

分散式系統期中第十一組 Kafka 解析

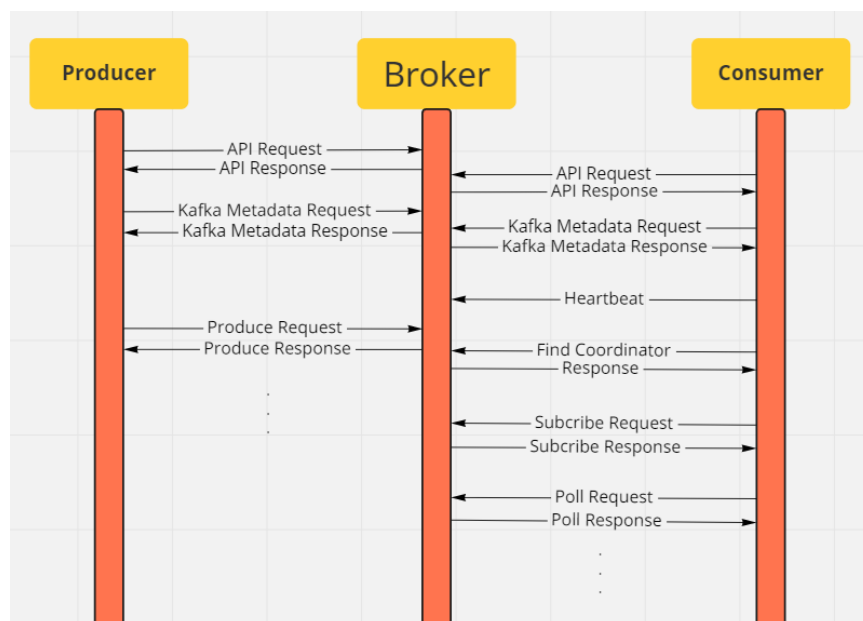
壹、問題目的與背景

Kafka 建立一個數據中心來作為 Message Queue Middleware，它的高產出及低延遲可以有效防止突發性的大數據湧入，並且因為將資訊都傳入 Kafka 處理，可以適當的解放其他系統的設計靈活度。Kafka 作為一個速度快的分散式 event streaming platform，他可以使大量的資料持續的在正確的時間流到正確的位置。其中心為 Broker 的構造也使 Producer 與 Consumer 不必知道互相的位置，可輕鬆替換系統中的組件。很多公司會使用它來做即時的資料串流或反饋。上述特性使 Kafka 常用在以下情景。第一即時的付款與金融交易。如證券交易所與銀行；第二追蹤跟監控卡車，貨物。如物流業；第三持續的搜集用戶的操作並做即時的反饋。如 netflix 的推薦下一部的功能即使用到 Kafka。

若沒有中心的 Broker 構造，分散式系統中的每個組件就必須知道互相的位置，整個系統的耦合度極高。即便自己寫了個簡易的 Broker，它掛掉整個系統就癱瘓了。但 Kafka 能讓各 Broker 間可以相互協調、取代，使穩定度大幅提升。

貳、架構解析

Kafka 中的節點可以分為三種，分別是 Broker, Producer, 可大致理解為以下時序圖，由 Consumer 先向 Broker 訂閱某個 topic。當任一 Producer 向這個 topic 發布消息的時候，所有訂閱此 topic 的 Consumer 都能收到此消息。



108753208 葉冠宏 109753209 李庭慶 110703065 詹松霖
110753113 張皓博 110753147 王盛泰 110753161 鄭仁傑

一、Kafka Producer's view(著重核心功能 send)

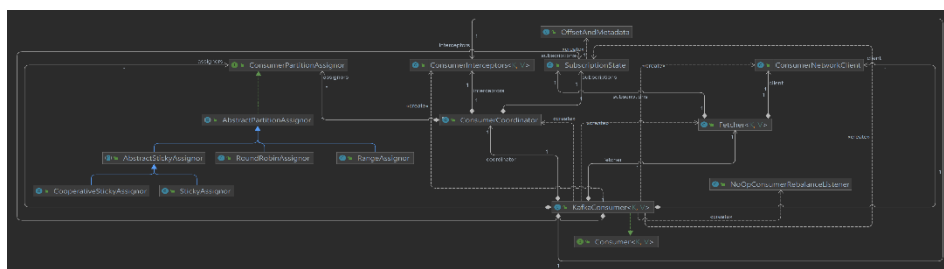
當 send 被 call 時，Producer 會先向 Cluster 發送 API version 與 Kafka metadata 的請求。做 protocol 的確認與獲得主要負責該 topic,partition 的 Broker 的資訊。取得的資料會被存進 **ProducerMeta**，方便之後的 send 可以直接由此處調用所需資料。

接著要發送的訊息會被暫存到 `RecordAccumulator`，當暫存的 `record` 總大小超過 `ProducerConfig` 所設定的 `BATCH_SIZE`，或是經過設定的 `LINGER_MS` 時長後，會把 `RecordAccumulator` 中的 `records` 包裝成一個 `Produce Request` 發送到 `Broker`。`Broker` 在處理好訊息後會回傳一個 `Producer Response`，讓 `Producer` 可在 `CallBack` 中對 `error` 做處理。若是想立即發送訊息也可以使用 `flush()` 來將當前 `Accumulator` 中的訊息發出。

Kafka 有類似於 Mqtt 的 sharding 的概念，叫做 partition。可以把一個 topic 切成多段處裡，使系統可以平行擴充，架設多個 Broker 來分擔壓力。那麼如何決定一個訊息該發到哪個 partition? Producer 可以在 send 的時候"選擇"加入 partition number 以及 key 的參數。若有設定 partition number，則該訊息就會被發到對應編號的 partition。若沒有 partition number，則由 key 的 hash 決定。若只單純發送 value 的話，一個 producer 就會流輪發送到不同 partition。

二、Kafka Consumer's view

以 Consumer 的觀點來分析 Kafka，著重在 Consumer 核心功能 Subscribe(), Poll()以及 Kafka 透過 Rebalance 來進行有效率的資源分配。會先透過 Class Diagram 對於 Kafka Consumer 整體架構有初步的了解，整體的類別圖會著重在 KafkaConsumer 與核心功能 Class 間的繼承、實作介面、呼叫，能有助於對 Kafka Consumer 的理解以及往後的實作細節。Rebalance 功能會使用 Sequence Diagram 時序圖來表示，因為觸發 Rebalance 事件需要類似 Topic 中 Partitions 數量改變，Consumer Group 中有新成員加入或退出，這樣的事件都是涉及時間及元件之間



108753208 葉冠宏 109753209 李庭慶 110703065 詹松霖
110753113 張皓博 110753147 王盛泰 110753161 鄭仁傑

的互動，因此會用較直觀的時序圖來表達這樣的關係。

上圖的 Class Diagram 透過 IntelliJ IDEA 內建的 UML 所繪製。所呈現的 Class Diagram 並非是與 Consumer 所有相關聯的功能都繪製在此圖。此圖著重在實作 Consumer 介面的 KafkaConsumer 中核心功能。

- **ConsumerCoordinator**：繼承 **AbstractCoordinator** 介面，管理與 consumer coordinator 相關的功能例如 Group Registration, Group/Leader Selection, State Assignment 功能。且透過此才能與 **ConsumerGroupMetadata** 溝通。
- **ConsumerPartitionAssignor**：用來定義 Consumer partitions 如何分配的介面，Kafka 四種分配策略也是繼承此介面分別是 **RangeAssignor**, **RoundRobinAssignor**, **StickyAssignor**, **CooperativeStickyAssignor**。
- **ConsumerNetworkClient**：提供 Consumer 與網路接取的相關功能
- **Fetcher**：負責向 Broker 獲取訊息的相關功能且是 Thread-safety
- **NoOpConsumerRebalanceListener**：實作 **ConsumerRebalanceListener** callback interface，當觸發 Rebalance 時所採取的行動，前提是 Consumer 採用 Kafka auto-manage group membership 才適用。
- **SubscriptionState**：用來追蹤紀錄 Consumer 的 topics, partitions, and offsets
- **OffsetAndMetadata**：當 offset 被確認後允許使用者提交額外的元資料(哪個節點哪個時間)。
- **ConsumerInterceptors<K, V>**：存有 list 傳入 custom interceptors，可以在 poll(), offset 傳回成功與否前進行攔截及修改。

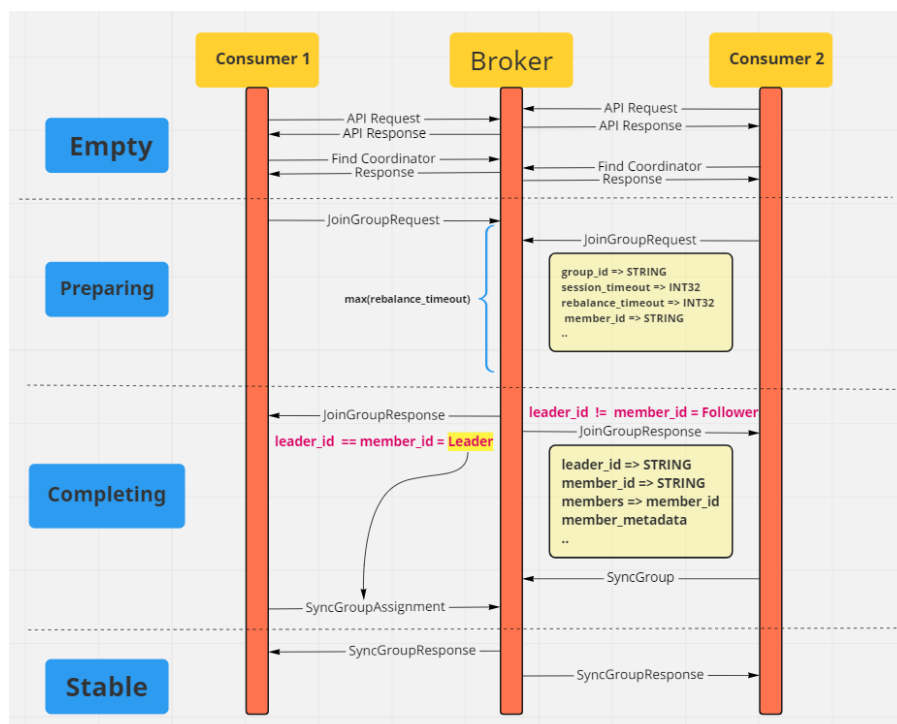
[Kafka Scbscribe Source code](#)，用此原始碼說明 Kafka 訂閱的細節實作。

1. 先確定 Consumer 在不在且取得 lock(acquireAndEnsureOpen())
2. 檢查 GroupID 存不存在(maybeThrowInvalidGroupIdException())
3. 訂閱的 Topic 不能是 empty or null
4. 檢查有無 Partition assigner
5. 訂閱動作(fetcher.clearBufferedDataForUnassignedTopics(topics))
6. 把新訂閱的加到 currentTopicPartitions.add()
7. 把非此次的訂閱的 Topic 從 buffer 中清掉
8. 訂閱的 Topic(s) 寫入 log
9. 更新時戳
10. 釋放掉 lock

[Kafka Poll Source code](#)，用此原始碼說明 Kafka Poll 的細節實作。

1. 先確定 Consumer 在不在且取得 lock(acquireAndEnsureOpen())
2. 紀錄此次的 poll 時間及距離上次 poll 的時間
3. 檢查 Consumer 有沒有訂閱主題(hasNoSubscriptionOrUserAssignment())
4. 每個 Poll 都要先確保對外連接的 client 是否被喚醒
5. 嘗試更新分配的 metadata 且不會影響到 Join Group 的 Timer
6. 先確保 fetch 到的 record 數量不是 0
7. 回傳的值都是上次 Fetch 的內容此段在做檢查並讓效率提高作 Pipeline
8. 如果 fetch 到的 records 是 empty 寫入 log(可能是因為某些因素被中止而導致的)
9. 回傳 fetch 到的 records(Default 每次最大 fetch 的 record 數量 500)
10. 重複 5-10 直到 Timer 過期
11. 回傳一個空的 Key Value 值
12. 釋放掉 lock(release())
13. 將此次的 Poll 的衡量指標記錄到 kafkaConsumerMetrics

接著用下圖用來解釋 Consumer 中重要的 Rebalance 機制，以及 Consumer Group 不同的狀態。運用兩個不同的 Consumer 加到同一個 Consumer Group 的情況來解釋且有用 Wireshark Trace 過程再將此過程透過 Miro 工具繪製。



108753208 葉冠宏 109753209 李庭慶 110703065 詹松霖
110753113 張皓博 110753147 王盛泰 110753161 鄭仁傑

Ms	Time	Source	Destination	Protocol	Length	Info
	1538 21:21:53.045443	127.0.0.1	127.0.0.1	Kafka	231	Kafka JoinGroup v7 Request (Group=demo, Member=consumer-demo-1-...
	1546 21:21:53.173331	127.0.0.1	127.0.0.1	Kafka	65	Kafka Response (Unencoded, Request Missing)
	1548 21:21:53.174646	127.0.0.1	127.0.0.1	Kafka	103	Kafka Fetch v13 Request [Unsupported API version][Malformed Pac...
	1590 21:21:53.688323	127.0.0.1	127.0.0.1	Kafka	65	Kafka Response (Unencoded, Request Missing)
	1592 21:21:53.690208	127.0.0.1	127.0.0.1	Kafka	103	Kafka Fetch v13 Request [Unsupported API version][Malformed Pac...
	1640 21:21:54.204379	127.0.0.1	127.0.0.1	Kafka	65	Kafka Response (Unencoded, Request Missing)
	1642 21:21:54.205585	127.0.0.1	127.0.0.1	Kafka	103	Kafka Fetch v13 Request [Unsupported API version][Malformed Pac...
	1654 21:21:54.720170	127.0.0.1	127.0.0.1	Kafka	65	Kafka Response (Unencoded, Request Missing)
	1656 21:21:54.721266	127.0.0.1	127.0.0.1	Kafka	103	Kafka Fetch v13 Request [Unsupported API version][Malformed Pac...
	1698 21:21:55.232695	127.0.0.1	127.0.0.1	Kafka	65	Kafka Response (Unencoded, Request Missing)
	1700 21:21:55.233845	127.0.0.1	127.0.0.1	Kafka	103	Kafka Fetch v13 Request [Unsupported API version][Malformed Pac...
	1704 21:21:55.357951	127.0.0.1	127.0.0.1	Kafka	138	Kafka Heartbeat v4 Request (Group=demo, Member=consumer-demo-1-...
	1706 21:21:55.359206	127.0.0.1	127.0.0.1	Kafka	60	Kafka Heartbeat v4 Response [Group rebalance in progress]
	1708 21:21:55.362479	127.0.0.1	127.0.0.1	Kafka	231	Kafka JoinGroup v7 Request (Group=demo, Member=consumer-demo-1-...
	1710 21:21:55.365093	127.0.0.1	127.0.0.1	Kafka	186	Kafka JoinGroup v7 Response (Member=consumer-demo-1-c68770e6-ae...
	1712 21:21:55.365660	127.0.0.1	127.0.0.1	Kafka	344	Kafka JoinGroup v7 Response (Member=consumer-demo-1-b4ce73ce-9f...
	1714 21:21:55.367247	127.0.0.1	127.0.0.1	Kafka	322	Kafka SyncGroup v5 Request (Group=demo, Member=consumer-demo-1-...
	1716 21:21:55.368059	127.0.0.1	127.0.0.1	Kafka	154	Kafka SyncGroup v5 Request (Group=demo, Member=consumer-demo-1-...
	1718 21:21:55.369860	127.0.0.1	127.0.0.1	Kafka	107	Kafka SyncGroup v5 Response
	1720 21:21:55.370187	127.0.0.1	127.0.0.1	Kafka	103	Kafka SyncGroup v5 Response
	1722 21:21:55.371257	127.0.0.1	127.0.0.1	Kafka	102	Kafka OffsetFetch v8 Request [Unsupported API version][Malform...

Consumer Group 狀態為 Empty

1. Consumer 會先向 Broker 要後續溝通所需的 API 及版本
2. Consumer 會收到 Broker 所回應的 API 版本
3. Consumer 會去找一個特定 Broker 作為 coordinator 負責監控再加入 Consumer Group 後的相關事項(Heartbeat, 啟動 Rebalance 等)
4. Consumer 會收到回應的特定 coordinator

Consumer Group 狀態為 Preparing Balance

5. Consumer 會去申請加入 ConsumerGroup
 - group id : 要加入的 group id
 - session_timeout : 不用 heartbeat 也能確保 Consumer 還在
 - rebalance_timeout : 用來等 Consumer 重新加入群組的時間

Consumer Group 狀態為 Completing Balance

6. Consumer 會收到 Broker 的 JoinGroup 回應
 - 通常第一個送要 JoinGroupRequest 的消費者會是 Leader
 - 可以由 leader_id 及 member_id 去判斷為 Leader 或 Follower
 - 內有各組員訂閱狀況
7. Follower 會送出 SyncGroup 給 Broker
 - 但內容為空意思在等待 leader 分配後的結果
8. Leader 會收集 JoinGroup 回應後做出分配並傳給 Broker
 - 分配策略就涉及到上面所提 ConsumerPartitionAssignor 介面
 - RangeAssignor, RoundRobinAssignor, StickyAssignor,

Consumer Group 狀態為 Completing Balance

9. 最後每位 Consumer 會收到被分配的狀況直到下次觸發 Rebalance 事件

三、Kafka Zookeeper Broker's view

動機：Zookeeper 做好了分散式系統的協調服務讓分散式系統直接引用，免去從頭開始實作協調服務的麻煩。Zookeeper 具有以下特色 Reliability:可以在有些節點 fail 時，依然持續運作。Simplicity 可以藉由 shared hierarchical namespace 協調 process。Speed 可以快速的處理 Reading 的 workloads。Scalability 是 horizontally scalable，可以藉由僅僅增加 node 來擴充。

Broker 在 Zookeeper 中的註冊

- 為了記錄 Broker 的註冊訊息，在 Zookeeper 上專門創建了屬於 Kafka 的一個節點，其路徑為/brokers
- Kafka 的每個 Broker 啟動時，都會到 Zookeeper 中進行註冊，告訴 Zookeeper 其 broker.id。而在整個群集中，broker.id 全局唯一，並在 Zookeeper 中創建其屬於自己的節點，其節點路徑為/brokers/ids/{broker.id}。
- 創建完節點後，Kafka 會將該 broker 的 broker.name 及 port 記錄到該節點。
- broker 節點屬性為臨時節點，當 broker 失效時，Zookeeper 會刪除該節點，可以方便監控到 broker 節點的變化及 rebalance。
- 可以設定 broker 中的 data 持續儲存時間長短，因為維持越久的 data，會花費越多的成本來維護。

Topic 在 Zookeeper 中的註冊

- 在 Kafka 中，所有 topic 及 broker 的對應關係都由 Zookeeper 進行維護。在 Zookeeper 中，建立專門的節點來記錄這些訊息，其節點路徑為/brokers/topics/{topic_name}。
- 為了保障數據的可靠性，每個 Topic 的 partitions 實際上是存在備份的，並且備份數量由 Kafka 機制中的 replicas 來控制。為了保障數據的一致性，Kafka 為每一個 partition 找一個節點作為 leader，其餘備份作為 follower。當 producer push 的訊息寫入 partition 時，作為 leader 的 broker 會將訊息寫入自己的分區，同時還會將訊息複製到各個 follower，實現同步。而 follower 只是跟隨 leader 的變動去更新資料。如果某個 follower 掛掉，leader 會再找一個替代，並同步訊息。如果 leader 掛了，follower 們會選舉出一個新的 leader 代替，繼續服務，這些都是由 Zookeeper 完成的。

Consumer 在 Zookeeper 中的註冊

當新的 consumer 註冊到 Zookeeper 時，Zookeeper 會創建專用的節點來保存相關的訊息，其節點路徑為 /consumers/{group_id}。其節點下有三個子節點，分別為[ids, owners, offsets]。

108753208 葉冠宏 109753209 李庭慶 110703065 詹松霖
110753113 張皓博 110753147 王盛泰 110753161 鄭仁傑

- **ids** 節點:紀錄該消費組中正在消費的 **consumer**。
- **owners** 節點:紀錄該消費組消費的 **topic** 訊息。
- **offsets** 節點:紀錄每個 **topic** 的每個分區的 **offset**。

借助 Zookeeper 實現 Rebalance Sequence

Consumer 在消費時只需制定 **topic**，借助 **Zookeeper** 可以根據 **partition** 的數量和 **consumer** 的數量做到均衡的動態配置。**Consumer** 在啟動時會到 **Zookeeper** 下以自己的 **consumer-id** 創建臨時節點 **/consumer/[group-id]/ids/[consumer-id]**，並對 **/consumer/[group-id]/ids** 註冊監聽事件。當 **consumer** 發生變化時，同一個 **group** 的其他 **consumer** 會得到通知。

紀錄 Consumer 進度的 offset

在 **consumer** 對指定的訊息進行消費的過程中，需要定時的將 **partition** 消息的消費進度 **offset** 記錄到 **Zookeeper** 中，以便在該 **consumer** 進行重啟或者其他 **consumer** 重新接管該消息 **partition** 的消費權後，能夠從之前的進度開始繼續進行消費。**Offset** 在 **Zookeeper** 中由一個專門的節點進行紀錄，其節點路為 **/consumers/[group-id]/offsets/[topic]/[broker_id-partition_id]**

紀錄 consumer 與 partition 的關係

在 **Kafka** 中，規定每個 **partition** 只能被同組的一個消費者進行消費。因此需要在 **Zookeeper** 上記錄下 **partition** 與 **consumer** 之間的關係。每個 **consumer** 一旦確定了對一個 **partition** 的消費權利，需要將其 **consumer ID** 寫入到 **Zookeeper** 對應消息分區的臨時節點上。

例如 **:/consumers/[group-id]/owners/[topic]/[broker_id-partition_id]** 其中，**[broker_id-partition_id]** 就是一個消息分區的標示，節點內容就是該消息分區 **consumer** 的 **consumer ID**。

broker 啟動流程

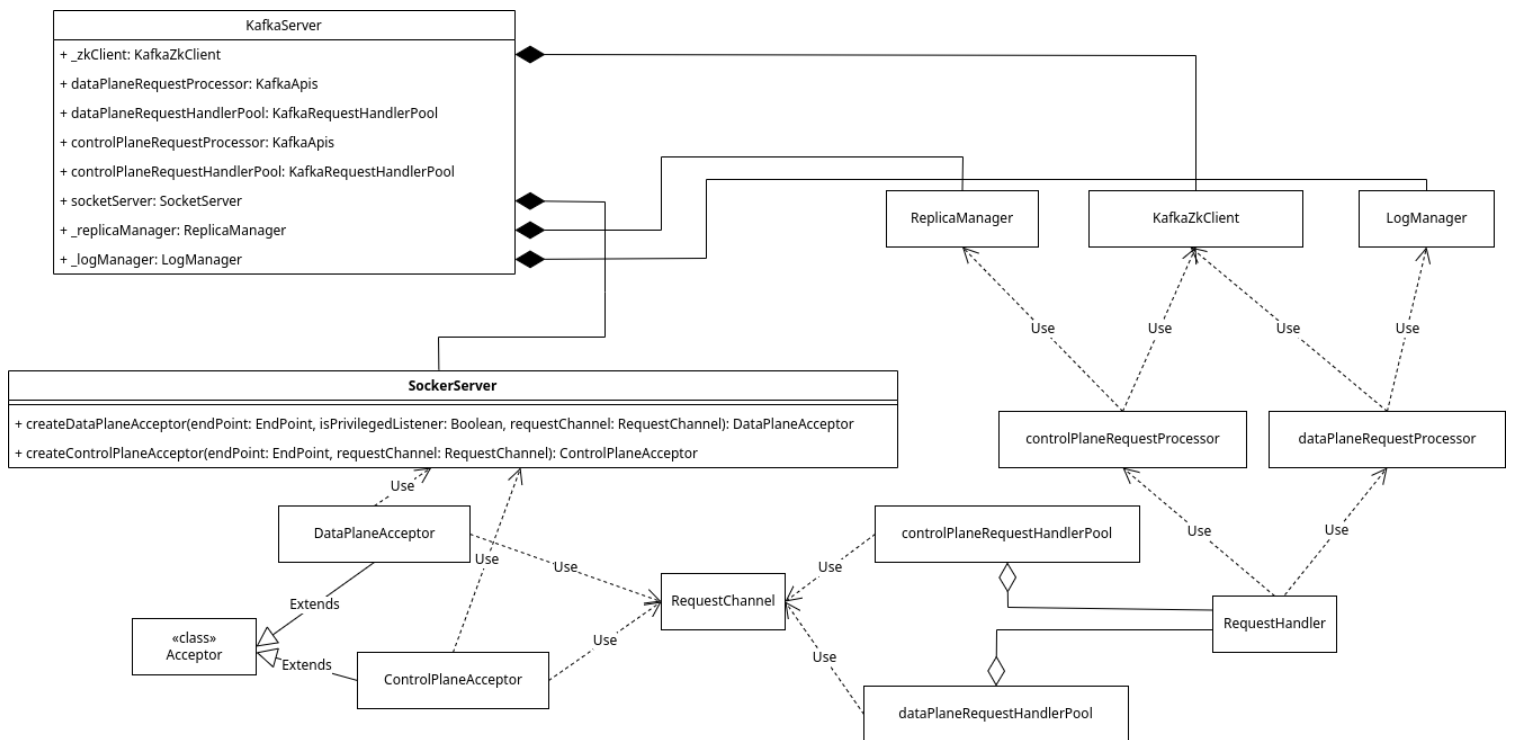
broker 是從路徑 **core/src/main/scala/kafka/Kafka.scala** 開始執行，會以讀入的 **properties** 檔來啟動不同的 **broker**，如果以 **config/server.properties** 的範例設定來設定，會採用 **Zookeeper** 來同步 **replica node**，若以 **config/kraft/server.properties** 的範例設定設定時，則會以 **Raft** 的方式來同步。以 **Zookeeper** 來設定則會產生 **KafkaServer** 的物件，**KafkaServer** 的主要成員如下圖。



