**Introdução ao Scala e seu papel na Engenharia de Dados**

A engenharia de dados é uma área essencial no processamento e análise de grandes volumes de dados. Para lidar com essa demanda, é fundamental contar com **linguagens de programação eficientes**. No contexto da engenharia de dados, podemos utilizar diversas linguagens e entre as mais utilizadas temos Python, Java e Scala. Dentre elas, o Scala é bastante interessante e muitas big techs, como Nubank e Airbnb, utilizam para solucionar seus desafios de Big Data.

O Scala foi criado por Martin Odersky em 2001 com o objetivo de ser uma linguagem moderna e versátil. Martin queria combinar os melhores aspectos da programação orientada a objetos com os da programação funcional em uma única linguagem. O nome "Scala" vem de "Scalable Language" (linguagem escalável), pois a ideia era que a linguagem pudesse ser dimensionada para diferentes tamanhos de projetos e sistemas.



Uma das principais razões pelas quais o Scala se destaca na engenharia de dados é a sua integração com o **Apache Spark**, um dos frameworks mais populares para processamento distribuído de Big Data. O Spark foi originalmente desenvolvido em Scala e fornece uma API nativa para essa linguagem, aproveitando ao máximo seus recursos e capacidades. O Scala e o Spark se complementam de maneira eficiente, permitindo que engenheiros de dados tenham um alto desempenho para processamento e análise de dados em larga escala.

**Sintaxe Básica do Scala**

A sintaxe básica do Scala é um universo rico e abrangente, que envolve desde a definição de tipos de dados e variáveis até o domínio das estruturas de controle, a criação de funções e métodos poderosos e a manipulação de coleções de dados. Neste artigo de Scala, mergulharemos em cada um desses elementos essenciais que usamos na engenharia de dados e em outras aplicações.

**Tipos de dados e variáveis**

No Scala, os tipos de dados e variáveis são fundamentais para armazenar e manipular informações em um programa. Podemos usar uma variedade de tipos de dados básicos, como Int (para números inteiros), Double (para números de ponto flutuante), Boolean (para valores true/false) e String (para texto).

val idade: Int = 25 *// Declaração explícita de uma variável idade do tipo Int*

val nome = "João" *// Inferência automática do tipo String*

val salario: Double = 2500.50 *// Declaração explícita de uma variável salario do tipo Double*

O interessante de declarar algo em Scala é que temos dois tipos de variável var e val. Quando criamos uma variável em Scala do tipo var podemos alterar ela durante o nosso código, mas a val é imutável, se não há uma boa razão para alterar o valor das suas variáveis, devemos sempre utilizar o tipo val. Isso é uma característica da imutabilidade, uma prática recomendada em programação funcional, pois torna o código mais seguro a efeitos colaterais inesperados.

**Trabalhando com coleções (listas, conjuntos, mapas)**

Em Scala, é fundamental entender como lidar com coleções de dados, como listas, conjuntos e mapas. Essas estruturas de dados são amplamente utilizadas para armazenar e manipular informações de forma eficiente. Vamos explorar brevemente cada uma dessas coleções.

**1. Listas**

Uma lista é uma coleção ordenada de elementos, permitindo duplicatas. Ela pode ser criada com a notação List(elemento1,elemento2, …).

val lista = List(1, 2, 3, 4, 5)

*// Acessar elementos da lista*

val primeiroElemento = lista(0) *// Retorna 1*

*// Adicionar elemento à lista*

val novaLista = lista :+ 6 *// Retorna List(1, 2, 3, 4, 5, 6)*

*// Filtrar elementos pares da lista, ou seja, cujo a divisão por 2 o resto é igual a 0*

val numerosPares = lista.filter(\_ % 2 == 0) *// Retorna List(2, 4)*

Podemos aplicar uma transformação a cada elemento da lista, utilizando, por exemplo, a função .map().

*//Multiplicando cada elemento por 2*

val numerosDobrados = lista.map(\_ \* 2) *// Retorna List(2, 4, 6, 8, 10)*

**2. Conjuntos**

Ao contrário de uma lista, um conjunto é uma coleção não ordenada de elementos e não permite duplicatas. Eles podem ser criados usando a notação Set(elemento1, elemento2, ...).

val conjunto = Set(1, 2, 3, 4, 5)

*// Verificar se um elemento está presente no conjunto*

val contemTres = conjunto.contains(3) *// Retorna true*

*// Adicionar elemento ao conjunto*

val novoConjunto = conjunto + 6 *// Retorna Set(1, 2, 3, 4, 5, 6)*

*// União de conjuntos*

val outroConjunto = Set(4, 5, 6, 7, 8)

val uniao = conjunto.union(outroConjunto) *// Retorna Set(1, 2, 3, 4, 5, 6, 7, 8)*

**3. Maps**

Um "mapa" é uma coleção de pares chave-valor, onde cada chave é única. Para os programadores em Python, Map lembra bastante um dicionário. Um Map pode ser criado com a notação Map(chave1 -> valor1, chave2 -> valor2, ...).

val mapa = Map("a" -> 1, "b" -> 2, "c" -> 3)

*// Acessar valor através da chave*

val valorB = mapa("b") *// Retorna 2*

*// Adicionar novo par chave-valor ao mapa*

val mapanovo= mapa + ("d" -> 4) *// Retorna Map("a" -> 1, "b" -> 2, "c" -> 3, "d" -> 4)*

*// Remover par chave-valor do mapa*

val outroMapa = mapanovo - "c" *// Retorna Map("a" -> 1, "b" -> 2)*

**Integração com frameworks de Big Data**

O Scala é amplamente utilizado na integração com frameworks de Big Data, sendo o Apache Spark o principal exemplo. Essa combinação poderosa permite o processamento distribuído de dados em larga escala. Com o Scala, podemos expressar transformações e manipulações de dados de forma concisa, aproveitando a API do Spark. Essa integração permite realizar operações complexas, como filtragem, mapeamento e junção de dados distribuídos, possibilitando análises e manipulações eficientes.



