



# Big Data Technology and Application

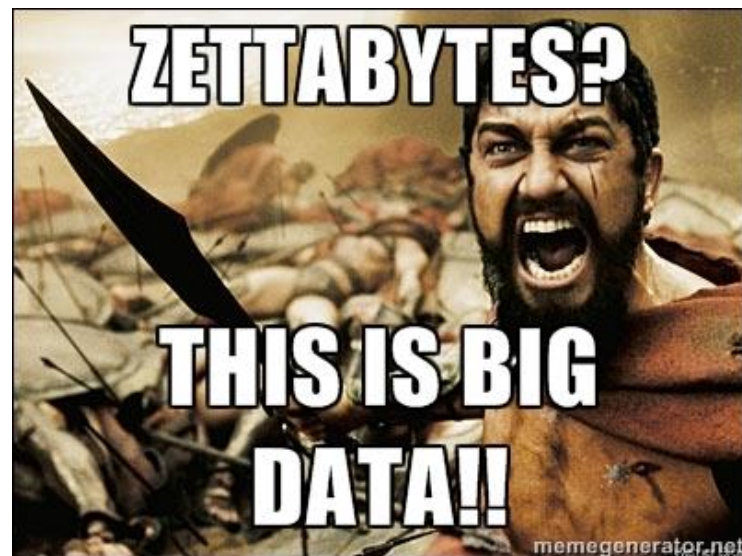
# Introduction to NoSQL Databases





# 大綱

- NoSQL簡介
- NoSQL興起的原因
- NoSQL與關聯式資料庫的比較
- NoSQL的四大類型
- NoSQL的三大理論基礎
- 從NoSQL到NewSQL資料庫
- Redis DB與MongoDB





# ■ NoSQL簡介



最初表示“反SQL”運動  
用新型的非關聯式資料庫取代關聯式資料庫

現在表示關聯式與非關聯式資料庫各有優缺點  
彼此都無法互相取代

通常，NoSQL資料庫具有以下幾個特點：

- (1) **靈活的可擴展性**：支援**橫向擴展**，不同於關聯式資料庫系統
- (2) **靈活的資料模型**：關聯式資料庫是由代數理論發展，故其關聯式資料模型使用上有著嚴格的規範，如：在使用前需明確定義欄位有哪些、它們的資料型態...等，並需遵守多種限制條件，無法動態更改架構。而NoSQL則沒有太多限制
- (3) **與雲端運算緊密融合**：可支援橫向擴展，故可結合雲端運算底層基礎架構



現在已經有很多公司使用了NoSQL資料庫：

- Google
- Facebook
- Mozilla
- Adobe
- Foursquare
- LinkedIn
- Digg
- McGraw-Hill Education
- Vermont Public Radio
- 百度...







