**一、分布式缓存**  
**特点**：

* 应用进程与缓存进程分布在不同的服务器上，通过网络协议（如RPC、HTTP）进行通信。
* 缓存服务独立部署，可横向扩展，支持高并发和大容量数据存储。
* 数据存储在独立的缓存节点中，与应用服务器解耦。

**优点**：

1. **解耦性**：缓存系统与应用分离，便于独立维护、升级和扩展。
2. **容量扩展**：支持分布式架构，可通过集群方式扩展存储容量，适应大数据量场景。
3. **高可用**：通常具备主从复制、数据分片等功能，提升系统容错能力。
4. **跨应用共享**：不同应用可以共享同一套缓存服务，避免数据冗余。

**缺点**：

1. **性能损耗**：网络通信引入延迟，相比本地缓存，读写速度较慢。
2. **复杂性**：需要额外配置和维护缓存集群，涉及网络配置、负载均衡、数据一致性等问题。
3. **成本**：需要额外的服务器资源部署缓存节点。

**使用对象**：

**1. Redis**

* + 高性能的，开源的，基于内存的数据结构存储系统，支持多种数据结构。
  + **核心特性：**
    - 键值存储，支持丰富的数据结构。
    - 支持原子性操作。
    - 有两种持久化方式（RDB和AOF）。
    - 支持事务且有多种集群模式。
  + **优点：**速度极快，数据结构丰富，简单易用，功能全面，高可用和可扩展性好。
  + **缺点：**成本较高，单线程模型，在处理复杂命令可能造成阻塞。
  + **适用场景：**高性能缓存，轻量级消息队列，会话管理，分布式锁。

**2. Memcached**

* + 高性能的分布式内存对象缓存系统，运用减轻数据库负载，通过缓存数据来加速Web应用。
  + **核心特性：**
    - 基于slab allocation的内存管理机制。
    - 键值存储，简单的字符串数据结构。
    - 多核支持，分布式。
  + **优点：**速度极快(极简设计),内存管理高效(slab allocation减少内存碎片)，多线程，部署简单。
  + **缺点：**只支持简单键值对，无内置高可用和持久化，内存利用率受限于slab大小。
  + **适用场景：**纯粹的，简单的分布式缓存(尤其是大规模读请求的场景，如减轻数据库压力)，或者仅需键值对存储的会话管理。

**3. Hazelcast**

* + 开源的内存数据网格平台，提供分布式的数据结构、分布式计算、缓存、流处理等功能。它提供了一种将数据和计算分布在集群节点上的方式。
  + **核心特性：**
    - 分布式数据结构，分布式时间处理，分布式执行器。
    - Sql查询，事务支持，集群管理和自动发现。
    - 数据分片和备份，客户端/服务器部署模式。
  + **优点：**一体化解决方案，真正的分布式，易于集群部署和扩展，支持SQL查询，提供高性能的内存计算能力。
  + **缺点：**相对复杂，功能多，学习内容多，资源消耗过大，社区不够成熟。
  + **适用场景：**内存数据网格，分布式缓存，分布式任务和任务执行，流处理，应用状态管理。

1. **Apache Ignite**
   * 开源的分布式内存计算平台和数据网格，旨在存储和处理PB级数据，提供高性能的内存计算，分布式SQL,事务和持久化能力。
   * **核心特性：**
     + 分布式数据网格(键值存储)。
     + ANSI-99 SQL存储(通过JDBC),具有ACID事务。
     + 拥有持久化。
     + 内存计算，流处理，机器学习API。
   * **优点：**SQL支持，降低适用门槛，有ACID事务来保持数据的一致性，内置持久化，可作为主存储，内存计算，功能强大。
   * **缺点：**复杂性高，学习和运维成本高，资源消耗高。
   * **适用场景：**内存数据库(作为主数据存储)，事务处理(需要ACID保证的分布式应用)，高性能分布式缓存，高速数据分析(近乎实时)，大数据加速，分布式服务网络。

**分布式缓存总结：**

* **Redis：**当你需要一个高性能、多功能、灵活的内存数据存储，主要用于缓存、计数、排行榜等场景时，首选Redis。
* **Memcached：**当你只需要一个极简、极快的纯粹键值对缓存，并且不需要复杂数据结构、持久化或内置高可用时，可以考虑Memcached。
* **Hazelcast：**当你需要一个易于部署、一体化的内存数据网格，用于分布式缓存、状态管理或轻量级分布式计算时，选择Hazelcast。
* **Apache Ignite：**当你需要一个强大的、带有SQL和ACID事务支持的内存数据库/计算平台，用于高性能OLTP/OLAP混合负载或作为应用的主数据存储时，选择Apache Ignite。

