O que há de novo no Java 11

Introdução

Por décadas, o Java tem sido uma das linguagens de programação mais usadas no software corporativo, e seu release é mais rápido do que nunca. A partir do Java 9, lançado em 2017, a cadência de lançamento foi alterada para a cada seis meses. O Java 10 adicionou novos recursos à plataforma. Para muitos usuários, no entanto, as mudanças não foram substanciais o suficiente para justificar o esforço de migração para o Java 10. Mas o Java 11 vai além dos novos recursos típicos e das correções de bugs.

Como um desenvolvedor Java, é difícil acompanhar os desenvolvimentos mais recentes e maiores incluídos em cada versão. Este relatório explica os benefícios da atualização para o Java 11 e oferece uma visão geral rápida, mas condensada e perspicaz de seus recursos atraentes.

Migrar de uma versão anterior do Java pode não ser tão simples quanto seria de se esperar. O Java 11 remove uma variedade de tecnologias e APIs da plataforma. Neste relatório, você aprende sobre as estratégias de migração que ajudam a acelerar o mais rápido possível. Mas, apesar dos obstáculos em potencial, os novos recursos do Java 11 fazem valer a atualização.

Convergência do Oracle JDK e OpenJDK

O OpenJDK, a contraparte de software livre do Oracle JDK, existe desde o Java 7. Na maioria das vezes, os recursos de linguagem, as principais bibliotecas Java e as ferramentas de ambas as distribuições são muito semelhantes, se não exatamente iguais. Além desses aspectos, as versões do Oracle JDK antes do Java 11 incluem recursos comerciais como o Flight Recorder, o Java Mission Control e o Z Garbage Collector que podem ser ativados passando a opção `-XX:+UnlockCommercialFeatures` ao iniciar uma nova JVM. Com o Java 11, os recursos comerciais tornam-se gratuitos e foram adicionados ao OpenJDK. Discutimos alguns desses recursos em uma seção posterior.

O objetivo de longo prazo da Oracle era convergir ambas as distribuições, tornando o ferramental comercial totalmente aberto. Com o Java 11, esse marco foi alcançado, juntamente com algumas pequenas diferenças de embalagens e cosméticos . Então, qual distribuição você deve escolher para o seu projeto, você pode perguntar? Vamos dar uma olhada nas implicações de licença para cada opção.

Licenciamento do Oracle JDK e OpenJDK

Há um único fator de distinção entre o Oracle JDK e o OpenJDK: o contrato de licença. O OpenJDK continua a ser licenciado com a Licença Pública Geral GNU, versão 2 (GPL 2), com uma exceção de caminho de classe. Esta licença permite que você use Java para projetos de código aberto, bem como produtos comerciais, sem quaisquer restrições ou atribuição para fins de desenvolvimento e produção. A exceção de caminho de classe separa os termos de licença do OpenJDK dos termos de licenças aplicáveis a bibliotecas externas empacotadas com o programa executável. Com relação ao Oracle JDK, a licença mudou da Licença de Código Binário (BCL) para as tecnologias Oracle Java SE para uma licença comercial. Como resultado, você não pode mais usar o Oracle JDK para produção sem comprar o modelo de assinatura do Java SE .

NOTA

O modelo de assinatura também se aplica a qualquer release de patch do Java 8 publicado depois de janeiro de 2019, se for usado para produtos comerciais.

É importante que você entenda as implicações da licença para fazer uma escolha certa para sua organização. Se você está trabalhando em software de missão crítica e gostaria de contar com o suporte da Oracle para patches de longo prazo, a licença comercial do Oracle JDK provavelmente é a melhor opção. Para softwares de código aberto e produtos que exigem apenas versões de correção de curto prazo para os próximos seis meses, o OpenJDK deve ser suficiente. De qualquer forma, você precisará decidir qual distribuição Java usar assim que fizer o upgrade para o Java 11. Esteja ciente de que existem distribuições alternativas gratuitas do OpenJDK de outros fornecedores. Para mais informações, veja a postagem do blog Java Champions "Java Is Still Free" .

Cadência de lançamento de seis meses: o que significa para você

Nos primeiros dias de Java, os novos lançamentos eram um grande negócio. Eles costumavam acontecer apenas a cada dois ou três anos e continham muitas mudanças significativas, às vezes violentas, como Generics no Java 5, ou alterações de sintaxe no Java 8 ou no sistema de módulos Java 9. A migração de um projeto de software para uma nova versão do Java muitas vezes exigia muito planejamento e esforço. Alguns projetos até tiveram que adiar o processo de produção de software até que a migração estivesse totalmente concluída.

Muitos líderes do setor mudaram para um modelo de Entrega Contínua para impulsionar a inovação e reduzir os riscos: menos alterações, lançamentos mais frequentes. A Oracle

está comprometida em seguir este exemplo lançando uma nova versão principal a cada seis meses e uma versão menor a cada dois meses. Como resultado, a atualização para uma nova versão do Java terá menos impacto em seu projeto. A Figura 1-1 mostra as datas esperadas das próximas versões principais e secundárias do Java.

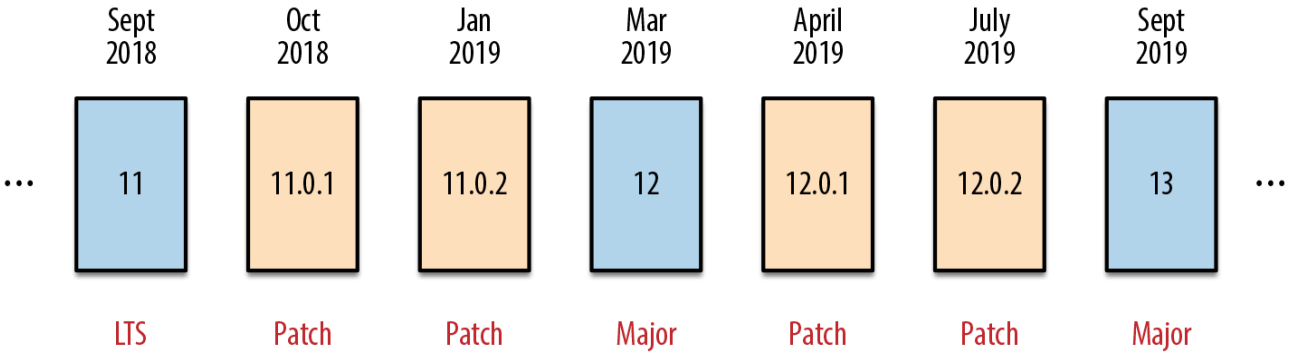

Figura 1-1. Ciclo de lançamento projetado do Java

Com lançamentos mais frequentes, vem a responsabilidade de garantir a estabilidade da plataforma. Por essa razão, a Oracle fornece uma versão de suporte de longo prazo a cada três anos. A seguir, veremos o que significa "suporte de longo prazo" na prática.

Suporte de longo termo

A Oracle oferece suporte Premier vitalício para todas as principais versões de Java que foram designadas como uma versão de suporte a longo prazo (LTS). O benefício de uma versão LTS é que os clientes pagantes receberão atualizações de atualizações periódicas mesmo após a próxima versão principal ter sido lançada. Por exemplo, o próximo lançamento LTS após o Java 11 será o Java 17, planejado para setembro de 2021. Enquanto isso, você continuará recebendo novos lançamentos, conforme explicado na seção anterior. Para qualquer versão principal do Java, os usuários do OpenJDK receberão apenas mais dois lançamentos de correção. Depois disso, supõe-se que você precisará atualizar para a próxima versão principal.

NOTA

Em 30 de outubro de 2018, a Amazon anunciou o suporte gratuito de longo prazo para os tempos de execução Java OpenJDK 8 e OpenJDK 11 no Amazon Linux 2, um sistema operacional de servidor Linux da Amazon Web Services (AWS). Em 14 de novembro, James Gosling anunciou a Corretto , uma distribuição de suporte a longo prazo pronta para produção de Java.

Os lançamentos LTS são especialmente atraentes para clientes empresariais mais conservadores, com uma taxa de atualização mais lenta. Você pode não ter certeza se

precisa de uma licença comercial ou gratuita para seus projetos de software. Uma aposta segura é começar com uma das distribuições livres baseadas no OpenJDK. Se você perceber posteriormente que precisa de suporte comercial, ainda poderá alternar para o Oracle JDK e pagar pela licença.

Com o conhecimento básico do Java 11 em seu currículo, vamos mergulhar nos novos recursos que esta versão tem a oferecer. Eles podem ser pequenos em comparação com outros lançamentos. No entanto, eles podem melhorar a forma como você trabalha com o Java no dia-a-dia.

