JAVA

11 A 17

SÓ AS MELHORES PARTES

Do Java 11 ao 17:
só as melhores partes

Elder Moraes
instagram.com/eldermoraes

Premissas

Premissas

1. A abordagem será do ponto de vista de um dev enterprise
2. Sendo assim, só serão consideradas as versões LTS (11 e 17)
3. Não vai falar sobre a 18? Não, não é LTS
4. Base de comparação: Java 8 (se ainda está em uma versão anterior, atualiza)
5. Não serão cobertas funcionalidades que estejam como:
   - Preview: ainda pode mudar
   - Experimental: essas funcionalidades apresentam apenas 25% da sua implementação concluída
   - Incubator: muito parecido com a experimental, mas com um método de distribuição diferente no JDK
   - Deprecate: o gato subiu no telhado... prepare-se pra não usar mais
   - Removal: já era...

Metodologia

Metodologia

1. São 16 features na versão 11 e 13 features na 17... 29 no total
2. Então poucos minutos aqui com você, e não várias horas
3. Então, baseado na premissa “dev enterprise”, vou falar apenas sobre o que interessa a um dev enterprise
4. Marquei cada feature com a seguinte legenda:
   - S: de “sim”, interessa (vale a pena estudar)
   - N: de “não”, não interessa (estude se estiver curioso)
   - C: de “citar" (#ficaadica)

Metodologia

JAVA 11	INTERESSA?
181: Nest-Based Access Control	N
309: Dynamic Class-File Constants	N
315: Improve Aarch64 Intrinsics	N
318: Epsilon: A No-Op Garbage Collector	C
320: Remove the Java EE and CORBA Modules	C
321: HTTP Client (Standard)	S
323: Local-Variable Syntax for Lambda Parameters	N
324: Key Agreement with Curve25519 and Curve448	N
327: Unicode 10	N
328: Flight Recorder	S
329: ChaCha20 and Poly1305 Cryptographic Algorithms	N
330: Launch Single-File Source-Code Programs	S
331: Low-Overhead Heap Profiling	N
332: Transport Layer Security (TLS) 1.3	S
333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)	N
335: Deprecate the Nashorn JavaScript Engine	N
336: Deprecate the Pack200 Tools and API	N

Metodologia

JAVA 17	INTERESSA?
306: Restore Always-Strict Floating-Point Semantics	N
356: Enhanced Pseudo-Random Number Generators	S
382: New macOS Rendering Pipeline	N
391: macOS/AArch64 Port	N
398: Deprecate the Applet API for Removal	N
403: Strongly Encapsulate JDK Internals	S
406: Pattern Matching for switch (Preview)	N
407: Remove RMI Activation	N
409: Sealed Classes	C
410: Remove the Experimental AOT and JIT Compiler	C
411: Deprecate the Security Manager for Removal	C
412: Foreign Function & Memory API (Incubator)	N
414: Vector API (Second Incubator)	N
415: Context-Specific Deserialization Filters	N

Metodologia

- Vale a pena
  - 321: HTTP Client (Standard)
  - 328: Flight Recorder
  - 330: Launch Single-File Source-Code Programs
  - 332: Transport Layer Security (TLS) 1.3
  - 356: Enhanced Pseudo-Random Number Generators
  - 403: Strongly Encapsulate JDK Internals

Metodologia

- #ficaadica
  - 318: Epsilon: A No-Op Garbage Collector
  - 320: Remove the Java EE and CORBA Modules
  - 409: Sealed Classes
  - 410: Remove the Experimental AOT and JIT Compiler
  - 411: Deprecate the Security Manager for Removal

Pra quem é

Pra quem é

1. Pra quem estava procurando um *roadmap* de estudos simples, prático e direto ao ponto
2. Pra quem estava procurando argumentos pra convencer alguém de que atualizar a versão do Java é algo bom para o projeto e para o negócio

