30. The basics of Scala. Properties. Pros and cons.

Scala была создана в начале 2000-ых годов в Федеральной политехнической школе города Лозанна (в Швейцарии). В январе 2004 года язык был выпущен для платформы JVM, а в июне того же года для .NET Framework (хотя поддержка для .NET была прекращена примерно в 2012 году)/

Scala объединяет три подхода: статическую типизацию, ООП и функциональный подход. Это позволяет разрабатывать как маленькие скрипты, так и крупные распределенные системы. Основной уклон, пожалуй, в сторону функционального программирования. Из-за этого, например, неизменяемость в языке Scala первостепенна, т.к. это одна из основ ФП.

**Плюсы:**  
Scala, как и Java, работает под JVM (Java Virtual Machine).

Scala совместим с Java-командами, поэтому если вы знаете Java, то сможете перейти и на Scala.

Scala может работать с любым кодом на Java, независимо от его сложности.

Чистейшая, незамутнённая объектно-ориентированность: даже сама программа — это один большой объект.

Сильные механизмы абстракции позволяют одинаково просто программировать большие и маленькие системы и легко масштабировать их.

Объединены объектно-ориентированный и функциональный подходы.

Scala поддерживаются конструкторы типов, типы высшего порядка и типы классов (объявляемые неявно), что серьезно упрощает работу с оборачивающими/контейнерными типами.

**Минусы:**

ФП, если эта область не знакомая, могут возникнуть вопросы. Например те же ссылочная прозрачность, чистые функции, функции первого класса и высшего порядка, лямбда-исчисление. Любопытно что это же дело можно отнести к плюсам, например за потокобезопастность.

Читать Scala-код сложнее, чем ту же самую программу на Java. Сложности опять таки возникают из-за подходов и в целом моментов ФП с его абстракциями.

Медленная компиляция — критичное место для больших проектов, когда постоянно нужно тестировать новые версии

Scala-программирование требует другого подхода к архитектуре и логике построения кода.

Плохая поддержка со стороны IDE, хотя это спорное утверждение, т.к. с той же идеей от жетбрейнс вполне себе было удобно.

Проблемы при использовании сторонних библиотек. Например, для работы Спарка требовалась определенная версия Скалы и кита разработки соответственно.

31. The basics of Scala. Mutability and Immutability.

Неизменяемые объекты и структуры данных - первоочередные граждане Scala. Это потому, что они предотвращают ошибки в распределенных системах и обеспечивают потокобезопасные данные. Однако мы также можем использовать изменяемые объекты, если это действительно необходимо.

В Scala, когда значение присваивается неизменной переменной, мы не можем переназначить его. Если бы мы попытались, это привело бы к ошибке компиляции. С другой стороны, мы можем переназначить значение изменяемой переменной.

Объявления в коде: val – для неизменяемой, var – для изменяемой.

П.

**class** **ImmutabilityCar**(color: **String**, val wheels: **Int**, var engine: **String**)  
**val** myCar = **new** **ImmutabilityCar**("blue", 4, "diesel")

myCar.wheels = 5

При попытке изменить колеса упадет ошибка компиляции

Для коллекций ситуация похожа.

Неизменяемость означает, что объект коллекции никогда не изменяется; вместо этого вы создаете новые объекты коллекции с помощью таких операций, как + или ++, которые возвращают новую коллекцию.

**val** myPets = pets :+ "Hamster"

**val** notPets = pets ++ **List**("Giraffe", "Elephant")

**val** yourPets = pets.updated(0, "Mice")

Все 3 варианта создадут новые коллекции

Изменяемый означает, что вы можете изменять коллекции на месте. Пример на коллекции: если у вас есть коллекция *С*, и вы добавляете элемент с помощью + =, то *С* изменился, как и все остальные ссылки на эту коллекцию. Чтобы создать изменяемую коллекцию, нам нужно импортировать из scala.collection.mutable.

**import** scala.collection.mutable.**ArrayBuffer**

**val** breakfasts = **ArrayBuffer**("Sandwich", "Salad")

Мы можем добавлять, обновлять или удалять элементы из изменяемой коллекции.

В некоторых изменяемых коллекциях, таких как Array, мы не можем добавлять или удалять элемент. Мы можем обновлять только элементы в коллекции.

В функциональном программировании есть понятие ссылочной прозрачности.

Выражение является ссылочно-прозрачным, если его можно заменить его значением без изменения поведения программы. Этого можно достичь, используя неизменяемые объекты. И именно поэтому неизменяемые объекты являются первоочередными гражданами Scala.