Novas características

Para muitos desenvolvedores, a principal motivação para atualizar para uma nova versão do Java é aumentar a produtividade aplicando os recursos mais recentes e melhores. Nesta seção, você aprenderá sobre as melhorias de linguagem e plataforma introduzidas com o Java 11. Começaremos examinando o recurso com maior impacto: o novo cliente HTTP.

Comunicação HTTP integrada e aprimorada com o HttpClient

Em um mundo de APIs de serviços e microserviços, a comunicação HTTP é inevitável. É comum precisar escrever um código que faça uma chamada para um terminal para recuperar ou modificar dados. `HttpURLConnection`, uma API para comunicação HTTP, existe há muito tempo, mas não conseguia acompanhar os requisitos dos aplicativos modernos. Como resultado, no passado, os desenvolvedores Java tinham que recorrer a bibliotecas mais avançadas e ricas em recursos, como o Apache HttpComponents ou o Square's OKHttp , que já suportavam comunicação HTTP / 2 e WebSocket.

A Oracle reconheceu essa falha no conjunto de recursos do Java e introduziu uma implementação de cliente HTTP como um recurso experimental com o Java 9. `HttpClient` cresceu e agora é um recurso final no Java 11. Se há um motivo para atualizar para o Java 11, é isso! Não há mais dependências externas que você precise manter como parte de seu processo de criação.

Em suma, a API HTTP do Java pode lidar com comunicação síncrona e assíncrona e suporta HTTP 1.1 e 2, bem como todos os métodos HTTP comuns, como GET, POST, PUT e DELETE. Uma interação HTTP típica com o módulo `java.net.http` se parece com isso:

1. Crie uma instância `HttpClient` e configure-a conforme necessário.

2. Crie uma instância `HttpRequest` e preencha as informações.
3. Passe a solicitação para o cliente, execute a solicitação e recupere uma instância de `HttpResponse`.
4. Processe as informações contidas no `HttpResponse`.

Soa abstrato? Vamos dar uma olhada em exemplos concretos de comunicação HTTP síncrona e assíncrona.

COMUNICAÇÃO SÍNCRONA

Primeiro, vamos escrever um programa que faça uma chamada GET síncrona. A Figura 1-2 mostra que uma chamada síncrona bloqueia o cliente até que a solicitação HTTP tenha sido manipulada pelo servidor e a resposta tenha sido enviada de volta.

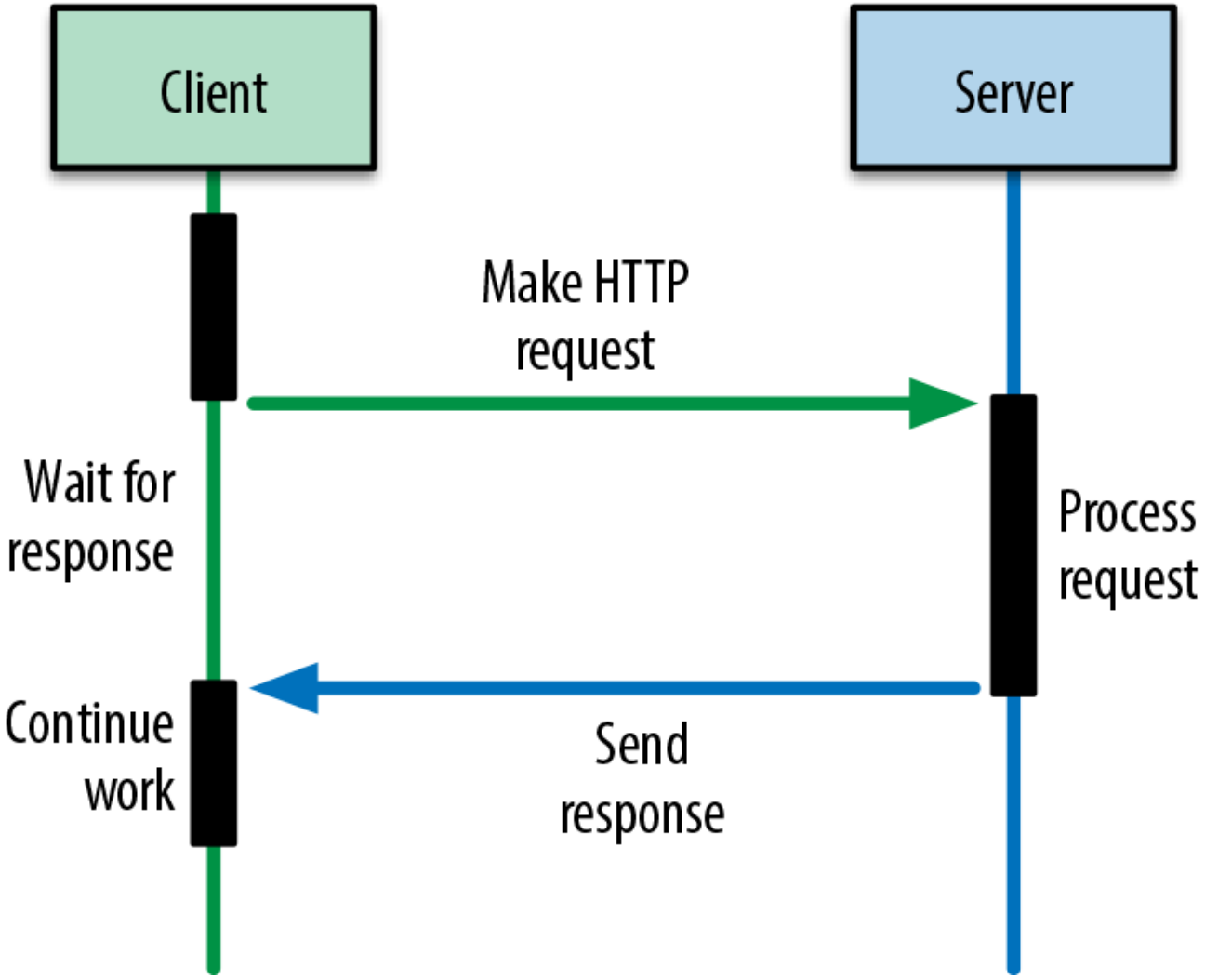

Figura 1-2. Comunicação HTTP Síncrona

Para fins de demonstração, usamos o Postman Echo , um serviço RESTful para testar clientes HTTP. O Exemplo 1-1 cria uma instância do cliente HTTP, executa uma chamada GET e renderiza algumas das informações de resposta no console.

Exemplo 1-1 Fazendo uma chamada GET síncrona com a API HttpClient

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;

HttpClient httpClient = HttpClient.newBuilder()
        .connectTimeout(Duration.ofSeconds(10))
        .build();                                                    ❶

try {
    String urlEndpoint = "https://postman-echo.com/get";
    URI uri = URI.create(urlEndpoint + "?foo1=bar1&foo2=bar2");
    HttpRequest request = HttpRequest.newBuilder()
            .uri(uri)
            .construir();                                            ❷
    HttpResponse<String> response = httpClient.send(request,
            HttpResponse.BodyHandlers.ofString());                   ❸
} catch(IOException | InterruptedException e) {
    throw new RuntimeException(e);
}

