim, registraremos o token com um carimbo de data/hora e definiremos uma expiração de cinco minutos no futuro. Geralmente, esse tempo é mais do que suficiente para que um token de ID seja processado e vinculado a uma sessão de usuário em uma RP. Lembre-se de que a RP não precisa usar o token de ID em nenhum recurso externo, portanto, o tempo limite pode e deve ser relativamente curto.



Também adicionaremos o valor nonce, mas somente se o cliente o tiver enviado na solicitação original para o endpoint de autorização. Este valor é análogo ao parâmetro state em muitos aspectos, mas encerra um vetor de ataque entre sites ligeiramente diferente.



Então, assinaremos com a chave do servidor e serializaremos como um JWT.



Por fim, emitiremos o token junto com o token de acesso, modificando a resposta do token existente.

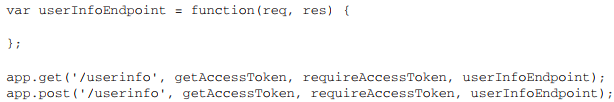


E isso é tudo o que precisamos fazer. Embora pudéssemos armazenar nosso token de ID junto com nossos outros tokens, se quiséssemos, ele nunca é passado de volta para o servidor de autorização ou qualquer recurso protegido; portanto, não há necessidade real de fazer isso. Em vez de agir como um token de acesso, ele atua como uma asserção do servidor de autorização para o cliente. Depois de enviá-lo ao cliente, estamos praticamente prontos.

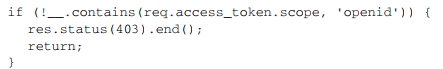
Criando o endpoint UserInfo

Em seguida, adicionaremos o endpoint UserInfo ao nosso recurso protegido. Abra protectedResource.js para esta parte do exercício. Observe que, embora o IdP seja um único componente lógico no protocolo OpenID, é aceitável e válido implementá-lo como servidores separados, como estamos fazendo aqui. Importamos as funções auxiliares getAccessToken e requireAccessToken de exercícios anteriores. Elas usarão o banco de dados local para pesquisar não apenas as informações do token, mas também as informações do usuário associadas a ele. Nosso IdP fornecerá informações do usuário de /userinfo em resposta a solicitações HTTP GET ou POST. Devido a limitações no

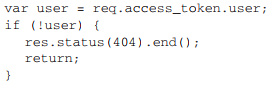
O framework Express.js que estamos usando em nosso código, temos que definir isso de forma um pouco diferente dos exercícios anteriores, usando uma variável de função nomeada externamente para nosso código manipulador, mas o efeito é praticamente o mesmo de antes.



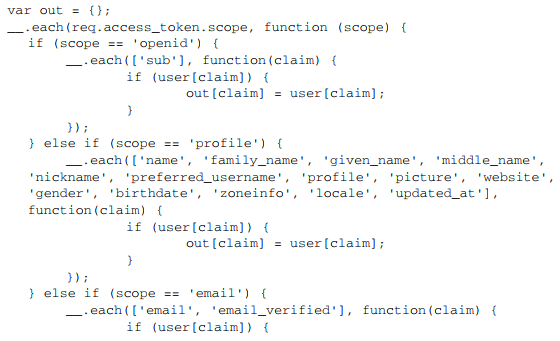
Em seguida, verificaremos se o token recebido contém pelo menos o escopo openid. Caso contrário, retornaremos um erro.

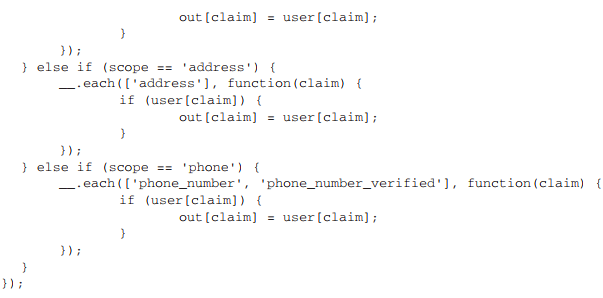


Mais uma vez, precisamos obter o conjunto correto de informações do usuário em nosso repositório de dados. Basearemos isso no usuário que autorizou o token de acesso, de forma semelhante à forma como enviamos as informações em um dos exercícios do Capítulo 4. Se não encontrarmos um usuário, retornaremos um erro.



Em seguida, precisamos construir a resposta. Não podemos retornar o objeto de informações do usuário inteiro, pois o usuário pode ter autorizado apenas um subconjunto dos escopos disponíveis. Como cada escopo mapeia para um subconjunto das informações do usuário, examinaremos cada um dos escopos no token de acesso e adicionaremos as declarações associadas ao nosso objeto de saída à medida que avançamos.