**二、本地式缓存**  
**特点**：

* 缓存进程与应用进程在同一JVM中，数据读写直接在内存中完成，无需网络通信。
* 适用于对性能要求极高且数据量较小的场景。

**优点**：

1. **极致性能**：无网络开销，访问速度极快，适用于低延迟场景。
2. **简单轻量**：无需独立部署，开发和维护成本低。
3. **线程安全**：可通过同步机制或线程安全数据结构保障并发安全。

**缺点**：

1. **容量限制**：受JVM堆内存限制，无法存储大量数据。
2. **数据隔离**：缓存数据仅存在于单个应用实例中，无法跨节点共享，可能导致数据不一致。
3. **扩展性差**：无法通过集群扩展容量，不适合分布式场景。

**使用对象**：

1. **自实现缓存（基于HashMap）**：
   * 使用LinkedHashMap实现LRU淘汰策略。
   * 利用读写锁保障并发安全。
   * **优点**：实现简单，无需引入第三方依赖。
   * **缺点**：功能受限，需自行扩展，稳定性和可靠性依赖开发者实现。
   * **适用场景**：数据量小、业务简单的场景。
2. **Guava Cache**：
   * Google开源的本地缓存库，功能丰富，线程安全。
   * **核心特性**：
     + 支持容量限制、基于时间和访问的过期策略。
     + 提供统计功能（如命中次数、加载耗时等）。
     + 基于LRU算法实现，同时支持自定义淘汰策略。
     + 支持异步加载数据（通过CacheLoader）。
   * **优点**：成熟稳定，功能全面，易于使用。
   * **缺点**：相比Caffeine，性能略低，但足够满足大多数场景。
   * **适用场景**：需要复杂缓存策略的中小型应用。
3. **Caffeine**：
   * 新一代高性能本地缓存库，号称“接近理论最优性能”。
   * **核心特性**：
     + 基于W-TinyLFU算法（结合LRU和LFU优点），更精准的缓存淘汰策略。
     + 性能优于Guava，尤其在读写密集型场景。
     + 支持异步加载、监听器、权重计算等高级功能。
   * **优点**：极致性能，功能先进，API与Guava相似，迁移成本低。
   * **缺点**：相对较新，部分场景可能需评估兼容性。
   * **适用场景**：对性能要求极高的场景，或需要复杂缓存策略的大型应用。
4. **Encache：**
   * 纯Java的进程内缓存框架，具有快速，精干等特点，是Hibernate中默认的CacheProvider。
   * 具有大量功能，极高的扩展性。
   * **核心特征：**
     + 支持多种缓存淘汰算法。
     + 缓存支持堆内存储，村外存储，磁盘存储（支持持久化）三种。
     + 支持多种集群方案，解决数据共享问题。
   * **优点：**性能高效，易于使用和集成，功能丰富，成熟稳定。
   * **缺点：**分布式能力相对较弱，可能会在内存中有较大开销，持久化能力有限。
   * **适用场景：**数据量在单机内存和磁盘容量范围内，且对分布式强一致性要求不高的场景。

**本地式缓存总结：**

* **自实现缓存：**当你只是简单存储几个静态配置，使用ConcurrentHashMap。
* **EnCache：**当你目前的项目很老，对磁盘缓存有需求，用EnCache。
* **Guava Cache：**当你的项目不算大，对性能要求不高时，使用Guava，但还是建议迁移到Caffeine,提高性能。
* **Caffeine：**目前java生态中性能和命中率最高的本地缓存，用它准没错。

**三、总结与选型建议**

1. **何时选择分布式缓存？**
   * 数据量较大，超出单机内存容量。
   * 需要跨应用或跨节点共享数据。
   * 系统具备高可用和扩展性需求（如分布式集群）。
   * 可接受一定的网络延迟（例如毫秒级）。
2. **何时选择Java本地缓存？**
   * 数据量小，且访问频率极高（如热点数据）。
   * 对延迟极其敏感（微秒级），无法容忍网络开销。
   * 应用为单体架构，无需跨节点共享数据。
   * 追求简单轻量，无需复杂配置。

**四、参考文献**

[**本地缓存无冕之王Caffeine Cache-腾讯云开发者社区-腾讯云**](https://cloud.tencent.com/developer/article/2343991)

[**Home zh CN · ben-manes/caffeine Wiki**](https://github.com/ben-manes/caffeine/wiki/Home-zh-CN)

[**EhCache看这一篇就够了-CSDN博客**](https://blog.csdn.net/qq_37391229/article/details/118522247)

[**Java分布式缓存配置全攻略：5种主流框架选型对比及落地建议-CSDN博客**](https://blog.csdn.net/InitPulse/article/details/153695062)

[**Java本地缓存技术选型（Guava Cache、Caffeine、Encache） - 简书**](https://www.jianshu.com/p/e5dc3a18dcb8)

[**Redis - The Real-time Data Platform**](https://redis.io/)

[Hazelcast Documentation](https://docs.hazelcast.com/home/)