System.out.println (“Status Code:" + response.statusCode ());        ❹
System.out.println ("Headers:" + response.headers ()
        .allValues (“content-type"));                                ❹
System.out.println ("Body:" + response.body ());                    ❹
```

❶ Criando e configurando a instância do cliente HTTP. O cliente usa um tempo limite de conexão de 10 segundos.

❷ Criando e configurando a solicitação HTTP. Por padrão, uma solicitação HTTP usa o método GET.

❸ Enviando a solicitação HTTP e recuperando a resposta.

❹ Processamento dos dados retornados pela resposta HTTP; por exemplo, o código de status, os cabeçalhos e o corpo.

> **NOTA**
>
> Você deve ter notado que a API `HttpClient` depende muito do padrão do construtor. O padrão de construtor é especialmente útil ao criar objetos mais complexos que podem ser alimentados com parâmetros opcionais. Em vez de expor os construtores com um número de sobrecargas de parâmetros, o padrão do construtor fornece métodos expressivos para configurar o objeto, levando a uma API fluente que é mais fácil de entender e usar.

A API `HttpClient` não está limitada a apenas fazer uma chamada síncrona. Na seção a seguir, exploramos um exemplo de comunicação assíncrona.

COMUNICAÇÃO ASSÍNCRONA

A comunicação assíncrona é útil se você não quiser esperar a resposta e lidar com a resposta mais tarde ou se quiser processar uma lista de chamadas em paralelo. A Figura 1-3 mostra como isso se parece.

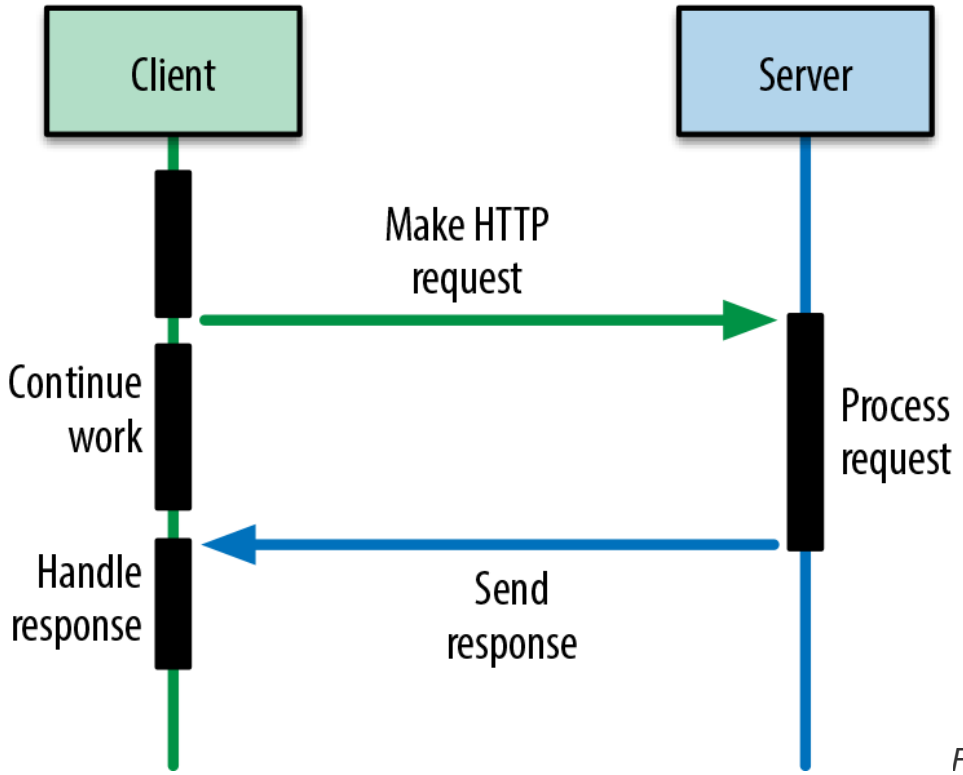

Figura 1-3. Comunicação HTTP assíncrona

Vamos supor que você queira implementar uma ferramenta para analisar links usados em uma página da Web e verificá-los executando uma solicitação HTTP GET contra eles. O

exemplo 1-2 define uma lista de URIs e os verifica emitindo uma chamada assíncrona e coletando os resultados.

Exemplo 1-2 Realizando uma chamada HTTP assíncrona e reagindo à resposta posteriormente

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.stream.Stream;

import static java.util.stream.Collectors.toList;

final List<URI> uris = Stream.of (
        "https://www.google.com/",
        "https://www.github.com/",
        "https://www.ebay.com/"
        ).map(URI::create).collect(toList());                    ❶

HttpClient httpClient = HttpClient.newBuilder()
        .connectTimeout(Duration.ofSeconds(10))
        .followRedirects(HttpClient.Redirect.ALWAYS)
        .build();

CompletableFuture[] futures = uris.stream()
        .map(uri -> verifyUri(httpClient, uri))
        .toArray(CompletableFuture[]::new);                      ❷
CompletableFuture.allOf(futures).join();                         ❷

private CompletableFuture<Void> verifyUri(HttpClient httpClient,
                                          URI uri) {
    HttpRequest request = HttpRequest.newBuilder()
            .timeout(Duration.ofSeconds(5))
            .uri(uri)
```

```
            .build();

    return httpClient.sendAsync(request,
            HttpResponse.BodyHandlers.ofString())
            .thenApply(HttpResponse::statusCode)
            .thenApply(statusCode -> statusCode == 200)
            .exceptionally(__ -> false)
            .thenAccept(valid -> {
                if(valid) {
                    System.out.println("[SUCESSO] Verified"
                            + uri);
                } else {
                    System.out.println("[FAILURE] Coud not "
                            + "verify " + uri);
                }
            });                                                    ❸
}
```

❶ Define uma lista de URIs que devem ser verificados.

❷ Verifica todos os URIs de forma assíncrona e avalia o resultado futuro.

❸ Envia uma solicitação HTTP assíncrona, verifica se o código de status da resposta está OK e reage ao resultado imprimindo uma mensagem.

Como os Exemplos 1-1 e 1-2 demonstram, a nova funcionalidade HttpClient é muito poderosa e expressiva. O HttpClient tem ainda mais recursos na loja do que os mencionados aqui. Ele também pode manipular fluxos reativos, autenticação e cookies, para citar alguns. Para mais informações, consulte o material de aprendizado da Oracle e os Javadocs .

Outro recurso interessante no Java 11 é a capacidade de iniciar um programa Java sem a necessidade de compilação. Vamos explorar como isso funciona.

Lançamento de programas de arquivo único sem compilação

Java não é uma linguagem de script. Para cada programa que você gostaria de executar, você precisa primeiro compilá-lo executando explicitamente o `javac`comando. Essa ida e volta torna menos conveniente tentar pequenos trechos de código para fins de teste. Java 11 muda isso. Agora você pode executar o código-fonte Java contido em um único arquivo sem a necessidade de compilá-lo primeiro, como mostra a Figura 1-4 .

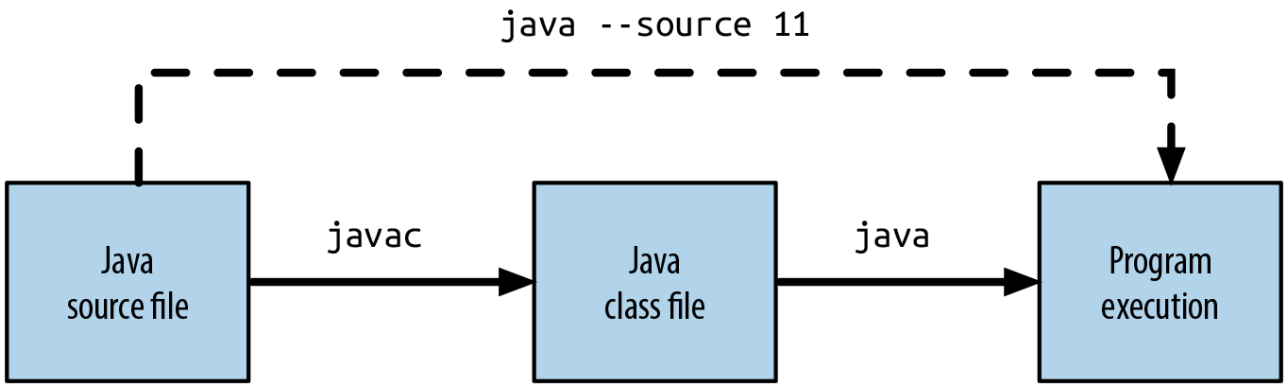

Figura 1-4. Executando um programa Java com e sem compilação

A nova funcionalidade é uma excelente plataforma de lançamento para iniciantes na linguagem Java e permite que os usuários experimentem a lógica de maneira ad hoc.

É fácil começar. Vamos escolher um exemplo simples. O arquivo HelloWorld.java renderiza a mensagem "Hello World" no console por meio de `System.out.println` como parte do método `main`:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println ("Hello World!");
    }
}
```

Para executar, execute o comando `java` com o Java 11. O Java compilará o arquivo de origem rapidamente e executará o programa com o arquivo de classe no caminho de classe. Qualquer erro de compilação que possa ocorrer será renderizado no console:

```
$ java -version
versão openjdk "11" 2018-09-25
OpenJDK Runtime Environment 18.9 (build 11 + 28)
OpenJDK 64-Bit Server VM 18.9 (compilação 11 + 28, modo misto)

$ java HelloWorld.java
Olá Mundo!
```

O lançamento de um programa de arquivo único sem compilação é muito conveniente, mas há algumas limitações que você precisa conhecer. Com o Java 11, o programa não pode usar nenhuma dependência externa além do módulo `java.base`. Além disso, você pode iniciar apenas um programa de arquivo único. Atualmente, os métodos de chamada

de outros arquivos de origem Java não são suportados, mas você pode definir várias classes no mesmo arquivo, se necessário. Eu pessoalmente vejo esse recurso como uma ferramenta útil para prototipagem rápida.

Na próxima seção, você aprenderá sobre os novos métodos que foram introduzidos nas classes de biblioteca existentes.

Novos métodos de biblioteca para seqüências de caracteres, coleções e arquivos

Com muita frequência, os desenvolvedores Java precisam confiar em bibliotecas externas, como o Google Guava ou o Apache Commons, para métodos convenientes que, de outra forma, precisariam copiar e colar de um projeto para outro. As implementações podem ser pequenas, mas criam superfícies de API adicionais que precisam ser mantidas e testadas. Java 11 adiciona métodos de conveniência para as APIs `String`, `Collections` e `Files`.

Vamos explorar a nova funcionalidade pelo exemplo. Por diversão, vamos escrever alguns testes JUnit que aplicam as novas APIs e verificar seu comportamento correto para diferentes casos de uso.

MELHORIAS NA API DE STRING

A API `String` adiciona quatro novos métodos de conveniência. Vamos dar uma olhada em cada um deles pelo exemplo.

Repetindo uma string

O primeiro método que olhamos é `String.repeat(Integer)`. A julgar pelo nome, você já deve ter adivinhado o objetivo da funcionalidade. Simplesmente repete uma corda n vezes.

Eu posso pensar em mais de uma situação em que o método `repeat` poderia ser útil. Imagine que você precise criar uma tabela de dados com espaços em branco acolchoados entre pares chave / valor para melhorar sua legibilidade. O caso de teste a seguir demonstra o uso do método para esse mesmo cenário:

```
@Test
void canRepeatString () {
    assertEquals ("CPU Usage:                 5%",
```

```
            renderInfo ("CPU Usage:", "5%"));
    assertEquals ("Memory Usage:          9.14 MB",
            renderInfo ("Memory Usage:", "9,14 MB"));
    assertEquals ("Free Disk:             96,5 GB",
            renderInfo ("Free Disk:", "96,5 GB"));
}

private String renderedInfo (String title, String value) {
    return title + " ".repeat(30 - title.length()
            - value.length()) + value;
}
```

Verificando espaços vazios ou em branco

Tenho certeza de que você precisou implementar a validação de dados em algum momento de sua carreira. Suponha que você queira verificar a validade de um valor inserido em um campo de entrada baseado na web. Claro, não queremos permitir valores em branco.

O Java 11 introduz o método `String.isBlank()`, que indica se uma string está vazia ou contém apenas espaços em branco. Agora que essa funcionalidade se tornou um recurso principal do Java, não é mais necessário usar o Apache StringUtils como dependências externas. Você pode ver seu uso no seguinte caso de teste:

```
@Test
void canCheckIfStringContainsWhitespaces () {
    String nameFormFieldWithoutWhitespace = "Duke";
    String nameFormFieldWithWhitespace = " ";
    assertFalse(nameFormFieldWithoutWhitespace.isBlank());
    assertTrue(nameFormFieldWithWhitespace.isBlank());
}
```

Removendo espaços em branco iniciais e finais

Validar strings anda de mãos dadas com cadeias sanitizantes. Os desenvolvedores geralmente encontram dados que precisam ser limpos para extrair as informações reais. Isso pode envolver a filtragem de caracteres ocultos, errados ou inúteis.

O método recém-introduzido `String.strip()` cuida da remoção de espaços em branco iniciais e finais. Você pode ser ainda mais específico removendo apenas os caracteres iniciais usando `String.stripLeading()` ou apenas os caracteres finais usando `String.stripTrailing()`. O caso de teste a seguir demonstra o uso de todos os três métodos:

```
@Test
void canStripStringOfLeadingAndTrailingWhitespaces() {
    String nameFormField = "       Java Duke";
    assertEquals ("Java Duke", nameFormField.strip());
    assertEquals ("Java Duke     ", nameFormField.stripLeading());
    assertEquals ("      Java Duke", nameFormField.stripTrailing());
}
```

Processando strings de múltiplas linhas

O processamento de strings de múltiplas linhas por linha pode ser entediante e consome muita memória. Primeiro, você deve dividir a string com base nos terminadores de linha. Em seguida, você precisa iterar sobre cada linha. Para sistemas de software que lidam com a operação de textos grandes, o consumo de memória pode ser uma grande preocupação.

O método `String.lines()` introduzido no Java 11 permite processar textos de múltiplas linhas como `Stream`. A `Stream` representa uma seqüência de elementos que podem ser processados seqüencialmente sem ter que carregar todos os elementos na memória de uma só vez. A seguir, é mostrado um caso de teste que usa o método na prática:

```
@Test
void canStreamLines() {
    String testString = "This\nis\na\ntest";
    List<String> lines = new ArrayList<>();
    testString.lines().forEach(line -> lines.add(line));
    assertEquals(List.of("This", "is", "a", "test"), lines);
}
```

APRIMORAMENTOS DA API DE COLEÇÕES

No passado, transformar uma coleção em um array era complicado. Você foi solicitado a fornecer uma nova instância da matriz, seu tipo genérico e o tamanho final da matriz.

O Java 11 torna a conversão mais conveniente. Em vez de passar uma instância de matriz, agora você pode fornecer uma função usando o método `Collection.toArray(IntFunction<T[]>)`. O exemplo de código a seguir compara a maneira antiga e a nova maneira de converter uma coleção em uma matriz:

```
@Test
void canTurnListIntoArray() {
    List<String> months = new ArrayList<>();
    months.add("January");
    months.add("February");
    months.add("March");
    assertArrayEquals(new String []
            { “January”, ”February”, ”March" },
            months.toArray(new String[months.size()])); ❶
    assertArrayEquals(new String[]
            { “January”, ”February”, ”March" },
            months.toArray(String[]::new)); ❷
}
```

❶ Convertendo uma coleção em uma matriz pré-Java 11.

❷ Convertendo uma coleção em um array usando o novo método no Java 11.

APRIMORAMENTOS DA API DE ARQUIVOS

Para os desenvolvedores, ler e gravar arquivos é uma operação muito comum. O código pré-Java 11 exige que você escreva código clichê desnecessário. Para contornar esse problema, os projetos geralmente introduzem um método de utilitário reutilizável ou recorrem a uma das muitas bibliotecas de código aberto.

O Java 11 adiciona os métodos `Files.readString(Path)` e `Files.writeString(Path, CharSequence, OpenOption)` com várias sobrecargas, o que facilita muito a leitura e gravação de arquivos. O trecho de código a seguir mostra os dois métodos em ação:

```
@Test
void canReadString() throws URISyntaxException, IOException {
    URI txtFileUri = getClass().GetClassLoader()
            .getResource("helloworld.txt").toURI();
    String content = Files.readString(Path.of(txtFileUri),
            Charset.defaultCharset());
    assertEquals("Hello World!", content);
}

@Test
void canWriteString() throws IOException {
    Path tmpFilePath = Path.of(
            File.createTempFile("tempFile", ".tmp").ToURI());
    Path returnedFilePath = Files.writeString(tmpFilePath,
            "Hello World!", Charset.defaultCharset(),
            StandardOpenOption.WRITE);
    assertEquals(tmpFilePath, returnedFilePath);
}
```

Aprimoramentos para Opcional e Predicado

O Java 8 adicionou o tipo `Optional`. Esse tipo é um contêiner de valor único que contém um valor opcional. Seu principal objetivo é evitar sobrecarregar o código com verificações desnecessárias de nulos. Você pode verificar se um valor está presente no contêiner sem realmente recuperá-lo chamando o método `isPresent()`.

Na versão mais recente do Java, você também pode solicitar o inverso: "O valor está vazio?" Em vez de usar a negação do método `isPresent()`, você pode agora usar o método `isEmpty()` para tornar o código mais fácil de entender. O caso de teste a seguir compara o uso de ambos os métodos:

```
@Test
void canCheckOptionalForEmpty () {
    String payDay = null;
    assertTrue(!Optional.ofNullable(payDay).isPresent());          ❶
    assertTrue(Optional.ofNullable(payDay).isEmpty());             ❷
    payDay = "Monday";
    assertFalse(!Optional.ofNullable(payDay).isPresent());         ❶
    assertFalse(Optional.ofNullable(payDay).isEmpty());            ❷
```

```
}
```

❶ Verificando se um valor `Optional` está presente antes do Java 11.

❷ Verificando se um valor `Optional` está presente no Java 11.

A `Predicate` é frequentemente usado para filtrar elementos em uma coleção de objetos. O Java 11 adiciona o método estático `Predicate.not(Predicate)`, que retorna uma instância `Predicate` predefinida que nega o dado `Predicate`.

Suponha que você queira filtrar um fluxo de strings, como certos meses. O novo método `Predicate` pode ser extremamente útil com a filtragem de todos os meses que não cumprem uma determinada condição. No código a seguir, você encontrará um exemplo que filtra todos os meses que não começam com a letra "M". Isso é muito mais expressivo do que passar:

```
(Predicate<String>)month -> month.startsWith("M").negate()
```

para o método `filter`.