JEP 318
Epsilon: A No-Op Garbage Collector

https://openjdk.java.net/jeps/318

JEP 318: Epsilon: A No-Op Garbage Collector

- Provavelmente o GC com o menor overhead possível
- Contexto: para fazer o seu trabalho, o GC utiliza algum processamento, o que pode penalizar a performance geral da JVM
- Ele gerencia alocação de memória, mas não faz liberação de memória
- Ou seja, quando o espaço em heap acaba, a JVM pára de funcionar
- *“Eita! Pra quem um garbage collector que não collecta o garbage?”*
  - Testes de performance
  - Testes de memória
  - Testes de interface da JVM
  - Jobs com tempo de vida extremamente curtos
  - Ajustes finos de latência
  - Ajustes finos de “vazão" (thoughput)

JEP 320
Remove the Java EE and CORBA Modules

https://openjdk.java.net/jeps/320

JEP 320: Remove the Java EE and CORBA Modules

- Estes módulos já estavam como *Deprecated* desde o Java 9
- Então, se você chegou até o 11 e ainda usa... não vai usar mais (não a partir do JDK)
- Se você usa algum módulo do Java EE, é só migrar para o Jakarta EE
- Se você usa algum módulo do CORBA... *você ainda usa CORBA*???

JEP 321
HTTP Client (Standard)

https://openjdk.java.net/jeps/321

JEP 321: HTTP Client (Standard)

- A *HTTP Client API* começou como *Incubator* no JDK 9, foi atualizada no JDK 10, e virou *Release* no 11
- O objetivo é fornecer uma opção melhor à que existia até então, a *URLConnection API,* que:
  - Foi feita baseada em protocolos obsoletos (como ftp e gopher)
  - É anterior ao HTTP/1.1
  - É difícil de usar, de manter e mal documentada
  - Só funciona em blocking mode (uma thread para cada request/response)

JEP 321: HTTP Client (Standard)

- A nova API utiliza *CompletableFutures* para gerenciar requests e responses em uma abordagem *non-blocking*
- O controle de fluxo e *backpressure* é feito através da API *java.util.concurrent.Flow* (reactive-streams.org)

"Mas Elder, o que é backpressure?"

Leia:
"Backpressure explained - the resisted flow of data through software"

JEP 321: HTTP Client (Standard)

- A API nova é:
  - Mais fácil de usar
  - Mais simples de manter
  - Muito mais rastreável
  - Muito mais possibilidades de uso tanto com HTTP/1.1 quanto com HTTP/2
- Tanto o módulo quanto o package são *java.net.http*

JEP 321: HTTP Client (Standard)

Instanciando um client

```
HttpClient client = HttpClient.newBuilder()
      .version(Version.HTTP_2)
      .followRedirects(Redirect.SAME_PROTOCOL)
      .proxy(ProxySelector.of(new InetSocketAddress("www-proxy.com", 8080)))
      .authenticator(Authenticator.getDefault())
      .build();
```

JEP 321: HTTP Client (Standard)

Criando uma request

```
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("http://openjdk.java.net/"))
    .timeout(Duration.ofMinutes(1))
    .header("Content-Type", "application/json")
    .POST(BodyPublishers.ofFile(Paths.get("file.json")))
    .build()
```

JEP 321: HTTP Client (Standard)

Obtendo um response síncrono

```
HttpResponse<String> response =
    client.send(request, BodyHandlers.ofString());
System.out.println(response.statusCode());
System.out.println(response.body());
```

JEP 321: HTTP Client (Standard)

Obtendo um response assíncrono

```
client.sendAsync(request, BodyHandlers.ofString())
      .thenApply(response -> { System.out.println(response.statusCode());
                               return response; } )
      .thenApply(HttpResponse::body)
      .thenAccept(System.out::println);
```

JEP 328
Flight Recorder

https://openjdk.java.net/jeps/328

JEP 328: Flight Recorder

- É um framework para coleta de dados com baixo overhead (no máximo 1% de impacto na performance)
- Serve para fazer troubleshooting de aplicações e da própria JVM
- O que ele faz:
  - Fornece APIs para produção e consumo de dados na forma de eventos
  - Permite configuração e filtro de eventos
  - Fornece eventos para SO, Hotspot e bibliotecas do JDK
- O que ele não faz:
  - Não fornece visualização ou análise dos dados coletados
  - Não habilita a coleta de dados por default
- Os eventos registrados são gravados em um arquivo que permite a análise *after-the-fact*

JEP 328: Flight Recorder

- O JFR (Java Flight Recorder) estende as capacidades da JEP 167 (*Event-Based JVM Tracing*), que cria eventos apenas para a Hotspot
- A JEP 167 também possui um backend rudimentar, onde os dados gerados a partir dos eventos são enviados para a *stdout*
  - O JFR substitui este backend com outro de alta performance, tanto para o Java quanto para a Hotspot

JEP 328: Flight Recorder

Como iniciar o JFR

```
$ java -XX:StartFlightRecording ...
```

JEP 328: Flight Recorder

Ou através do jcmd

```
$ jcmd <pid> JFR.start
$ jcmd <pid> JFR.dump filename=recording.jfr
$ jcmd <pid> JFR.stop
```

JEP 328: Flight Recorder

Produzindo eventos (extends Event)

```
import jdk.jfr.*;


@Label("Hello World")
@Description("Helps the programmer getting started")
class HelloWorld extends Event {
  @Label("Message")
  String message;
}
```

JEP 328: Flight Recorder

Produzindo eventos (commit)

```
public static void main(String... args) throws IOException {
    HelloWorld event = new HelloWorld();
    event.message = "hello, world!";
    event.commit();
}
```

JEP 328: Flight Recorder

Extração de dados do arquivo de eventos

```
import java.nio.file.*;
import jdk.jfr.consumer.*;

Path p = Paths.get("recording.jfr");
for (RecordedEvent e : RecordingFile.readAllEvents(p)) {
  System.out.println(e.getStartTime() + " : " + e.getValue("message"));
}
```

JEP 330
Launch Single-File Source-Code Programs

https://openjdk.java.net/jeps/330

JEP 330: Launch Single-File Source-Code Programs

- Torna o *java launcher* capaz de rodar uma aplicação Java que seja composta de um único arquivo .java
- É uma situação comum durante os primeiros estágios de aprendizado da linguagem, bem como na escrita de pequenos programas utilitários
- Nestes casos, de acordo com a própria especificação, ter que compilar antes de rodar é "*pura cerimônia*"
- Ou seja, a JEP 330 torna possível fazer algo simples assim:

```
$ java HelloWorld.java
```

JEP 332
Transport Layer Security (TLS) 1.3

https://openjdk.java.net/jeps/332

JEP 332: Transport Layer Security (TLS) 1.3

- Essa é uma feature bem direto ao ponto...
- O objetivo dela é implementar o TLS 1.3, de acordo com a RFC 8446 (*Request for Comments da IETF - Internet Engineering Task Force*)
- A versão 1.3 faz uma grande revisão em termos de segurança e performance
- O *Java Security Socket Extension* (JSSE) do JDK já fornece o framework e a implementação Java tanto para o TLS, quanto para o SSL e o DTLS

JEP 356
Enhanced Pseudo-Random Number Generators

https://openjdk.java.net/jeps/356

JEP 356: Enhanced Pseudo-Random Number Generators

- Um "*Pseudo-Random Number*" é um número "*quase aleatório*"... só não é aleatório mesmo porque é baseado em um número inicial, conhecido como *seed* (esse sim, podendo ser totalmente aleatório)
- Pseudo-Random Number Generators = PRNGs
- Esta JEP:
  - Torna mais fácil o uso de vários algorítmos de PRNGs
  - Melhora o suporte a streams através do uso de streams de objetos de PRNGs
  - Preserva o comportamento da *java.util.Random*
- Introduz uma nova interface, a *RandomGenerator*, que fornece uma API uniforme para tanto para os antigos quanto novos PRNGs
- A instância de um *RandomGenerator* é feita pela classe *RandomGeneratorFactory*

JEP 356: Enhanced Pseudo-Random Number Generators

Exemplo

```
//como era
RandomGenerator rg1 = new Random(42);
//como ficou
RandomGenerator rg2 = RandomGeneratorFactory.of("Random").create(42);
//e pra chamar um novo gerador
RandomGenerator rg3 =
RandomGeneratorFactory.of("L32X64MixRandom").create(42);
```

JEP 403
Strongly Encapsulate JDK Internals

https://openjdk.java.net/jeps/403

JEP 403: Strongly Encapsulate JDK Internals

- JEP que é parte do famoso *Project Jigsaw* (não, ele não era apenas sobre modularidade)
- O principal objetivo dela é desencorajar o uso de elementos internos da JDK
- Desde o JDK 9 até o 16, o padrão utilizado para conhecido como "*relaxed strong encapsulation*", que ainda permitia o acesso dos elementos internos herdados do JDK 8 (para fins de migração)
- A partir do JDK 17, o padrão passa a ser o "*strong encapsulation*", onde ficam inacessíveis todos os elementos não-públicos de packages exportadas, e todos os elementos de packages não exportadas
- A finalidade desta mudança é avançar tanto com a manutenibilidade quanto com a segurança de código
- Alguns dos elementos internos tinham privilégios de, por exemplo, definir uma nova classe em um *class loader* específico, ou ainda, acessar dados sensíveis como chaves de criptografia

JEP 403: Strongly Encapsulate JDK Internals

- Esse encapsulamento é operado através da flag "*--illegal-access*" (nomeada assim de propósito, segundo a especificação)
  - ***--illegal-access=permit***: permite acesso a todas as packages internas existentes no JDK 8
  - ***--illegal-access=warn***: idêntico anterior, mas exibe uma mensagem de aviso a cada acesso ilegal
  - ***--illegal-access=debug***: idêntico ao *warn*, mas além da mensagem ele também lança uma stack trace
  - ***--illegal-access=deny***: desabilita todos os acessos ilegais
- O modo *deny* é o default a partir do JDK 17
- 457 packages internas foram afetadas por essa mudança, segundo esta lista: https://cr.openjdk.java.net/~mr/jigsaw/jdk8-packages-denied-by-default

JEP 409
Sealed Classes

https://openjdk.java.net/jeps/409

JEP 409: Sealed Classes

- Interfaces e classes do tipo *Sealed* restringem quais classes ou interfaces podem extendê-las ou implementá-las
  - Fornece um modo mais declarativo que os *access modifiers* para restringir o uso de uma superclass
  - Será uma grande aliada na implementação do *pattern matching* (*Project Amber* - https://openjdk.java.net/projects/amber/design-notes/patterns/pattern-matching-for-java)
- Possui grande aplicação na construção de frameworks, por exemplo (onde esse tipo de limitação é mais comum)

JEP 409: Sealed Classes

Exemplo

```
package com.example.geometry;

public abstract sealed class Shape
    permits com.example.polar.Circle,
            com.example.quad.Rectangle,
            com.example.quad.simple.Square { ... }
```

JEP 410
Remove the Experimental AOT and JIT Compiler

https://openjdk.java.net/jeps/410

JEP 410: Remove the Experimental AOT and JIT Compiler

- Desde o JDK 10 que este compilador estava presente, podendo ser acessado via flag
- É o mesmo JIT Compiler presente no Graal VM
- Foi removido no JDK 16 e ninguém percebeu!
- Deixa de fazer parte do Open JDK, mas segue firme e forte como parte do Graal VM

JEP 411
Deprecate the Security Manager for Removal

https://openjdk.java.net/jeps/411

JEP 411: Deprecate the Security Manager for Removal

- Basicamente o seguinte: se você usa o Security Manager em alguma coisa... o gato subiu no telhado
- Está no JDK desde o 1.0
- Apesar do nome, não é lá tão seguro assim...

Com esta lista de features você passa a ter...

1. Um roadmap de estudos
2. Uma lista de argumentos para encorajar o uso das últimas versões LTS no seu projeto

Obrigado!