108753208 葉冠宏 109753209 李庭慶 110703065 詹松霖
110753113 張皓博 110753147 王盛泰 110753161 鄭仁傑

_zkClient 為 zookeeper server 的 client 端，會對其他相同連線到 zookeeper server 的 replica node 發送同步，dataPlaneRequestProcessor、dataPlaneRequestHandlerPool、controlPlaneRequestProcessor、controlPlaneRequestHandlerPool、socketServer 則是處理從網路來的請求，一開始會將 RequestProcessor 以及 RequestHandlerPool 註冊進 socketServer 的 data plane 以及 control plane，當有新的 Request 時，socketServer 便會從 RequestHandlerPool 產生對應的 RequestHandler 並執行，_replicaManager 是管理節點的 partition，依照 partition 的同步狀況來決定是否繼續同步該 partition，partition 內容紀錄則是靠 _logManager 來完成，_logManager 會根據設定檔中的 log.dirs 的路徑存放 consumer、topic、等與 broker 相關的資料。以預設的 config/server.properties 來說，會存放在 /tmp/kafka-logs。

下圖為 Kafka broker 的 UML 結構圖

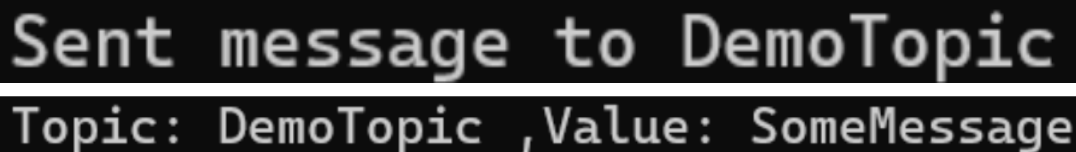


參、技術實作與心得

[Demo Source Code](#) 用此程式碼來說明以下實作內容。首先把 Cluster 架起來，開啟兩個 terminal，分別運行 Zookeeper 與 1 個 Kafka Broker。在 server.properties 中設定 broker 的 ip 與 host，這邊設定 localhost:9092。

```
.\zookeeper-server-start.bat .\zookeeper.properties  
.\kafka-server-start.bat .\server.properties
```