```
@Test
void canUsePredicateNotAsFilter() {
    List<String> months = List.of("Janeiro", "Fevereiro", "Março");
    Listar <string> filteredMonths = months
            .stream()
            .filter(Predicate.not(month -> month.startsWith("M")))
            .collect(Collectors.toList());
    assertEquals(List.of("Janeiro", "Fevereiro"), filteredMonths);
}
```

É isso para as extensões da API da biblioteca principal. O Java 11 também melhorou o tratamento da palavra-chave `var`. Em seguida, analisamos essas mudanças.

USANDO A PALAVRA-CHAVE VAR NO LAMBDAS

A partir do Java 10, você pode declarar variáveis locais sem precisar atribuir o tipo. Em vez de declarar o tipo, a variável usa a palavra-chave `var`. No tempo de execução, a atribuição da variável determina automaticamente o tipo; é inferido. Mesmo ao usar a palavra-chave `var`, a variável ainda é digitada estaticamente.

Com o Java 11, agora você também pode usar a palavra-chave `var` para parâmetros de um Lambda. A utilização de `var` nos parâmetros do Lambda tem um grande benefício: você pode anotar a variável. Por exemplo, você poderia indicar que o valor da variável não pode ser nulo usando a anotação do JSR-303 `@NotNull` .

Vamos voltar ao método `Predicate.not(Predicate)` usado em um exemplo anterior. No exemplo a seguir, aprimoramos a definição do Lambda fornecendo a palavra-chave `var` e uma anotação:

```
import javax.validation.constraints.NotNull;

@Test
void canUseVarForLambdaParameters() {
    List<String> months = List.of (“January”, ”February”, ”March");
    List<String> filteredMonths = months
            .stream()
            .filter(Predicate.not((@NotNull var month) ->
                    month.startsWith("M")))
            .collect(Collectors.toList());
    assertEquals(List.of("January", "February"), filteredMonths);
}
```

Em seguida, vamos dar uma olhada na mudança do Unicode 8 para o Unicode 10.

ADOÇÃO DO UNICODE 10

O Java 10 usa o padrão Unicode 8 para localização. O padrão Unicode evolui continuamente, então já era hora de o Java 11 seguir a versão mais recente, Unicode 10. A atualização do Unicode 8 para o 10 inclui 16.018 novos caracteres e 10 novos scripts. Um script é uma coleção de letras e outros sinais escritos usados para representar informações textuais. Por exemplo, o Unicode 10 adiciona Masaram Gondi , uma linguagem sul-central dravidiana.

Uma das mais esperadas adições ao Unicode 10 é o "Colbert emoji" , um rosto com uma boca neutra e uma sobrancelha erguida, tornada popular pelo comediante Stephen Colbert . O exemplo a seguir faz uso do emoji Colbert:

```
@Test
void canUseColbertEmoji() {
    String unicodeCharacter = "\uD83E\uDD28";
    assertEquals("🤨", unicodeCharacter);
}
```

Em seguida, analisamos uma mudança de nível mais baixo no Java 11: controle de acesso baseado em aninhamento. O impacto para o usuário final é menos visível, mas trata de uma dívida técnica de longa data que foi finalmente refatorada.

CONTROLE DE ACESSO BASEADO EM NINHO

Para entender a próxima mudança para o Java 11, precisamos dar uma olhada em uma classe aninhada. Vamos supor que você tenha uma classe externa chamada `Outer.java` e uma classe interna chamada `Inner.Java` dentro dela. `Inner.java` pode acessar todos os campos e métodos particulares `Outer.java` porque eles logicamente pertencem. A figura 1-5 visualiza a estrutura de dados.

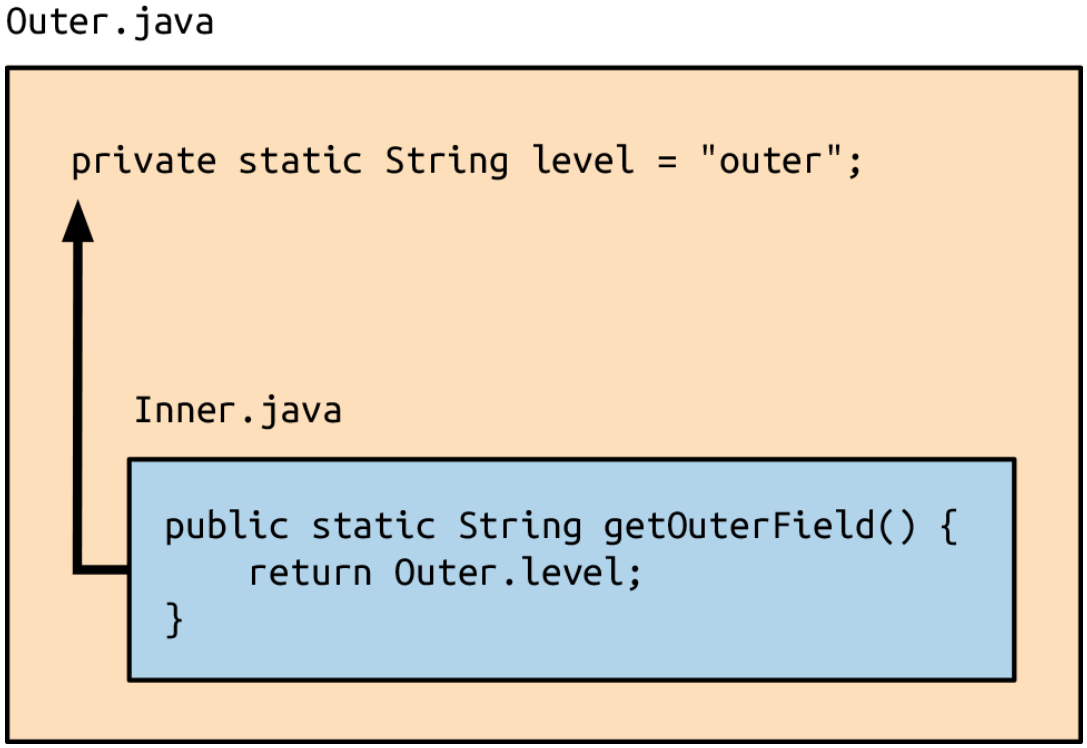

Figura 1-5 Acessando um campo pai de uma classe aninhada

Como você deve saber, o compilador Java cria um novo arquivo de classe para classes internas, neste caso `Outer$Inner.class`. Debaixo dos panos, o processo de compilação também cria os chamados métodos de ponte de ampliação de acessibilidade entre as classes.

O Java 11 paga a dívida técnica de ter que gerar esses métodos de ponte com a ajuda dos chamados companheiros de ninho. As classes internas agora podem acessar campos e métodos de classes externas sem trabalho adicional do compilador.

Você pode se perguntar: "As otimizações do compilador são ótimas, mas como isso me afeta como desenvolvedor"? Bem, antes do Java 11, não era possível acessar um campo da classe externa via reflexão sem definir o controle de acessibilidade para `true`. Com a alteração do controle de acesso baseado em aninhamento, esse controle não é mais necessário. A hierarquia de classes a seguir demonstra o comportamento alterado:

```
import java.lang.reflect.Field;
public class Outer {
    private static String level = "outer";
    public static class Inner {
        public static String getOuterViaRegularFieldAccess() {
            return Outer.level;
        }
        public static String getOuterViaReflection() {
            try {
                Field levelField = Outer.class
                        .getDeclaredField("level");
                // levelField.setAccessible(true);          ❶
                return levelField.get(null).toString ();
            } catch(NoSuchFieldException
                    | IllegalAccessException e) {
                throw new RuntimeException(e);
            }
        }
    }
    public static void main(String[] args) {
        Inner.getOuterViaRegularFieldAccess();              ❷
        Inner.getOuterViaReflection();
    }
}
```

❶ Necessário para o Java 10 ou anterior.

❷ Já é válido no Java 10.

Como com os exemplos de código anteriores nesta seção, também verificamos o comportamento correto com a ajuda de um teste, conforme mostrado a seguir (para obter informações mais detalhadas sobre essa alteração, consulte o JEP 181):

```
private final Outer.Inner = new Outer.Inner();

@Test
void canAccessPrivateFieldFromOuterClass() {
    assertEquals("outer", inner.getOuterViaRegularFieldAccess());
}

@Test
void canAccessPrivateFieldFromOuterClassViaReflection() {
    assertEquals("outer", inner.getOuterViaReflection());
}
```

O último recurso visível ao desenvolvedor que exploramos é o Java Flight Recorder, um recurso que costumava estar disponível apenas para uso comercial. Mantenha esta ferramenta em mente se você precisar criar um perfil de um aplicativo Java.

GRAVANDO EVENTOS DO SO E DA JVM COM O JAVA FLIGHT RECORDER

O Java Flight Recorder (JFR) é uma ferramenta para criação de perfil de um aplicativo Java em execução e captura de diagnósticos para análise adicional. Esta ferramenta não é nova no Java. Ele foi implementado como parte do Java 7, mas foi considerado um recurso comercial que você precisava ativar explicitamente usando as opções da JVM `-XX: +UnlockCommercialFeatures -XX:+FlightRecorder`.

No Java 11, o JFR foi transformada em open source e está incluído no OpenJDK. Para lhe dar uma impressão sobre como usar o JFR, vamos dar uma olhada nas etapas necessárias para capturar as métricas. Primeiro, você precisa executar um programa Java e fornecer as opções de VM necessárias para ativar a gravação. No exemplo a seguir, usamos a opção de gravação baseada em tempo definida como 60 segundos:

```
$ java -XX:StartFlightRecording=duration=60s,
filename=recording.jfr,settings=prifile,name=SampleRecording MyMain
Started recording 1. The result will be written to:
/Users/bmuschko/dev/projects/whats-new-in-java-11/recording.jfr
```

Como você deve ter notado na saída do console, os dados de criação de perfil foram capturados no arquivo recording.jfr . É hora de dar uma olhada visual para que possamos analisar as métricas. Conheça o Java Mission Control (JMC)!

O JMC pode inspecionar os dados produzidos pelo JFR e visualizá-lo em uma interface de usuário bem organizada. Esteja ciente de que o JMC não faz parte do JDK; você deve baixar e instalar de forma independente.

> **NOTA**
>
> Gravações JFR não são compatíveis com versões anteriores. Você precisará de uma versão de acesso antecipado do JMC 7 para inspecionar uma gravação produzida com o Java 11.

Para começar, basta apontar o JMC para o arquivo de gravação produzido pelo JFR. A Figura 1-6 mostra o JMC renderizando o consumo de memória de um aplicativo de amostra. Não vou aprofundar os detalhes sobre a funcionalidade do JFR e JMC. Há muitas informações disponíveis online se você planeja usar as ferramentas para seus próprios projetos.

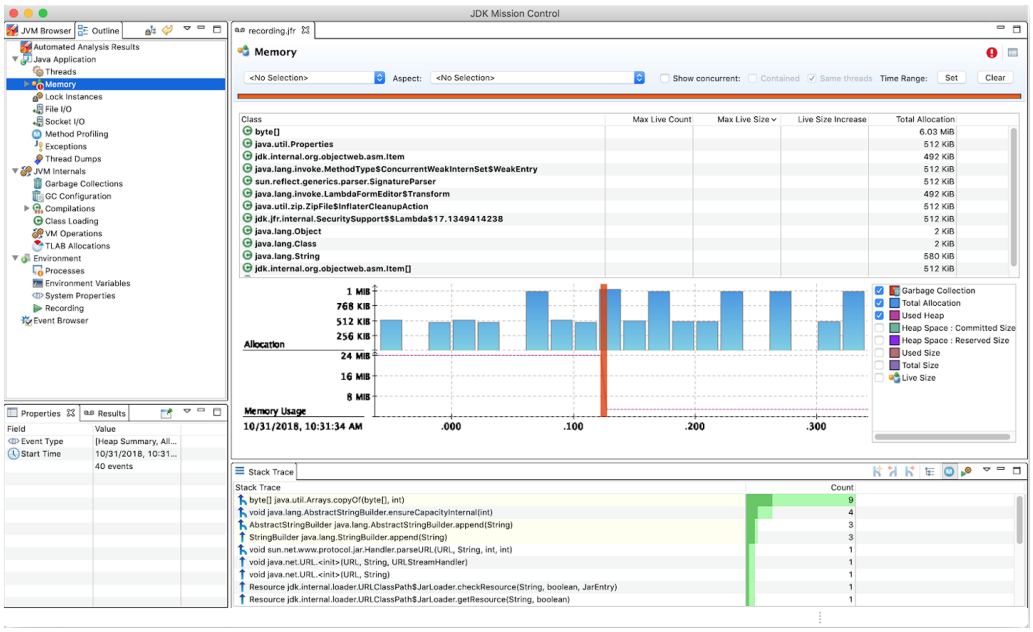

Figura 1-6. Renderização JMC registrou o consumo de memória

É isso para os novos recursos. Tenho certeza de que você descobrirá que as novas funcionalidades que discutimos aqui ajudarão você a se tornar mais produtivo no futuro. Em seguida, discutimos os recursos e APIs que foram removidos com o Java 11.

Recursos e APIs removidos

Atualizar um projeto para uma versão mais recente do Java parece fácil na teoria. Mas, na prática, existem vários fatores que complicam o processo de atualização. Um exemplo é o uso contínuo de recursos e APIs obsoletos. Uma boa prática é revisar periodicamente seu código, identificar o uso de funcionalidade reprovada e localizar um substituto o mais cedo possível.

O Java 11 remove recursos e APIs que foram marcados como obsoletos em versões anteriores. Nesta seção, revisamos cada um deles e propomos opções alternativas, quando aplicável. Esta seção também serve para fornecer informações que podem ajudar você a avaliar como uma remoção pode afetar seu projeto.

Módulos Java EE Removidos

O Java 9 iniciou o processo de modularizar o JDK. Ao modularizar o Java, tornou-se claro que algumas tecnologias e APIs pertenciam ao Java EE em vez do Java SE. A equipe Java preteriu esses módulos e os removeu com a próxima versão do LTS. Assim, com o lançamento do Java 11, essas tecnologias não fazem mais parte do JDK. A Tabela 1-1 lista os módulos removidos.

Tabela 1-1. Módulos removidos do Java 11

Módulo	Pacotes
JavaBeans Activation Framework (JAF)	javax.activation
Java Transaction API (JTA)	java.transaction
API de anotação do Commons	java.xml.ws.annotation
Arquitetura Java para ligação XML (JAXB)	jdk.xml.bind
API Java para serviços da Web baseados em XML (JAX-WS)	jdk.xml.ws
CORBA	javax.activity, javax.rmi, org.omg

Então, o que acontecerá se você usar um desses módulos em seu código e atualizar diretamente para o Java 11? Seu código não será mais compilado ou você poderá encontrar erros de tempo de execução. Embora as versões anteriores permitissem uma

solução alternativa adicionando manualmente o módulo `--add-modules`, isso não funciona mais para o Java 11.

Sua melhor aposta é encontrar um substituto para a funcionalidade disponível como uma dependência externa e adicioná-la ao seu classpath de compilação. Você descobrirá que muitos dos módulos removidos têm uma representação no Maven Central. A Tabela 1-2 lista algumas substituições possíveis. Para referências adicionais, consulte o JEP 320 .

Tabela 1-2 Bibliotecas de substituição para módulos removidos no Java 11

Módulo	Substituição na Central Maven
JavaBeans Activation Framework (JAF)	javax.activation:javax.activation-api
Java Transaction API (JTA)	javax.transaction:javax.transaction-api
API de anotação do Commons	javax.annotation:javax.annotation-api
Arquitetura Java para ligação XML (JAXB)	org.glassfish.jaxb:jaxb-runtime
API Java para serviços da Web baseados em XML (JAX-WS)	com.sun.xml.ws:jaxws-ri
CORBA	org.glassfish.corba:glassfish-corba

Para ilustrar um cenário de migração, vamos supor que você esteja trabalhando em um projeto que precise processar XML. Uma parte da base de código usa o JAXB para converter um POJO e seus valores de campo em XML. Na listagem a seguir, você pode encontrar um POJO, incluindo anotações típicas de JAXB:

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "person")
public class Person {
    private String name;
```

```
    @XmlElement(name = "fullName")
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Compilar a classe com o Java 11 resultaria em um erro semelhante, mostrado no trecho a seguir. As classes JAXB foram removidas do JDK:

```
/Users/bmuschko/dev/projetos/whats-new-in-java-11/src/main/java/
com/bmuschko/java11/removals/jaxb/Person.java
Erro:(3, 33) java: package javax.xml.bind.annotation
does not exist
Erro:(4, 33) java: package javax.xml.bind.annotation
does not existe
Erro:(6, 2) java: cannot find symbol
  symbol: class XmlRootElement
```

Para corrigir o problema de compilação, você precisa adicionar uma dependência externa ao seu arquivo de compilação. O trecho XML a seguir demonstra o uso da dependência de tempo de execução do Glassfish JAXB em um arquivo pom.xml do Maven :

```
<dependencies>
    <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
        <version>2.3.1</version>
    </dependency>
</dependencies>
```

Claro, você pode adicionar a mesma dependência a um script de construção Gradle. O script de construção de amostra a seguir declara a dependência no contexto de um projeto de biblioteca Java:

```
plugins {
    id 'java-library'
}

repositories {
    mavenCentral ()
}

dependencies {
    implementation 'org.glassfish.jaxb:jaxb-runtime:2.3.1'
}
```

Além dos módulos Java EE removidos, o Java 11 também removeu várias APIs. Como o impacto dessas remoções no usuário final é relativamente pequeno, não abordamos mais detalhes sobre isso aqui. Para mais informações, confira a lista de APIs removidas .

Applets e Java Web Start

Lembre-se daqueles dias em que costumávamos pensar em Java principalmente no contexto de Applets? Bem, esse tempo passou há muito tempo. Atualmente, o Java é usado em muitos outros contextos, como aplicativos de servidor, dispositivos móveis e dispositivos Internet of Things (IoT).

O avanço rápido para 2018, e os provedores dos navegador já começaram a retirar ou já retiraram o suporte para tecnologias baseadas plug-in como Flash e Java por causa de desempenho e preocupações com segurança. O Java 9 seguiu a tendência e o suporte preterido para Applets em 2017. O Java 11 fecha o círculo e remove os Applets completamente sem uma substituição.

O Java Web Start fornece uma maneira de baixar um aplicativo de área de trabalho Java a partir de um navegador, em vez de executá-lo no navegador. Para muitos usuários, foi uma surpresa que o Java 11 também remova o suporte para o Java Web Start. Por esse motivo, a conversão de um Applet para um aplicativo Java Web Start está fora de questão.

Se você ainda estiver mantendo aplicativos baseados em Applets ou Java Web Start, precisará procurar uma substituição se estiver planejando fazer a atualização para o Java 11. A próxima seção aborda o JavaFX, uma opção válida e poderosa para o desenvolvimento de aplicativos de desktop.

JavaFX migra para o OpenJFX

A partir do Java 11, o JavaFX - a tecnologia para a criação de aplicativos cliente Java - não é mais distribuído com o Oracle JDK. Mas isso não significa que o projeto esteja morto. Foi simplesmente reduzido para OpenJFX , a contraparte de código aberto do JavaFX. Consequentemente, o OpenJFX pode liberar novas versões independentemente do ciclo de lançamento do JDK. Especialmente com o envolvimento de toda a comunidade Java, esta é uma grande vitória. Provavelmente, podemos esperar lançamentos mais frequentes.

O OpenJFX fornece uma página de documentação muito acessível e detalhada, conforme mostrado na Figura 1-7 . Você pode integrar a tecnologia com os processos de construção Maven e Gradle. Começar a usar o OpenFX não poderia ser mais fácil.

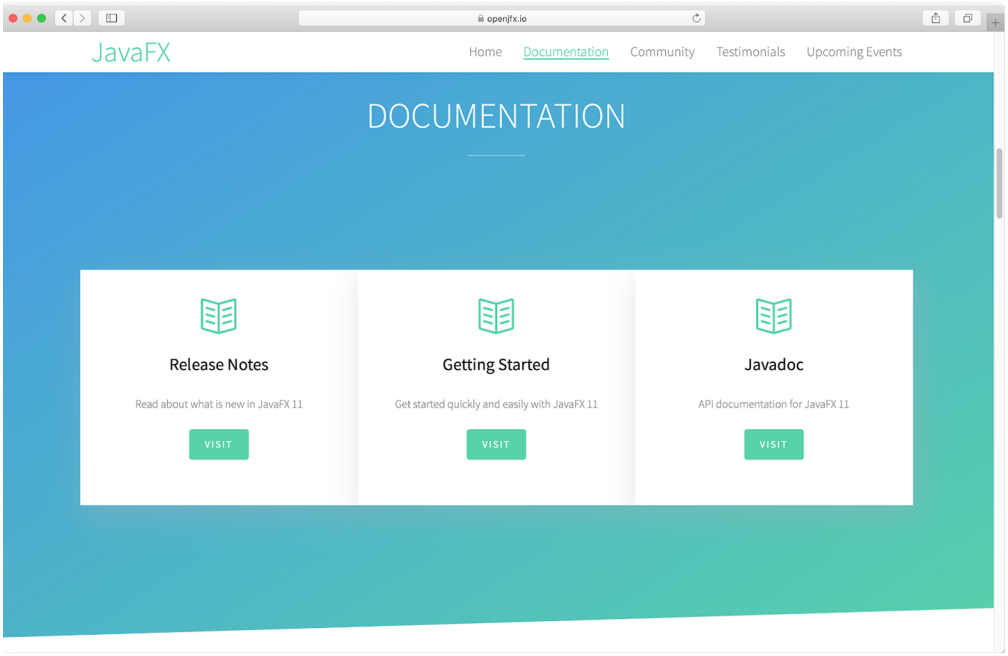

Figura 1-7. O lar da comunidade do OpenJFX

Como nas versões anteriores do Java, os recursos e as APIs passam por um ciclo de reprovação antes de serem realmente removidos, o que discutiremos na próxima seção.

Recursos e APIs descontinuados

O Java 11 descontinua o mecanismo JavaScript do Nashhorn, algumas opções de VM e suporta o esquema de compactação Pack200. Eu aconselho vivamente a incorporação das substituições recomendadas o mais rápido possível para garantir um procedimento de atualização suave para uma versão pós-Java 11. Vamos passar por eles um por um.

NOTA

É difícil acompanhar todas as versões do Java, especialmente se você estiver passando por várias versões. A ferramenta `jdeps` ajuda a detectar o uso de APIs reprovadas no código do projeto e em suas dependências. Integrar `jdeps` como parte de seu processo de Integração Contínua é uma ótima ideia. Você receberá feedback rápido e obterá insight profundo sobre o uso de APIs obsoletas.

Motor de JavaScript Nashorn

A equipe de Java está em uma missão para emagrecer o JDK. O próximo candidato a remoção é o Nashorn, o mecanismo JavaScript. Uma das principais motivações para desaprovar o Nashorn é o ritmo acelerado com que a linguagem ECMAScript constrói e suas APIs mudam. Consequentemente, é um desafio manter a compatibilidade e, ao mesmo tempo, garantir que novos recursos sejam adicionados.

Instanciar o mecanismo Nashorn ainda funciona bem, como mostra o seguinte trecho de código:

```
import javax.script.ScriptEngineManager;
new ScriptEngineManager().getEngineByName("nashorn");
```

A única coisa que você veria seria uma mensagem de aviso indicando que o recurso será removido em uma versão futura do Java:

```
Warning: Nashorn engine is planned to be removed from
a future JDK release
```

Você deve ter notado que a mensagem não é muito específica sobre qual versão tornará a remoção efetiva. Como mencionado por Thomas Wuerthinger, diretor de pesquisa sênior do Oracle Labs, isso provavelmente acontecerá assim que a GraalVM for considerada pronta para produção.

Embora a depreciação do mecanismo de JavaScript da Nashorn provavelmente tenha o maior impacto nos projetos de usuários, há algumas opções da JVM e outras ferramentas que foram marcadas para remoção posterior que merecem ser mencionadas.

Opções de VM e outras ferramentas

Existem duas opções de VM que merecem ser mencionadas e que foram descontinuadas com o Java 11, uma das quais já mencionamos em "Gravando Eventos do SO e da JVM com o Java Flight Recorder" . Antes do Java 11, os recursos comerciais estavam disponíveis apenas para clientes pagantes. Todas essas ferramentas foram abertas, então não há mais necessidade de sinalizadores - `XX:+UnlockCommercialFeatures`e `–XX:+LogCommercialFeatures`. Se você usá-los com o Java 11, ambos os sinalizadores gerarão uma mensagem de aviso ao executar um programa Java.

Outra bandeira que se tornou obsoleta é `–XX:+AggressiveOpts`. Este sinalizador foi originalmente destinado a ativar recursos experimentais do compilador C2. Agora, esses recursos foram totalmente integrados ou removidos.

Mais uma ferramenta e uma API que gostaria de mencionar em prol da integridade é o Pack200 . O Pack200 é um esquema de compactação para arquivos JAR que foi introduzido no Java 5. O mecanismo de compactação era especialmente popular no contexto de Applets. Como mencionado anteriormente, o suporte para Applets foi descartado, tornando essa ferramenta menos útil. Por essa mesma razão, o Java o marcou como obsoleto. Para mais informações, consulte o JEP 336 .

A última, mas muito importante, seção deste relatório abrange o desempenho e a segurança.

Desempenho e Segurança

Os desenvolvedores geralmente procuram os recursos convenientes e brilhantes de uma nova versão do Java que afetam diretamente e melhoram suas rotinas diárias de trabalho. Aspectos não funcionais como desempenho e segurança, por outro lado, tornam-se extremamente importantes depois que você libera seu projeto para produção. Um aplicativo lento não conquista clientes. As brechas de segurança em seu ambiente de tempo de execução levam a uma menor confiança com seus usuários. O Java 11 também pode fornecer essa frente. Primeiro, vamos dar uma olhada nas melhorias de desempenho.

Coleta de lixo

A coleta de lixo é um recurso interno do Java responsável pela criação e destruição de objetos. Vamos começar revisando as melhorias que foram feitas no coletor de lixo padrão. Mais tarde, analisamos os novos coletores de lixo experimentais que foram adicionados nesta versão.

MELHORIAS NO COLETOR G1 PADRÃO

O Java 7 introduziu o Coletor Garbage-First (G1). Mais tarde, ele se tornou o coletor de lixo padrão com o Java 9. O G1 é otimizado para alta probabilidade e taxa de transferência. Conforme mencionado por Thomas Schatzl , o Java 11 melhora os tempos de pausa em até 60% em processadores x64 com um consumo de memória altamente reduzido. Isso é um grande ganho para qualquer pessoa pronta para atualizar para o Java 11 em sistemas de produção.

Além disso, esta versão do Java introduz dois coletores de lixo experimentais. Vamos ver o que eles podem trazer para a mesa. Nós começamos com ZGC.

ZGC: UM COLETOR DE LIXO ESCALÁVEL E DE BAIXA LATÊNCIA

A coleta de lixo é um dos recursos atraentes do Java. Como desenvolvedor, você não precisa se preocupar em marcar objetos para remoção depois de usá-los. O coletor de lixo cuida dele nos bastidores. Um problema que você pode enfrentar quando a coleta de lixo entra em ação é o tempo de pausa. Isso resulta em uma experiência ruim do cliente, pois os usuários devem aguardar a resposta do aplicativo. Os tempos de espera são ainda mais prejudiciais para aplicativos essenciais comprometidos com capacidade de resposta, alto throughput e disponibilidade com Acordos de Nível de Serviço (SLAs) acordados.

O Z Garbage Collector (ZGC) é um coletor de lixo escalável e de baixa latência . O que isso significa na prática? Embora o ZGC não elimine completamente os tempos de pausa, garante um tempo máximo de pausa de 10 ms. Além disso, ele pode lidar com pilhas de pequenos a grandes (vários terabytes). Uma outra otimização que é importante mencionar é que, com os tempos de pausa do ZGC, não aumenta o tamanho do heap. Essas condições tornam o ZGC adequado para aplicativos com muita memória que lidam com muitos dados.

Você quer tentar executar um programa em Java com o ZGC? Você pode ativar o coletor de lixo fornecendo as opções da VM - `XX:+UnlockExperimentalVMOptions –XX: +UseZGC`. O ZGC foi marcado como experimental, portanto, você precisará ser cauteloso ao usá-lo para sistemas de produção. Para mais informações, consulte a página de documentação do ZGC .

O Java 11 adiciona outro coletor de lixo experimental: o Epsilon. Na próxima seção, vamos dar uma breve olhada em seus casos de uso e critérios de desempenho.

O COLETOR DE LIXO SEM OPERAÇÃO: EPSILON

O segundo coletor de lixo experimental disponível no Java 11 é chamado de Epsilon Garbage Collector. Epsilon é um coletor de lixo com pouca sobrecarga, também conhecido como coletor de lixo não operacional. Você pode ser capaz de adivinhar o que isso significa. Não recupera memória após o espaço ter sido alocado.

Embora esse comportamento pareça um pouco estranho a princípio, definitivamente faz sentido aplicar o Epsilon em certos casos de uso. Os exemplos incluem programas de curta duração, como aplicativos de linha de comando que executam apenas um único comando e, em seguida, encerram a JVM ou cenários nos quais você deseja testar um programa.

Para ativar o Epsilon para um programa, use as opções da VM `-XX:+UnlockExperimentalVMOptions -XX:+UseEpsilonGC`. Para mais informações, consulte o documento de proposta de aprimoramento do JEP 318 .

Complementamos a discussão desta seção com uma visão geral das melhorias de segurança no Java 11.

TLS 1.3 e Suporte para Algoritmos Criptográficos

A segurança é da maior importância para qualquer sistema de software. Com a mais recente encarnação do TLS (Transport Layer Security), lançada em agosto de 2018, o desempenho do HTTPS ficou mais rápido e seguro do que nunca. O Java 11 foi capaz de incorporar o suporte para o protocolo TLS 1.3 logo de cara. É importante observar, no entanto, que o TLS 1.3 não suporta todos os recursos oferecidos pelo novo protocolo TLS. Você pode encontrar uma lista dos recursos suportados no JEP 332 .

Então, como você usa o TLS 1.3 com o Java 11? Bem, a API não mudou realmente. O Java 11 apenas adiciona novas constantes para protocolos e pacotes de criptografia. Isso torna a migração do código de aplicativo existente para a nova versão do TLS extremamente fácil.

O exemplo de código a seguir deve fornecer uma ideia aproximada sobre o uso do TLS 1.3. O código cria um `ServerSocket` para representar uma instância do servidor em execução na porta 8080. Além disso, ativa o protocolo TLS 1.3 e fornece um dos novos conjuntos de criptografia que vem com a versão mais recente da especificação TLS, conforme ilustrado aqui:

```
try {
    Socket SSLServerSocket = (SSLServerSocket)
            SSLServerSocketFactory.getDefault()
            .createServerSocket(8080);
    socket.setEnabledProtocols(new String[]
            {"TLSv1.3"});                          ❶
    socket.setEnabledCipherSuites(new String[]
            {"TLS_AES_256_GCM_SHA384"});           ❷

    try {
        new Thread(() ->
                System.out.println("Server started on port"
                + socket.getLocalPort())).start();
    } finally {
        if(socket != null && !socket.isClosed()) {
            socket.clos();
        }
    }
} catch (IOException e) {
    throw new RuntimeException(e);
}
```

❶ Ativa o protocolo TLS 1.3.

❷ Ativa um dos novos conjuntos de criptografia no TLS 1.3.

Resumo e referências para leitura adicional

Isso conclui nossa visita aos recursos e alterações mais recentes no Java 11. O Java 11 pode não vir com um "recurso fantástico", mas há boas razões para você atualizar. Você pode encontrar a maior parte do código usado neste relatório no repositório do GitHub, "O que há de novo no Java 11" , implementado como casos de teste. Se você quiser mergulhar no âmago da questão, confira a lista do JEP . O resumo a seguir fornece uma recapitulação de alto nível das mudanças oferecidas com o Java 11.

Oracle JDK e OpenJDK convergiram funcionalmente

A principal diferença é que o Oracle JDK exigirá que os usuários comprem uma licença para ambientes de produção. Java agora segue um ciclo de lançamento de seis meses para versões principais. Além disso, você pode esperar duas versões de patch entre cada versão principal para corrigir erros críticos. Java 11 é uma versão LTS. Os clientes

pagantes do Oracle JDK receberão versões adicionais menores e de correção. O OpenJDK continua sendo uma distribuição gratuita de código aberto.

Recursos mais recentes e aprimoramentos da API

Provavelmente, o maior recurso do Java 11 é a implementação do cliente HTTP, uma API para comunicação HTTP que pode atender às demandas do desenvolvimento moderno de aplicativos. Analisamos a capacidade de executar programas Java de arquivo único sem a necessidade de compilar o código. A funcionalidade é útil se você quiser testar rapidamente trechos de código Java para fins de prototipagem. A equipe Java adicionou métodos de conveniência aos módulos padrão. Como resultado, os desenvolvedores podem se livrar de bibliotecas externas que implementam métodos auxiliares semelhantes. O Java 11 também adota o Unicode 10, controle de acesso baseado em ninho, e as ferramentas de criação de perfil de código aberto Java Flight Recorder e Java Mission Control.

APIs e depreciações removidas

Há um número de APIs e módulos Java EE removidos no Java 11, e isso tem um grande impacto quando você faz upgrade de um projeto a partir de uma versão anterior do Java. Mas há outras opções para bibliotecas externas que podem ocupar seu lugar. Já faz muito tempo, mas com o Java 11, os Applets e o Java Web Start estão finalmente fora. Desacoplar o JavaFX do JDK será útil para garantir versões consistentes e periódicas do Java. Várias funcionalidades também foram marcadas para remoção em versões futuras.

Desempenho e segurança

O Java 11 tornou o coletor de lixo padrão mais eficiente e introduziu dois novos coletores de lixo experimentais: o ZGC, que é adequado para aplicativos com uso intensivo de memória, e o Epsilon, que tem como alvo programas com uma memória previsível de curto prazo. O Java 11 suporta a versão mais recente do protocolo Transport Layer Security, o TLS 1.3, tornando a comunicação HTTPS mais rápida e segura do que nunca.

Embora possa exigir algum trabalho para integrar, acho que os novos recursos, mudanças e otimizações no Java 11 valem o esforço porque facilitam um estilo de programação mais eficiente e fornecem desempenho e segurança aprimorados na produção.

Referência

What’s New in Java 11? por Benjamin Muschko - O’Reilly Media Inc. - 2019