С неизменяемыми объектами мы можем безопасно использовать параллелизм. Это связано с тем, что нет опасности одновременного доступа нескольких потоков к одним и тем же данным, если их нельзя изменить.

Однако использование неизменяемых объектов и коллекций может вызвать проблемы с производительностью. Создание новой копии коллекции для добавления элемента менее эффективно, чем изменение существующей коллекции.

Поэтому там, где нам нужно, мы можем вместо этого использовать изменяемые объекты.

32. The basics of Scala. Methods on collections.

Методы при работе с коллекциями

Для примеров используем коллекцию

val a = "foo bar baz"

1) Методы фильтрации

Это те методы, которые фильтруют исходную коллекцию в подмножество исходной коллекции без изменения каких-либо данных в процессе фильтрации.

Таким образом, метод фильтрации создаст подмножество исходной коллекции без изменения каких-либо элементов с помощью алгоритма (например, умножения каждого элемента на два, например). Методы:

* distinct
* drop
* dropRight
* dropWhile
* filter
* filterNot
* find
* head
* headOption
* init
* last
* lastOption
* slice
* tail
* take
* takeRight
* takeWhile
* toSet
* withFilter

Примеры:

a.distinct # "fo barz"

a.drop(4) # "bar baz"

a.head # f

a.takeRight(3) # baz

2) Трансформаторные и/или редукционные методы

Методы преобразователя принимают по крайней мере одну коллекцию входных данных и применяют к этой коллекции пользовательский алгоритм для создания новой выходной коллекции. Некоторые из этих методов также могут уменьшить исходную последовательную коллекцию до одного (или нескольких) производных значений. Методы:

* +
* ++
* −
* −−
* collect
* diff
* distinct
* flatMap
* map
* reverse
* sortWith
* takeWhile
* zip
* zipWithIndex

Примеры:

a.map(\_.toUpper) # FOO BAR BAZ

a.zip(0 to 10) # Vector((f,10), (o,11), (o,12), ( ,13), (b,14), (a,15), (r,16), ( ,17), (b,18), (a,19), (z,20))

3) Методы сортировки

Как следует из названия, методы сортировки позволяют вам создать новую последовательность из вашей начальной последовательности, в которой новая последовательность была отсортирована:

* sortBy
* sortWith
* sorted

Примеры:

a.sortBy # this is a bit long; see below

a.sortWith(\_ < \_) # " aabbfoorz"

a.sortWith(\_ > \_) # "zroofbbaa "

a.sorted # " aabbfoorz"

4) Методы группировки

Эти методы позволяют вам взять существующую коллекцию и создать несколько групп из этой одной входной коллекции. Методы:

* groupBy
* grouped
* partition
* sliding
* span
* splitAt
* unzip

Примеры:

firstTen.groupBy(\_ > 5)         # Map(false -> List(1, 2, 3, 4, 5), true -> List(6, 7, 8, 9, 10))

firstTen.grouped(2)             # Iterator[List[Int]] = non-empty iterator

firstTen.grouped(2).toList      # List(List(1, 2), List(3, 4), List(5, 6), List(7, 8), List(9, 10))

5) Информационно-математические методы.

Эти методы предоставляют информацию о коллекции:

* canEqual
* contains
* containsSlice
* count
* endsWith
* exists
* find
* forAll
* hasDefiniteSize
* indexOf
* indexOfSlice
* indexWhere
* isDefinedAt
* isEmpty
* lastIndexOf
* lastIndexOfSlice
* lastIndexWhere
* max
* min
* nonEmpty
* product
* segmentLength
* size
* startsWith
* sum

Примеры

a.count(\_ == 'a') # 2

a.endsWith("baz") # true

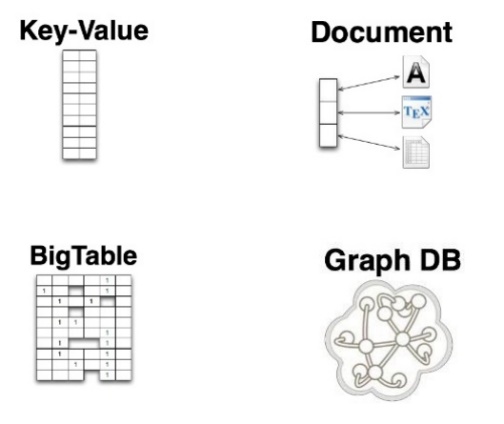
6) Другие

Несколько других методов сложнее классифицировать, в том числе:

* flatten - преобразует список списков в один список
* foreach - как цикл for, позволяющий перебирать элементы в коллекции
* mkString - позволяет создать строку из коллекции
* par - создает параллельную коллекцию из существующей коллекции
* view - создает ленивое представление для коллекции (см. рецепт 10.24)
* a.par # a parallel array, ParArray(f, o, o, , b, a, r, , b, a, z)

33. Classification and differences of NoSQL storages.

NoSQL использует разные хранилища данных для оптимизации для конкретных целей в зависимости от требований к системе, на рисунке 4 категории хранилищ данных NoSQL:



**1. key-value store - Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB**

Сохраняет данные (значения) и применяет к ним метку (ключ) и сохраняет их либо в памяти, либо в системе хранения, оптимизированной для быстрого извлечения данных.

Значение обычно можно получить, только сославшись на его ключ, поэтому научиться запрашивать конкретную пару ключ-значение обычно просто. Базы данных типа "ключ-значение" отлично подходят для случаев, когда вам нужно хранить большие объемы данных, но вам не нужно выполнять сложные запросы для их получения.

**2. document store - CouchDB, Couchbase, MongoDB, Berkeley DB XML**

Предназначен для хранения, поиска и управления документальной информацией. Хранилища документов сохраняют всю информацию об объекте как экземпляр в базе данных, а не распределяют данные по разным таблицам.

Хранит данные в документах, подобных объектам JSON (JavaScript Object Notation). Каждый документ содержит пары полей и значений. Значения обычно могут быть различных типов, включая такие вещи, как строки, числа, логические значения, массивы или объекты, и их структуры обычно совпадают с объектами, с которыми разработчики работают в коде.

Может масштабироваться по горизонтали для размещения больших объемов данных.

**3. wide column database - Apache HBase, Apache Cassandra, Apache Accumulo, Amazon SimpleDB**

Использует таблицы, строки и динамические столбцы, отличные от реляционных баз данных, поскольку для каждой строки не требуется один и тот же столбец, что позволяет горизонтальное масштабирование базы данных. Пример использования - хранение профилей пользователей.

wide column database (Хранилища с широкими столбцами обеспечивают большую гибкость по сравнению с реляционными базами данных, поскольку не обязательно, чтобы каждая строка имела одинаковые столбцы. Хранилища с широкими столбцами отлично подходят, когда вам нужно хранить большие объемы данных и вы можете предсказать, какими будут ваши шаблоны запросов. Хранилища с широкими столбцами обычно используются для хранения данных Интернета вещей и данных профилей пользователей.

**4. graph database - Neo4j, MarkLogic, Ontotext, OrientDB**

Хранит данные в виде узлов и ребер. Узлы хранят данные объекта, а ребра хранят информацию о взаимосвязи узлов. Используется для просмотра взаимосвязей и закономерностей в данных. Варианты использования - социальные сети, искусственный интеллект и механизм рекомендаций.

Graph отлично подходят для тех случаев, когда вам нужно изучить отношения для поиска паттернов, таких как социальные сети, обнаружение мошенничества и механизмы рекомендаций.

В таблице показаны типы баз данных NoSQL и связанные с ними функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тип/Класс** | **Производительность** | **Масштабируемость** | **Гибкость** | **Сложность** |
| Document Store | 4 | 2 | 4 | 1 |
| Column Store | 4 | 4 | 3 |  |
| Kev value store | 4 | 4 | 4 | 4 |
| Graph-based | 2 | 2 | 4 | 4 |

**34. Big data use-cases.**

**Большие данные** - Группа технологий и методов для эффективной обработки растущих объемов данных (структурированных и неструктурированных) в распределенных информационных системах.

Это наборы данных такого размера, что традиционные инструменты не могут принимать, хранить и обрабатывать их в течение приемлемого времени.Примеры использования больших данных.

*Case1: Cartao Elo.*

Единственная в Бразилии компания, выпускающая кредитные карты, обрабатывает более миллион транзакций в день.

Задача: Elo стремится повысить качество обслуживания потребителей с помощью предоставления персонализированных, контекстных предложений – до того как покупатели поучат свою кредитную карту

Решение: Cloudera Enterprise (Apache Sentry, Apache Impala, Cloudera Navigator) + База данных Oracle. 7Тб сегодня, рост: 10Гб в день. Путем сбора данных, не связанных с транзакциями из социальных сетей/каналов, торговых маяков и других новых источников. Elo оснащено для предоставления клиентам личных, контекстно-зависимых связей/коммуникаций.

*Case 2: Bank Mandiri*

Крупнейший банк Индонезии - компания стремится быть не просто самый большим, но лучшим банком в Индонезии.

Задача: Mandiri осознали необходимость модернизации своей инфраструктуры с помощью более экономичных технологий, которые улучшили бы качество обслуживания клиентов. Их предыдущая среда состояла из 27 различных систем, которые были дорогостоящими и жесткими, которым было сложно получить неструктурированные данные из таких источников, как социальные сети.

Решение полагается на хранилище данных Cloudera (13 ТБ) для поддержания своего конкурентного преимущества, облегчая предоставление управляемое данными , персонализированное обслуживание клиентов при снижении затрат на ИТ-инфраструктуру на 99 процентов.

Case #3: Experian

Experian входит в тройку крупнейших мировых кредитных бюро. Хранит и обрабатывает информацию о кредитных историях ~ 1 миллиарда заемщиков-физических лиц

Задача: клиенты стали чаще запрашивать обновленную информацию о последних покупательских привычках, моделях просмотра в Интернете и активности в социальных сетях, чтобы они могли реагировать в режиме реального времени.

Решение: Использование Cloudera Enterprise с Hadoop, Hbase и Hive. В результате производительность обработки увеличилась в 50 раз. Теперь они могут обрабатывать 100 миллионов записей в час с помощью новой среды Hadoop, тогда как раньше они обрабатывали 50 миллионов совпадений в день.

Case #4: Transamerica

Группа компаний по страхованию жизни и управлению активами

Задача: Transamerica имеет несколько направлений бизнеса в разных регионах. Из-за этого один клиент мог появиться в нескольких системах, причем каждая система собирала разную информацию.

Решение: использует Cloudera Enterprise (Hive, Impala, Spark) для разделения разрозненных данных и создания единого комплексного представления как неструктурированных, так и структурированных данных, включая данные профиля клиента, данные о потенциальных клиентах и ​​партнерах, историю запросов и веб-журналы. Масштаб данных около 50 ТБ.

Case #5: mBank

Один из ведущих польских финансовых институтов, входящий в группу Commerzbank. Обслуживание более 4,1 миллиона розничных клиентов в Польше и более 800 000 розничных клиентов в Чехии и Словакии.

Проблема: лежащие в основе унаследованные ИТ-системы, которые больше не могли справляться с постоянно растущими объемами данных

Решение: построение современной ИТ-инфраструктуры с Cloudera Enterprise, которая позволяет интегрировать данные и быстро заполнять свои хранилища данных. В результате сократились ежедневные процессы интеграции данных на 66 процентов. Ускорение выполнения запросов в 4 раза.

Case №7: Сбер

Использует большие данные и машинное обучение в кредитном рейтинге - не только традиционные данные (социально-демографические параметры, кредитная история, история транзакций, финансовая отчетность), но и графики взаимоотношений с клиентами на основе данных о денежных переводах и данных социальных сетей

В 2015 году компания добавила в модели данные от сотовых операторов, что улучшило качество классификатора.

Большое количество активных SIM-карт и малое время их работы, небольшие и многочисленные платежи по счетам, странная география звонков указывают на мошенничество и снижают вероятность одобрения кредита

Новостные тексты с автоматическим анализом настроений также используются в различных моделях.

Результат: постоянный рост качества скоринговых моделей.

Case #8: Western Union

Крупнейший оператор денежных переводов на международном рынке

Задача: в 2013 году Western Union выполняла в среднем 29 транзакций в секунду; каждая транзакция генерирует данные как об отправителях, так и о получателях

Решение: интеграция Cloudera Enterprise Data Hub с Apache Flume и Apache Sqoop, который передает структурированные данные из нескольких хранилищ данных, а также неструктурированные данные, включая потоки кликов, поведенческие данные, журналы и данные настроений, собираемые такими инструментами, как транзакционные, маркетинговые и другие информационные средства системы.

Case #9: MasterCard

MasterCard - платежная система с 22 тысячами финансовых учреждений в 210 странах мира.

Задача: MasterCard должны управлять большим объемом, скоростью и разнообразием, чем большинство компаний.

Решение: Cloudera Enterprise + Cloudera Search помогает быстро и легко выявлять продавцов-мошенников и снижать риски. Платформа распределенного хранения и обработки данных (CDH) была интегрирована для динамического масштабирования и управления нагрузкой и сложностью алгоритмов поиска.

Case #10: NYSE - ICE

Межконтинентальная биржа (ICE) - управляет ведущей сетью мировых бирж фьючерсов, акций и опционов на акции. NYSE - мировой лидер в области привлечения капитала, листинга и торговли акциями.

Задача: масштабирование для обработки и анализа огромных потоков данных.

Решение: Cloudera Enterprise Data Hub (Apache Impala, Apache Spark, Cloudera Navigator) решила проблемы организации с данными. Новый центр корпоративных данных поддерживает обработку и аналитику более 20 ПБ данных, при этом ежедневно добавляется 30 ТБ свежих данных.