Cluster 架好後創建 Topics，這邊創建一個名叫 DemoTopic 的 Topic。接著寫 Consumer 來訂閱 DemoTopic 的訊息。再來 Producer 發布一條 "SomeMessage" 的訊息到 DemoTopic。相關執行順序如先執行 [Admin](#) 創建 Topic 後，接著執行 [Producer](#)，Producer 輸出如下圖(Sent Message to DemoTopic)。再來 [Consumer](#) 可看到(Topic:DemoTopic, Value:SomeMessage)。



Sent message to DemoTopic
Topic: DemoTopic ,Value: SomeMessage

肆、評論

對於單純的任務，沒有太多的節點且訊息量不大不適用於 Kafka，例如單純在晚上時 Publisher 發出「晚上了」給 Subscriber，早上時發出「早上了」，且 Subscriber 數量不多，此時使用 Kafka 就顯得大材小用了。當每筆 data 都需要保存一定時間且此資料本身並沒有什麼額外價值，但 Kafka 本身又會複製儲存副本，此時使用 Kafka 就需要龐大的維護、營運、儲存成本但卻為了簡單的任務而選擇了效能過剩的軟體及架構，只會增加 TCO(總擁有成本)。

上面的解析可以了解到 Kafka 具有負載平衡、減少耦合、擴展性、順序保證、可恢復性等特性，因此對於需要上述特色的則是適用於 Kafka，實際案例如 Netflix 推薦電影的功能，當企業的服務涉及範圍廣，且使用者通常會在首頁不斷下滑尋找有興趣的影單，因此面對需求量大且對於可用性及延遲有較高的要求就適用於 Kafka。

伍、結語

透過此次的期中 Kafka 解析，從三個不同分析系統的角度去看整體架構，可作為與課程 MQTT 相關的後續深入學習，也透過類別圖、時序圖、自定義圖片分析系統行為及設計細節，最終透過實作了解實際運行起來的成果並透過 Wireshark 檔案確認細節上有無錯誤。整理出以下特色且會將期末實作設計的更能凸顯出 Kafka 的特色及長處。

- **Kafka 設計**的初衷在於處理即時資料提供一個統一、高吞吐、低延遲的平台，因此設計的重點更著重在效能與擴充性，但也包括了 MQ 本來就具有的優點，如：不同步的溝通、可靠性等等，底下分別提到這些特性。
- **負載平衡**：透過 partition 的設計，producer 發布訊息在不同 broker 上，以 partition 為單位，可以依據 partition 目錄的訊息量、broker 目前的負載做負載平衡。
- **減少耦合**：透過 topic 的設計，producer、consumer 只要關心 broker 在哪，以及所選的 topic 即可，不須知道對方的資訊。
- **順序保證**：在 partition 中，又切割成 segment 為儲存單位，收到的訊息皆以循序的方式儲存，可以確保 partition 內部的順序儲存；對於 consumer 來說，由於一個訊息只能有一個 consumer 消耗，因此當 consumer 數量大於等於 partition 時，還可以再確保收到的訊息順序。
- **不同步溝通**：kafka 設計中允許 In-Sync Replicas 以外的副本不必同步更新，透過設定 min.insync.replicas 參數以及 In-Sync Replicas 數目，在可靠度與效率之間可以有彈性的調整。
- **擴展性**：可依照需求增加或減少 partition。
- **可恢復性**：在 partition 中儲存的訊息中，透過 HW 機制以及 broker 定時向 zookeeper 更新參數，當某些 broker 故障時，因為有副本存在，並不影響其他節點運作，而在系統恢復之後，可以利用副本以及 HW 恢復遺失的資料。
- **緩衝**：利用 message queue 做為緩衝，consumer 採用 pull 的方式獲取資料，可以照顧到不同效率的 consumer 設備，使 producer、consumer 不必互相等待。
- **高速存取**：直接存取磁碟而不必先複製到記憶體，且存取代價為 $O(1)$ 。

陸、分工

<https://hackmd.io/@VzDleuYhQTClswU9sMpD5w/BkOtPD-Vc>

問題(目的)與背景: 仁傑

架構解析: Broker: 庭慶 冠宏; Consumer: 皓博 Producer: 詹霖

技術實作與心得: 詹霖 評論: 仁傑 結語: 盛泰 書面資料: 皓博