Além disso, o Scala também é bastante utilizado em conjunto com a plataforma Databricks. O [Databricks](https://www.alura.com.br/artigos/databricks-o-que-e-para-que-serve) é um ambiente de colaboração baseado em nuvem, projetado especificamente para o desenvolvimento de aplicações de Big Data e Machine Learning. Com o Scala, os engenheiros de dados podem tirar o máximo proveito do Databricks para criar pipelines de dados avançados, explorar conjuntos de dados de maneira interativa e executar análises sofisticadas. A plataforma Databricks oferece suporte nativo ao Scala, facilitando o desenvolvimento e a implantação de soluções de engenharia de dados de alto desempenho.

Essas integrações possibilitam análises e manipulações eficientes de dados, permitindo o desenvolvimento de **soluções escaláveis na área de engenharia de dados**. Mas é somente com Scala que temos soluções eficientes? A resposta é não, podemos desenvolver soluções usando Spark com Python também, através da API PySpark, por exemplo, por isso é interessante entender as diferenças dessas duas linguagens.

**Scala vs Python: Comparando as linguagens na Engenharia de Dados**



Mesmo com ganhos de performance, poucas pessoas engenheiras de dados começam sua carreira aprendendo Scala. Para início de uma jornada em dados, a curva de aprendizado do Python é muito interessante, sendo um canivete suiço para diversas áreas de dados. Ao considerar a escolha entre Scala e Python para a engenharia de dados, é importante analisar as vantagens e desvantagens de cada uma das linguagens. Embora o Python seja amplamente adotado e conhecido por sua versatilidade, o Scala oferece benefícios específicos que podem ser valiosos para a manipulação de dados em larga escala.



**Vantagens do Scala**

* Desempenho: o Scala é executado na máquina virtual Java (JVM), o que permite um desempenho eficiente e otimizado. Isso é especialmente benéfico ao lidar com grandes volumes de dados e processamento distribuído.
* Integração com o Apache Spark: o Spark, um dos principais frameworks de processamento distribuído de Big Data, foi originalmente desenvolvido em Scala. O Scala possui uma API nativa para o Spark, facilitando a construção de aplicações escaláveis e de alto desempenho para engenharia de dados.
* Programação funcional: o Scala suporta programação funcional de forma nativa, o que é útil para lidar com transformações complexas de dados. A programação funcional pode facilitar a expressão de lógica de manipulação de dados e facilitar a manutenção e teste de código.
* Programação Orientada a Objetos: o Scala também suporta programação orientada a objetos, dando suporte aos conceitos de classes, objetos, herança etc.

**Desvantagens do Scala**

* Curva de aprendizado: o Scala pode ter uma curva de aprendizado mais íngreme em comparação com outras linguagens, devido à sua sintaxe e conceitos avançados, como programação funcional e tipos de dados avançados. Isso pode exigir um investimento de tempo e esforço para dominar a linguagem.
* Ecossistema de bibliotecas: embora o Scala tenha um conjunto sólido de bibliotecas para engenharia de dados, o ecossistema de bibliotecas pode não ser tão vasto quanto o do Python. Isso significa que pode haver menos opções prontas para uso e pode ser necessário desenvolver mais códigos personalizados em alguns casos.
* Pouca adoção: uma desvantagem do Scala para engenharia de dados é a sua menor adoção e popularidade em comparação a linguagens como Python ou Java. Embora o Scala tenha ganhado destaque na comunidade de desenvolvimento de software e engenharia de dados, especialmente devido à sua integração com o Apache Spark, seu uso ainda pode ser menos comum.

Ao considerar o uso do Scala na engenharia de dados, é importante ponderar essas vantagens e desvantagens em relação aos requisitos específicos do projeto e à experiência da equipe. O Scala pode ser uma escolha poderosa para aplicações que requerem alto desempenho, processamento distribuído e manipulação complexa de dados, desde que se esteja disposto a investir no aprendizado da linguagem e lidar com um ecossistema de bibliotecas relativamente menor. Embora possa exigir um investimento maior em aprendizado, o Scala oferece recursos essenciais para lidar com desafios complexos, por isso é uma excelente escolha para profissionais da area de Engenharia de Dados por construir soluções escaláveis e eficientes para enfrentar os constantes desafios do mundo dos dados.

Se quiser saber mais sobre essa linguagem e como ela é usada nas empresas, recomendamos esse episódio do Hipster.Tech, em que o Paulo Silveira, CEO e co-fundador da Alura, discute o assunto com o Juliano Alves, engenheiro de software na TransferWise em Londres, com a Daniela Pretuzalek, tech principal na Toughtworks, e com o Flavio Brasil, engenheiro de software no Twitter: