大数据 - 数据与价值

大数据是指无法在合理时间内用常规软件工具进行捕获、管理和处理的庞大和复杂数据集。

这一概念不仅涉及数据的数量，还涉及数据的多样性、速度和价值。

大数据的“4V”特征

1 体量（Volume）

数据量巨大，从TB（太字节）到PB（拍字节）乃至更多。

2 多样性（Variety）

数据类型多样，包括结构化数据、半结构化数据和非结构化数据。

3 速度（Velocity）

数据流入的速度极快，需要实时或近实时处理。

4 价值（Value）

数据潜在的价值高，但需要通过分析挖掘。

大数据的应用领域

- **商业智能和营销**：通过分析消费者行为数据，优化营销策略，提高客户满意度和忠诚度。
- **金融服务**：用于风险管理、欺诈检测、客户数据管理和算法交易。
- **医疗保健**：提高疾病诊断的准确性，优化治疗方案，进行个性化医疗。
- **智慧城市**：提高城市管理的效率和居民的生活质量。
- **物联网（IoT）**：可用于智能家居、工业自动化、环境监测等领域。

就业方向

1. **数据分析师**：负责分析和解释复杂的数据集，帮助企业做出基于数据的决策。
2. **数据科学家**：使用统计学、机器学习和数据分析技术来分析数据并提取洞察力，以解决复杂的业务问题。

3. **大数据工程师**：设计、构建和维护大规模数据处理系统。他们负责开发、测试和优化大数据解决方案。
4. **机器学习工程师**：专注于使用机器学习算法和工具来处理和分析大数据。
5. **商业智能（BI）开发者**：利用大数据工具和软件来开发分析报告，为企业提供决策支持。

数据存储技术

HDFS（Hadoop分布式文件系统）

HDFS是Apache Hadoop的核心组成部分，专为大规模的数据存储和分析而设计。

对象存储解决方案

对象存储是一种存储数据的方法，将数据及相关的元数据封装在一起。

云平台上的大数据存储服务

云平台上的大数据存储服务为用户提供了灵活、可扩展的存储解决方案。

HDFS（Hadoop分布式文件系统）

概念

HDFS是一个分布式文件系统，专为在普通硬件上运行而设计，提供高吞吐量的数据访问，适用于大数据集的应用。它是Apache Hadoop项目的一部分。

特点

- 容错性：通过在多个节点上复制数据来提高容错性，即使部分节点失败，数据也不会丢失。
- 高吞吐量：优化大批量数据的读写操作，特别适合大规模数据处理任务。
- 可扩展性：能够存储PB级别的数据，通过增加更多节点来扩展存储能力。

适用场景

适用于大规模数据处理项目，如大数据分析、数据仓库。

对象存储解决方案

概念

对象存储是一种存储架构，以对象的形式存储数据，每个对象包含数据、元数据和全局唯一标识符。对象可以在任何位置存储，易于扩展。

特点

- 无目录层次：数据作为对象存储，取消了传统文件系统的目录结构，简化了数据访问。
- 元数据丰富：每个对象包含可扩展的元数据，提高了数据管理的灵活性和效率。
- 可扩展性：非常适合云环境，可以无限扩展，支持海量数据存储。

适用场景

适用于需要存储大量非结构化数据的场景，如图片、视频存储、大规模备份和归档。

云平台上的大数据存储服务

概念

云平台提供的大数据存储服务，如Amazon S3、Google Cloud Storage，提供了可扩展、高可用性和安全性的数据存储解决方案。

特点

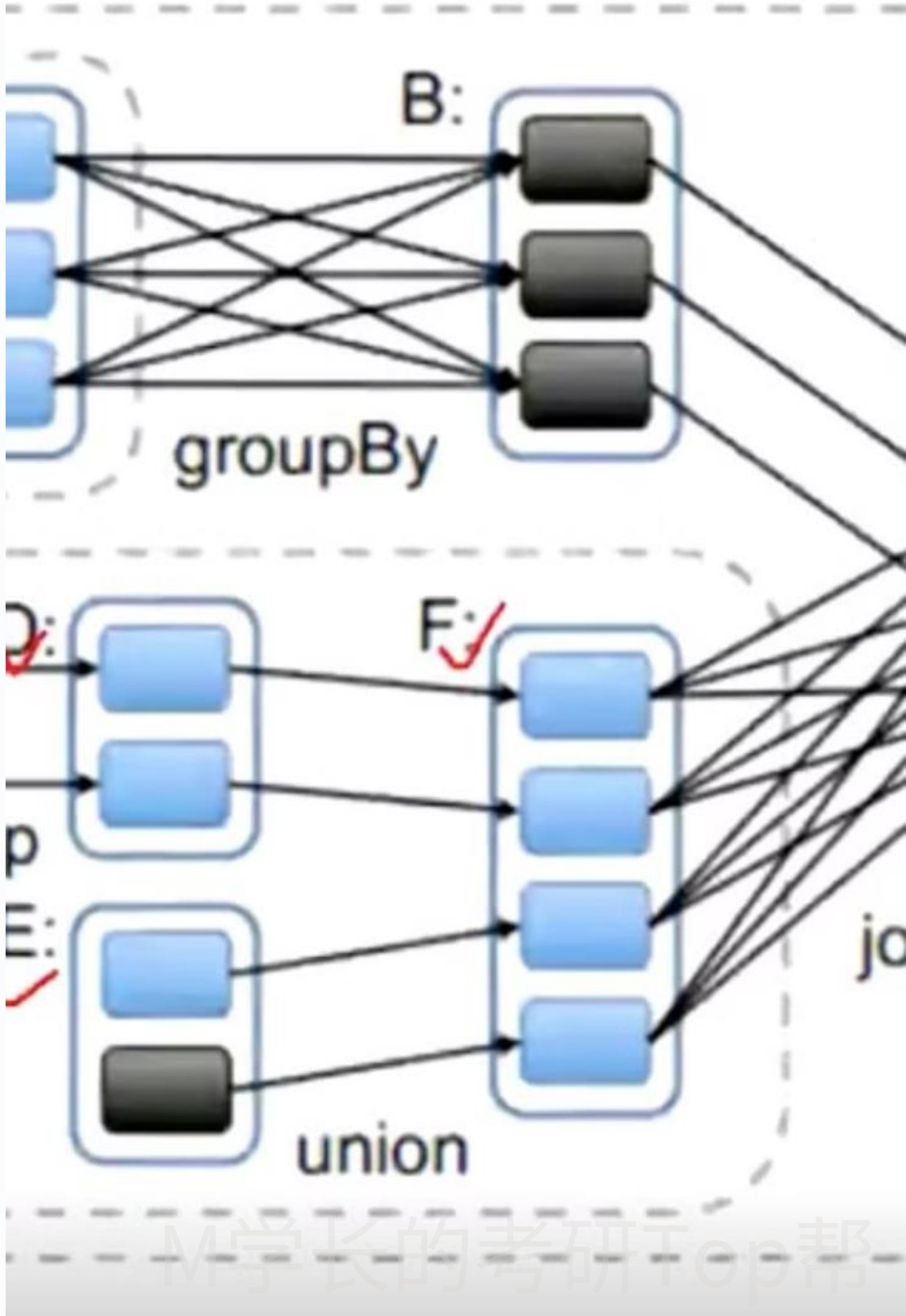
- 弹性扩展：根据需求自动增减存储容量，支付所用即所得。
- 数据安全和隐私保护：提供数据加密、访问控制和持久性保证。
- 全球访问：数据可以存储在全球多个地区，优化访问速度和可靠性。

适用场景

适用于所有规模的企业，特别是需要灵活管理成本、实现全球部署的大数据项目。

数据处理框架

数据处理框架指的是在大数据环境下，用于处理和管理数据的架构和工具集合。这些框架可以包括数据的提取、转换、加载（ETL），数据仓库，数据湖等。它们为企业提供了有效地收集、存储、处理和分析海量数据的能力。



Spark

- **概念：** Spark是一个开源的大数据处理框架，提供了强大的数据处理能力，支持批处理、流处理、机器学习和图计算等。
- **特点：**
  - **内存计算：** 将数据加载到内存中进行计算，大幅提高处理速度。
  - **灵活的数据处理：** 支持多种数据处理模式，易于开发复杂的大数据应用。
- **适用场景：** 适用于需要快速数据处理和分析的应用，如实时分析、机器学习项目。

Flink

概念：Flink是一个开源的流处理框架，也支持批处理，以数据流为中心，提供高吞吐量、低延迟的数据处理能力。

特点：

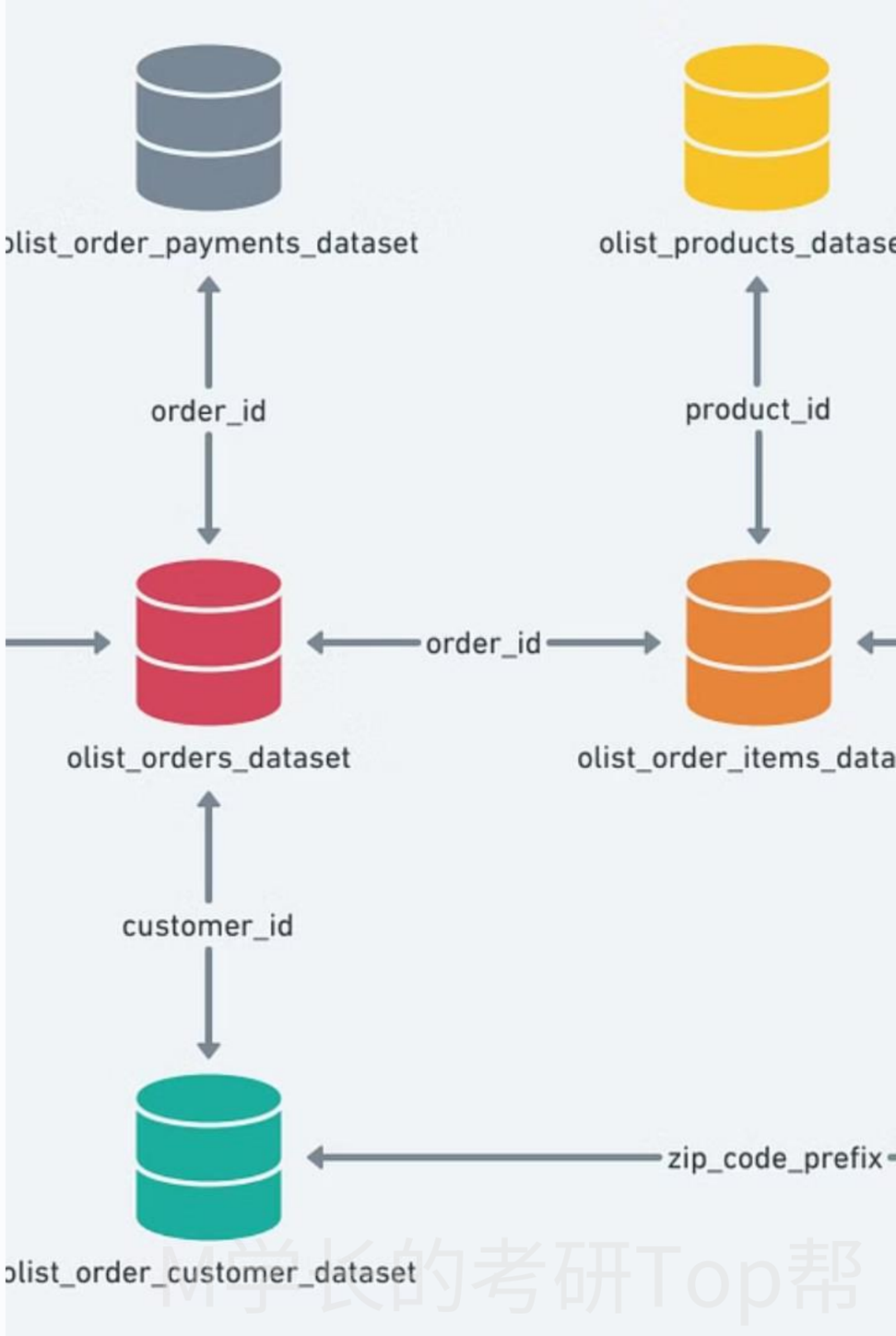
- **真正的流处理**：以流为本质，提供精确的事件时间处理和状态管理。
- **高性能**：优化的执行引擎，支持高并发和低延迟处理。

适用场景：特别适合需要连续不断处理数据流的应用，如实时监控、事件驱动应用。

大数据数据库

大数据数据库是指设计用于处理和存储庞大、快速增长数据集的数据库系统。

它们能够有效管理和处理大规模数据集，支撑现代数据密集型应用。



大数据数据库的关键特点

- **高性能：** 优化查询执行计划、减少数据读取延迟、提高数据处理的并行性。
- **可扩展性：** 通过水平扩展来增加处理能力，简单增加更多的服务器节点。
- **容错与高可用性：** 采用分布式架构，保持数据的完整性和服务的可用性。

传统的数据库 RDBMS

Relational Database Management System中文翻译为“关系型数据库管理系统”。它是一种用于存储、管理和检索数据的软件系统，其核心设计基于关系模型。在关系模型中，数据被组织成一系列相互关联的表格（或称关系），这些表格通过预定义的键（通常是主键和外键）来建立联系。

关系型数据库管理系统使用结构化查询语言（SQL）作为主要的数据访问和管理接口，允许用户创建、更新、查询和删除数据库中的数据，并且能够保证数据的一致性、完整性和可靠性。RDBMS通常支持事务处理、并发控制以及数据备份与恢复等功能，确保数据在多用户环境下的安全性与稳定性。

常见的RDBMS产品包括Oracle、Microsoft SQL Server、MySQL、PostgreSQL、IBM DB2等。

RDBMS的问题

然而，尽管关系型数据库管理系统提供了许多优势，但也存在一些问题。首先，对于大规模数据的处理，RDBMS可能会遇到性能瓶颈。

局限性

- **扩展性限制**：RDBMS通常依赖垂直扩展，即通过提升单台服务器硬件性能应对数据增长。然而，在大数据场景下，数据量和访问量增速远超单一服务器承载能力，数据表过大时，查询和写入操作显著变慢。大数据应用则需要水平扩展，将数据分布于多台机器以实现并行处理
- **表结构灵活性不足**：RDBMS中的数据模型预定义严格，面对复杂、多变的大数据时缺乏足够的灵活性。尤其当大数据包含半结构化或非结构化数据时，关系模型的匹配度有限。频繁变动的业务需求下，表设计和管理变得复杂。

局限性

- **查询性能挑战**：海量数据查询可能导致RDBMS响应时间过长，特别是在全表扫描和复杂JOIN操作中，且频繁磁盘I/O影响效率。相比之下，Hadoop、Spark等大数据处理框架结合分布式计算模型能更高效地进行批量数据分析。
- **事务与一致性要求**：虽然RDBMS坚持ACID特性保证交易处理的一致性，但在大数据环境下，强一致性的代价可能过高。很多大数据应用可接受最终一致性，部分NoSQL系统因此牺牲了一定的一致性，换取更好的可用性和扩展性。

大数据数据库

NoSQL数据库

NoSQL 数据库 是一种 **非关系型数据库**，它不同于传统的关系型数据库（RDBMS），更灵活地存储和管理大规模的非结构化或半结构化数据。NoSQL 不使用固定的表结构、行和列，而是根据需求采用不同的数据模型，如键值、文档、列族或图数据库。

"NoSQL" 最初意为 "Not Only SQL"，强调它不仅限于 SQL 的使用，旨在应对互联网时代的大数据、高并发等需求。

NoSQL 数据库的特点

- **灵活的模式（Schema-less）**：NoSQL 数据库通常没有固定的表结构，可以灵活地存储各种格式的数据（JSON、XML、二进制等）。
- **高扩展性（Scalability）**：支持 横向扩展（Scale Out），通过增加节点来提升存储和计算能力。相较于 RDBMS 的纵向扩展（升级硬件），成本更低。
- **高性能（High Performance）**：通过去掉关系型数据库的复杂查询和事务功能，优化了高并发读写操作的性能。
- **高可用性（High Availability）**：许多 NoSQL 数据库内置分布式架构，具有容错机制和自动副本同步功能。
- **多种数据模型**：NoSQL 提供多种数据存储模型，适应不同的业务场景，如键值对、文档、列族和图结构。

NoSQL 数据库的分类

键值型数据库

键值型数据库以键值对的形式存储数据，就像哈希表一样。

它非常适合需要快速查询简单数据关系的场景，例如缓存和会话数据存储。

文档型数据库

文档型数据库以文档为单位存储数据，通常使用 JSON 或 BSON 格式。

它擅长存储结构化或半结构化数据，数据之间关系较弱。

列族型数据库

列族型数据库以列为中心存储数据，支持动态列定义。

它适合需要高效写入和分布式查询的场景，例如日志存储和时间序列数据。

图型数据库

图型数据库使用图结构存储数据，节点表示实体，边表示实体之间的关系。

它适合处理复杂关系的场景，例如社交网络、推荐系统和路径计算。

JSON

JSON是一种轻量级的数据交换格式，它以易于阅读和编写的文本格式来表示结构化数据。它由键值对组成，键和值之间用冒号分隔，键值对之间用逗号分隔，整个结构包括在花括号中。JSON被广泛应用于Web应用程序和大数据处理中。

```
{
  "name": "张三",
  "age": 30,
  "isStudent": false,
  "courses": ["语文", "数学", "英语"],
  "address": {
    "city": "北京",
    "street": "中关村南大街"
  }
}
```

NoSQL 的优势

- **大规模数据支持**: NoSQL 数据库可以轻松处理海量数据， 适合 PB 级别的存储需求。
- **灵活性和高并发**: NoSQL 数据库提供灵活的数据模型和高并发读写能力， 可以满足多种业务场景的需求。
- **高可用性和高性能**: NoSQL 数据库通过分布式架构和优化设计， 实现了高可用性和高性能。

NoSQL 的局限性

- 大多数 NoSQL 数据库不支持关系型数据库的强 ACID 特性，事务支持有限。
- 没有统一的查询语言，不同数据库需要学习不同的接口和操作方法。
- 不适合需要多表 JOIN 或复杂 SQL 查询的场景。
- 许多 NoSQL 数据库更倾向于可用性和分区容错性，而在一致性上做出妥协。
- 相比 RDBMS 的成熟生态，NoSQL 的工具链和生态相对较少。

NoSQL 的适用场景

高并发访问

网站的会话管理、用户信息缓存等高并发需求场景。

示例：Redis 用于处理短时高频访问的缓存。

海量数据存储

日志、传感器数据、时间序列数据等需要海量存储的场景。

示例：Cassandra 用于分布式日志存储。

灵活的业务需求

数据模式经常变化的场景，如快速迭代中的互联网应用。

示例：MongoDB 用于动态结构的电商商品信息管理。

复杂关系查询

处理社交网络中的好友关系、推荐系统中的商品关系。

示例：Neo4j 用于社交图谱和路径优化。

NewSQL数据库

NewSQL数据库试图结合关系型数据库和NoSQL数据库的优点，旨在提供一种同时具有强一致性、高事务性和良好扩展性的数据库解决方案。

NewSQL数据库

特点

- 事务支持：提供ACID（原子性、一致性、隔离性、持久性）事务支持，确保数据的准确性和一致性。
- SQL支持：支持标准SQL查询，方便开发者利用现有的数据库知识和工具。

分布式架构

采用分布式架构设计，通过数据分片和复制实现数据库的水平扩展和高可用性。

典型技术：如Google Spanner利用全球同步的原子钟技术解决了分布式系统中的一致性问题，支持跨多个数据中心的强一致性事务。

缺点

复杂度较高：为了实现分布式事务和强一致性，NewSQL数据库通常采用了更为复杂的内部机制，可能导致运维难度增大。

资源消耗较大：为了保证强一致性和高可用性，可能会增加硬件资源的需求和网络开销。

NewSQL 与 RDBMS、NoSQL 的对比

特性	NewSQL	RDBMS	NoSQL
查询语言	SQL（标准）	SQL（标准）	不支持标准化（键值、文档查询等）
事务支持	ACID	ACID	通常为 BASE 理论（最终一致性）
扩展性	水平扩展（Scale Out）	纵向扩展（Scale Up）	水平扩展（Scale Out）
性能	优化了高并发和大数据	高性能，但扩展性有限	高并发和高扩展性
数据模型	表格（关系型）	表格（关系型）	灵活（键值、文档、列族等）
适用场景	高并发、高一致性、事务处理	复杂查询、多表关联	海量数据、灵活结构、实时流处理

MongoDB简介

MongoDB 是一种开源的、基于文档的 **NoSQL 数据库**，以其灵活性、高性能和易用性著称。它使用类似 **JSON** 的 **BSON（二进制 JSON）** 格式存储数据，适合处理大规模、半结构化或非结构化的数据。

MongoDB是目前 NoSQL 数据库中最受欢迎的选择之一。

MongoDB 的核心特点

- MongoDB 使用 BSON 格式存储数据，每个文档可以包含嵌套的字段和数组，类似 JSON 文件。
- MongoDB 具有模式灵活性（Schema-less），无需预定义表结构，方便开发者处理各种类型的数据。
- MongoDB 提供快速的读写性能，支持内存中计算（in-memory computing），加速数据访问。
- MongoDB 内置分片（Sharding）支持，可轻松实现横向扩展（Scale Out）。
- MongoDB 具备副本集（Replica Set）功能，提供数据的高可用性和容错性。

MongoDB 的核心概念

1. **数据库（Database）**：
   - 是 MongoDB 中的顶层容器，一个数据库可以包含多个集合。
2. **集合（Collection）**：
   - 类似于关系型数据库中的表，用于存储文档，但没有固定的模式。
   - 例如，一个 `users` 集合可以存储不同结构的用户数据。
3. **文档（Document）**：
   - MongoDB 的基本存储单元，使用 BSON 格式。如下

```
{
    "name": "Alice",
    "age": 25,
    "hobbies": ["reading", "traveling"]
}
```

MongoDB 的核心概念

- **字段（Field）**：
- 文档中的键值对，类似关系型数据库中的列。
- **副本集（Replica Set）**：
- 一个主节点（Primary）和多个从节点（Secondary），实现数据高可用性和容灾。
- **分片（Sharding）**：
- 将数据分布在多个节点上以实现横向扩展，通过分片键（Shard Key）确定数据分布位置。
- **聚合管道（Aggregation Pipeline）**：
- 一种数据处理框架，支持复杂的数据转换和聚合操作。

MongoDB 的主要功能

CRUD操作：Create：插入文档到集合。 Read：查询文档， 支持条件查询、排序、分页。 Update：更新文档，支持部分字段更新。 Delete：删除文档或集合。

索引：单字段索引、多字段复合索引、地理空间索引、全文检索索引。

聚合框架：通过聚合管道实现数据过滤、分组、排序和统计等操作。

分布式特性：支持数据分片和副本集，提供高可用性和扩展性。

事务支持：提供多文档事务支持， 确保数据一致性。

MongoDB的优势

1 高性能

MongoDB提供高性能的数据读写操作，特别是对于复杂查询和大数据量的处理。通过有效的索引策略和分片技术，MongoDB能够提供快速的查询响应和高吞吐量。

2 高可用性

通过副本集的机制，MongoDB能够实现自动故障转移和数据备份，保证数据的高可用性和一致性。

3 易于扩展

MongoDB的分片功能使得它可以水平扩展，支持大规模的数据集和高并发的访问，非常适合云计算和大数据的应用场景。

4 灵活的数据模型

MongoDB的文档数据模型提供了极大的灵活性，可以轻松应对数据模式的变化，非常适合快速迭代开发和多变的应用需求。

MongoDB 的局限性

- MongoDB 更关注可用性和分区容错性，但对强一致性需求的场景可能需要额外配置事务。
- 对于关系高度复杂的场景，MongoDB 不如关系型数据库方便。
- MongoDB 默认会使用大量的内存和磁盘空间，需要硬件支持。
- 对于习惯了 SQL 的开发者，需要一定时间适应其查询语言和架构设计。

MongoDB的应用场景

- **Web和移动应用**： MongoDB的灵活性和扩展性使其成为构建现代Web和移动应用的理想选择。
- **大数据分析**： MongoDB的动态模式、强大的聚合框架和索引支持使其非常适合处理大数据分析场景。
- **内容管理系统**： 对于内容管理系统（CMS），如博客平台、新闻网站和数字资产管理系统等，MongoDB能够提供灵活的内容存储、高效的内容检索和易于扩展的存储解决方案。

MongoDB与SQLite比较

使用场景

MongoDB适合大数据量存储和高并发读写的场景。

SQLite适用于轻量级应用，需要简单、可靠且独立的数据存储解决方案。

数据模型

MongoDB的文档模型提供了更高的灵活性，适合需求变化快速的开发环境。

SQLite的关系模型适合结构化数据存储，且有严格模式要求。

扩展性和性能

MongoDB支持集群部署，可以处理大规模的数据和高并发请求。

SQLite作为嵌入式数据库，运行在客户端，不适合高并发和分布式环境。

代码实践

必要的头文件

```
#include <mongocxx/client.hpp>
#include <mongocxx/instance.hpp>
#include <bsoncxx/json.hpp>
#include <bsoncxx/builder/stream/document.hpp>
```

初始化MongoDB实例：

MongoDB C++驱动程序通过mongocxx::instance管理全局资源，通常在程序开始时创建一个全局实例：

```
mongocxx::instance instance{}; // 创建并初始化MongoDB全局实例
```

创建MongoDB客户端：

使用给定的URI连接到MongoDB服务器：

```
mongocxx::client client{mongocxx::uri{"mongodb://localhost:27017"}};
```

选择或创建数据库： 从客户端获取或引用数据库：

```
mongocxx::database db = client["my_database"];
// 使用名为"my_database"的数据库
```

操作集合

- 插入文档：

```
bsoncxx::builder::stream::document document{};
document << "username" << "user1" << "password" << "pass1";

auto collection = db["users"];
bsoncxx::stdx::optional<mongocxx::result::insert_one> result =
collection.insert_one(document.view());
```

查询文档

```
bsoncxx::builder::stream::document filter_doc{};
filter_doc << "username" << "user1";

auto cursor = collection.find(filter_doc.view());
for (auto&& doc : cursor) {
    std::string username = doc["username"].get_utf8().value.to_string();
    // 处理查询结果...
}
```

更新文档

```
bsoncxx::builder::stream::document update_doc{};
update_doc << "$set" << bsoncxx::builder::stream::open_document
           << "password" << "new_password"
           << bsoncxx::builder::stream::close_document;

bsoncxx::builder::stream::document filter_doc{};
filter_doc << "username" << "user1";

collection.update_one(filter_doc.view(), update_doc.view());
```

删除文档

```
bsoncxx::builder::stream::document filter_doc{};
filter_doc << "username" << "user1";

collection.delete_one(filter_doc.view());
```

面试常问问题

什么是大数据？它的主要特征是什么？

大数据是指体量巨大、类型多样、生成速度快的数据集合，具有“4V”特征：

1 体量（Volume）

数据量巨大，从TB到PB乃至更多。

2 多样性（Variety）

数据类型多样，包括结构化数据、半结构化数据和非结构化数据。

3 速度（Velocity）

数据流入的速度极快，需要实时或近实时处理。

4 价值（Value）

数据潜在的价值高，但需要通过分析挖掘。

MongoDB与传统关系型数据库相比有哪些优势？

回答：

MongoDB具有以下优势：

- **灵活的文档模型**：支持动态模式，方便存储复杂和多样化的数据。

- **高可用性和可扩展性**：内置复制和分片机制，支持水平扩展。

- **高性能**：适合读写密集型应用，支持索引优化查询。

- **丰富的查询语言**：支持复杂查询、聚合操作和全文搜索。

什么是数据分片（Sharding）？它的作用是什么？

回答：

数据分片是将数据库中的数据水平切分到多个节点上，每个分片存储数据的一个子集。其作用是提升数据库的性能和可扩展性，支持大规模数据存储和高并发访问。

解释什么是大数据的流处理，常见的流处理框架有哪些？

回答：

流处理是对实时生成的数据流进行持续的、即时的处理和分析。常见的流处理框架包括：

- **Apache Storm**：实时流处理系统，适用于实时数据分析。

- **Apache Flink**：支持流批一体的处理引擎，适用于复杂事件处理。

- **Apache Kafka Streams**：基于Kafka的流处理库，集成度高，易于部署。

对比SQL NoSQL NewSQL

特性	**SQL**	**NoSQL**	**NewSQL**
数据模型	关系型（表格、行列）	多种模型（键值、文档、列族、图）	继承SQL，支持关系模型，优化扩展性
一致性	强一致性（ACID）	最终一致性（CAP定理）	强一致性（ACID）
扩展性	垂直扩展	水平扩展	水平扩展
查询能力	强，支持复杂查询	弱，复杂查询困难	强，支持SQL查询
性能	适合中小规模应用	高并发、低延迟、大规模分布式应用	高性能，适合高并发、分布式环境
典型数据库	MySQL、PostgreSQL、Oracle	MongoDB、Cassandra、Redis	Google Spanner、TiDB、CockroachDB

什么是MongoDB的文档模型？相比关系型数据库中的表格模型，它有什么优势？

回答：

MongoDB的文档模型是以文档（类似JSON格式的BSON）为核心的数据存储形式，每个文档由键值对组成，可以存储复杂的嵌套数据结构。

与表格模型的对比和优势：

- **灵活性**：文档模型支持动态模式（Schema-less），可以在同一个集合中存储结构不同的数据，方便处理非结构化和半结构化数据。
- **自然的对象映射**：文档与应用程序中的对象（如JSON、JavaScript对象）自然对应，无需额外的对象-关系映射（ORM）。
- **嵌套结构**：支持嵌套文档存储，适合表示复杂的数据关系，不需要表的关联（Join）。
- **高性能**：通过文档嵌套减少了传统表格模型中的表关联操作，查询和写入速度更快。
- **扩展性**：文档的自包含特性使得水平扩展更为容易，特别是在分布式环境中

什么是“最终一致性”？MongoDB如何实现？

最终一致性是CAP理论中的一种一致性模型，指的是在分布式系统中，虽然数据在某一时刻可能是不同步的，但经过一段时间后，所有副本都会达到一致状态。

- **副本集机制**：MongoDB通过副本集（Replica Set）实现数据的复制，副本集中的主节点负责写入，副节点异步复制主节点的数据。
- **读写分离**：在高并发场景中，客户端可以选择从副节点读取数据，此时可能会遇到延迟导致的数据不一致，但最终会同步。
- **一致性级别选项**：
  - 默认的“读取首选项”（Read Preference）可以配置为优先从主节点读取以确保一致性。
  - 可以通过“写入确认”（Write Concern）设置写入操作是否需要等待数据复制到副节点后才返回，来权衡一致性和性能。

适用场景：高并发读操作：允许数据短时间不一致，例如社交媒体的点赞计数。分布式系统中对性能要求较高且不需要严格一致的场景。

MongoDB与Redis有什么区别？它们在使用场景上如何选择？

特性	MongoDB	Redis
数据模型	文档数据库，支持JSON/BSON格式	键值数据库，数据存储在内存中
一致性模型	支持最终一致性，事务支持较弱	单线程模型，强一致性，通过事务支持原子性
存储方式	持久化存储为主，支持水平扩展	内存存储为主，可选持久化，读写速度极快
扩展性	水平扩展较好，适合分布式系统	水平扩展较弱，更多用于单节点或小规模集群
性能	查询速度较快，适合复杂数据查询	超高性能，低延迟，适合高频读写操作
使用场景	数据量大、查询需求复杂的场景	实时数据处理、高频缓存

MongoDB是如何实现高可用性的？什么是副本集？

高可用性实现：MongoDB通过副本集（Replica Set）实现高可用性。副本集是一组包含主节点（Primary）和副节点（Secondary）的MongoDB实例，用于数据复制和故障转移。

副本集的组成与工作原理：

1. **主节点（Primary）**：
   - 负责处理所有的写操作和强一致性读操作。
   - 将写入的数据同步到副节点。
2. **副节点（Secondary）**：
   - 异步从主节点复制数据。
   - 在主节点故障时，通过选举机制成为新的主节点。
3. **仲裁节点（Arbiter）**：
   - 不存储数据，仅参与选举，确保选举机制顺利进行（适用于副节点数目为偶数时）。

谢谢大家