O resultado final é um objeto que contém todas as reivindicações do usuário correto que foram autorizadas por esse usuário para este cliente. Esse processo oferece uma flexibilidade incrível em termos de privacidade, segurança e escolha do usuário. Retornaremos esse objeto como JSON.



A função final se parece com a Listagem 14 do Apêndice B. Com duas pequenas adições, transformamos nosso servidor OAuth 2.0 funcional em um IdP do OpenID Connect também. Conseguimos reutilizar muitos dos componentes que exploramos em capítulos anteriores, como a geração de JWT (capítulo 11), o processamento de tokens de acesso de entrada (capítulo 4) e a varredura de escopos (capítulo 4). Há muitos recursos adicionais no OpenID Connect que mencionamos anteriormente, incluindo objetos de solicitação, descoberta e registro, mas a implementação deles fica como um exercício para o leitor (ou para o leitor de outro livro).

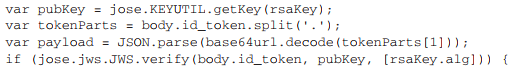
Analisando o token de ID

Agora que o servidor é capaz de gerar tokens de ID, o cliente precisa ser capaz de analisá-los. Usaremos um método semelhante ao usado no Capítulo 11, onde analisamos e validamos um JWT em nosso recurso protegido. Desta vez, o token é direcionado ao cliente, portanto, começaremos em client.js, em um editor. Configuramos estaticamente o cliente e o servidor com as informações um do outro, mas no OpenID Connect tudo isso pode ser feito dinamicamente usando o registro dinâmico de cliente e a descoberta de servidor. Como exercício adicional, extraia o código de registro dinâmico de cliente do Capítulo 12 e implemente a descoberta de servidor sobre essa estrutura. Primeiro, precisamos extrair o valor do token da resposta do token. Como ele é passado para nós na mesma estrutura em que o token de acesso está, o extrairemos desse objeto em nossa função de análise de resposta do token. Também descartaremos quaisquer informações antigas do usuário ou tokens de ID que possamos ter guardados de um login anterior.

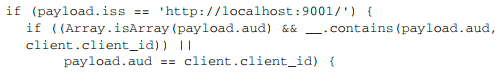




Em seguida, analisaremos a carga útil do token de ID em um objeto JSON e testaremos o conteúdo do token de ID, começando pela sua assinatura. No OpenID Connect, o cliente normalmente busca as chaves do servidor a partir de uma URL definida como JSON Web Key (JWK), mas nós a fornecemos estaticamente no código, juntamente com a configuração do servidor. Para um exercício adicional, configure o servidor para publicar sua chave pública e configure o cliente para buscar a chave do servidor quando necessário em tempo de execução. Nosso servidor usa o método de assinatura RS256 para seus tokens de ID, e estamos usando a biblioteca jsrsasign para manipular nossas funções JOSE, como fizemos no capítulo 11.



Em seguida, precisamos verificar alguns campos para garantir que façam sentido. Mais uma vez, extraímos cada verificação para sua própria instrução if aninhada, aceitando o token somente se todas as verificações forem aprovadas. Primeiro, vamos garantir que o emissor corresponda ao do nosso servidor de autorização e também que o ID do nosso cliente esteja na lista de público.



Então, garantiremos que os carimbos de data/hora de emissão e expiração façam sentido.

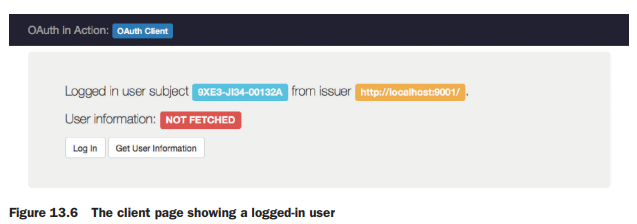


Alguns testes extras usam formas mais avançadas do protocolo, como comparar um valor nonce se tivéssemos enviado um na solicitação original ou calcular os hashes para o token ou código de acesso. Esses testes não são necessários para um cliente simples que usa o tipo de concessão de código de autorização e são deixados como exercícios para o leitor. Se, e somente se, todas essas verificações forem aprovadas, teremos um token de ID válido que podemos salvar em nossa aplicação. Na verdade, não precisamos mais salvar o token inteiro, pois já o validamos, então vamos salvar a parte do payload para que possamos acessá-lo posteriormente:



Em toda a nossa aplicação, podemos usar um par dos subvalores id\_token.iss e id\_token. do token de ID como um identificador globalmente único para o usuário atual. Essa técnica é muito mais resistente a colisões do que um nome de usuário ou endereço de e-mail, pois a URL do emissor automaticamente analisa os valores no campo de assunto. Assim que obtivermos o token de ID, enviaremos o usuário para uma página de exibição alternativa, mostrando que ele efetuou login com sucesso como o usuário atual.





Isso nos dá uma exibição incluindo o emissor e o assunto, bem como um botão para buscar as Informações do Usuário do usuário atual. A função de processamento final se parece com a Listagem 15 no Apêndice B.

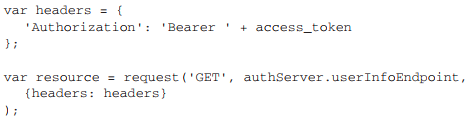
**Fetching the UserInfo (Obtendo o UserInfo)**

Após processarmos o evento de autenticação, é provável que queiramos saber mais sobre o usuário do que um único identificador legível por máquina. Para acessar as informações do perfil, incluindo informações como nome e endereço de e-mail, chamaremos o endpoint UserInfo no IdP usando o token de acesso que recebemos durante o processo do OAuth 2.0. É possível que esse token de acesso também seja usado para recursos adicionais, mas vamos nos concentrar especificamente em seu uso com o endpoint UserInfo. Em vez de baixar automaticamente as informações do usuário imediatamente após a autenticação, faremos com que nossa RP chame o endpoint UserInfo somente quando necessário. Em nossa aplicação, salvaremos isso no objeto userInfo e o renderizaremos em uma página web. Já incluímos o modelo de renderização no projeto para você, então começaremos criando uma função de manipulador para /userinfo no cliente.

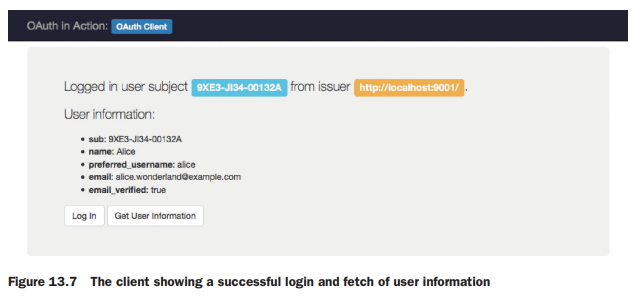
|  |
| --- |
|  |

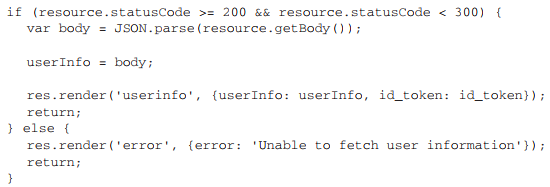


Esta chamada funciona como qualquer outro recurso protegido pelo OAuth 2.0. Neste caso específico, faremos uma solicitação HTTP GET com o token de acesso no cabeçalho de autorização.



O endpoint UserInfo retorna um objeto JSON que podemos salvar e processar como acharmos adequado. Se recebermos uma resposta bem-sucedida, salvaremos as informações do usuário e as enviaremos ao nosso modelo de renderização. Caso contrário, exibiremos uma página de erro.





Isso deve gerar uma página semelhante à mostrada na figura 13.7. E é só isso. Tente autorizar diferentes escopos e observe a diferença que isso faz nos dados retornados do endpoint. Se você já escreveu um cliente OAuth 2.0 (o que já aconteceu no capítulo 3), tudo isso deve parecer trivial, e por um bom motivo: o OpenID Connect foi projetado desde o início para ser algo construído sobre o OAuth 2.0. Para um exercício adicional, conecte a página /userinfo do cliente para exigir um login válido do OpenID Connect. Ou seja, deve haver um token de ID válido, bem como um token de acesso que possa ser usado para buscar informações do usuário já armazenadas no cliente quando alguém acessa essa página e, se não houver, o cliente iniciará automaticamente o processo do protocolo de autenticação.

**Resumo**

uitas pessoas acreditam erroneamente que o OAuth 2.0 é um protocolo de autenticação, mas agora você sabe a verdade sobre o assunto.

* O OAuth 2.0 não é um protocolo de autenticação, mas pode ser usado para criar um protocolo de autenticação.
* Muitos protocolos de autenticação existentes que foram criados usando o OAuth 2.0 estão em uso na web hoje, a maioria deles vinculados a provedores específicos.
* Os projetistas de protocolos de autenticação cometem muitos erros comuns com o OAuth 2.0. Esses erros podem ser evitados com um design cuidadoso do protocolo de autenticação.
* Com algumas adições importantes, o servidor de autorização e o recurso protegido do OAuth 2.0 podem atuar como um provedor de identidade, e o cliente OAuth 2.0 pode atuar como uma parte confiável.
* O OpenID Connect fornece um protocolo de autenticação de padrão aberto cuidadosamente projetado, construído com base no OAuth 2.0.

Agora que vimos um protocolo importante criado com base no OAuth 2.0, vamos analisar mais de perto vários outros que estão resolvendo casos de uso avançados.