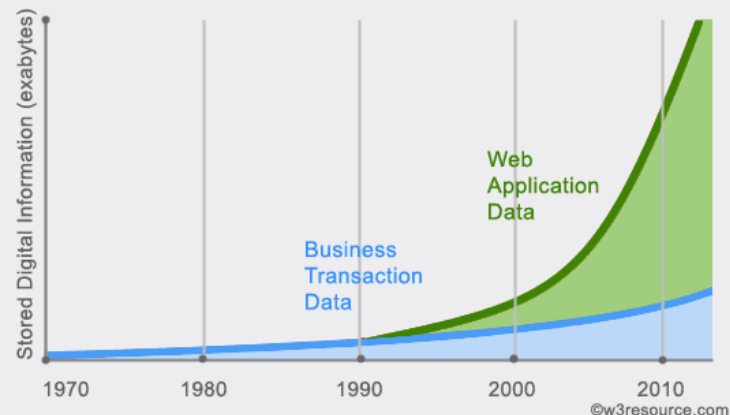
# NoSQL興起的原因

1、關聯式資料庫已經無法滿足Web2.0的需求。主要反應在以下方面：

- (1) 無法滿足巨量資料的管理需求
- (2) 無法滿足資料高動態即時的需求
- (3) 無法滿足高可擴展性和高可用性的需求



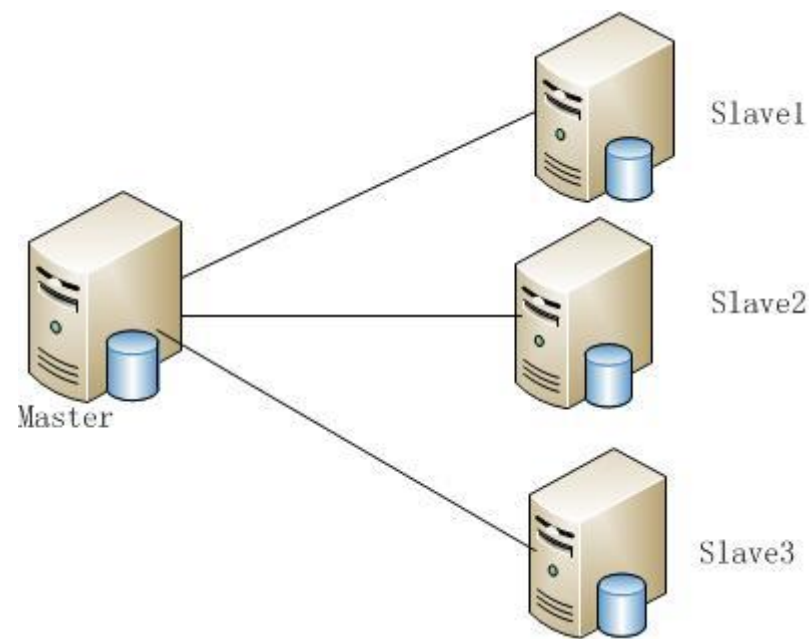
Web Applications Driving Data Growth





## MySQL集群是否可以完全解決問題？

- 複雜性：部署、管理、配置很複雜
- 資料庫複製：MySQL主從之間採用複製方式，只能是**非同步複製**，當主資料庫壓力較大時可能產生較大延遲，主從切換可能會遺失最後一部分更新交易，這時往往需要人工介入，備份和恢復不方便
- 擴充問題：如果系統壓力過大需要增加新的機器，這個過程涉及**資料重新劃分**，整個過程比較複雜，且容易出錯
- 動態資料遷移問題：如果某個資料庫組壓力過大，需要將其中部分資料遷移出去，遷移過程需要總控節點整體協調，以及資料庫節點的配合。這個過程很難做到自動化，通常需要**人工處理**





## 2、“One size fits all” (一體適用全部任務) 模式很難適用於截然不同的任務場域

- 關聯模型作為統一的資料模型既被用於批次資料分析任務，也被用於線上即時任務。但這兩個任務一個強調高吞吐量，一個強調低延時，已經演化出完全不同的架構。用同一套模型來抽象顯然是不合適的
- Hadoop就是針對批次資料分析
- MongoDB、Redis等是針對線上任務，兩者都拋棄了關聯模型





- 3、關聯式資料庫的關鍵特性包括完善的交易機制和高效能的查詢機制。但是，關聯式資料庫引以為傲的兩個關鍵特性，到了Web2.0時代卻成了雞肋，除了有一些額外成本外，主要表現在以下幾個方面：
- (1) Web2.0網站系統通常不要求嚴格的資料庫交易 (如：部落格文章發表成功與否)
  - (2) Web2.0並不要求嚴格的讀寫即時性 (如：部落格文章發表是否要讓其它用戶立即看到)
  - (3) Web2.0通常不包含大量複雜的SQL查詢 (不需正規化方式設計表格，去除關聯式資料庫的嚴謹結構特性，以儲存空間換取更好的查詢性能)





# ■ NoSQL與關聯式資料庫的比較

比較標準	RDBMS	NoSQL	備註
資料庫原理	完全支援	部分支援	<ul style="list-style-type: none"><li>■ RDBMS有關聯代數理論作為基礎</li><li>■ NoSQL沒有統一的理論基礎</li></ul>
資料規模	大	超大	<ul style="list-style-type: none"><li>■ RDBMS很難實現<b>橫向擴展</b>，縱向擴展的空間也比較有限，性能會隨著資料規模的增大而降低</li><li>■ NoSQL可以很容易透過添加更多設備來支援更大規模的資料</li></ul>
資料庫模式	固定	靈活	<ul style="list-style-type: none"><li>■ RDBMS需要定義資料庫模式，嚴格遵守<b>資料定義</b>和<b>相關限制條件</b></li><li>■ NoSQL不存在資料庫模式，可以自由靈活定義並儲存各種不同類型的資料</li></ul>
查詢效率	快	可以實現高效的簡單查詢，但是不具備高度結構化查詢等特性，複雜查詢的性能不盡人意	<ul style="list-style-type: none"><li>■ RDBMS借助於<b>索引機制</b>可以實現快速查詢（包括記錄查詢和範圍查詢）</li><li>■ 很多NoSQL資料庫沒有以複雜查詢為主的索引，雖然NoSQL可以使用MapReduce來加速查詢，但是，在複雜查詢方面的性能仍然不如RDBMS</li></ul>



比較標準	RDBMS	NoSQL	備註
一致性	強一致性	弱一致性	<ul style="list-style-type: none"><li>■ RDBMS嚴格遵守交易ACID模型，可以保證交易<b>強一致性</b></li><li>■ 很多NoSQL資料庫放鬆了對交易ACID四特性的要求，而是遵守BASE模型，只能保證<b>最終一致性</b></li></ul>
資料完整性	容易實現	很難實現	<ul style="list-style-type: none"><li>■ 任何一個RDBMS都可以很容易實現完整性限制，比如透過主鍵或者非空限制來實現<b>個體完整性</b>，透過主鍵、外來鍵來實現<b>參考完整性</b>，透過限制或者觸發器來實現<b>用戶自訂完整性</b></li><li>■ 但是，在NoSQL資料庫卻無法實現</li></ul>
擴展性	一般	好	<ul style="list-style-type: none"><li>■ RDBMS很難實現<b>橫向擴展</b>，縱向擴展的空間也比較有限</li><li>■ NoSQL在設計之初就充分考慮了橫向擴展的需求，可以很容易透過添加廉價設備實現擴展</li></ul>
可用性	好	很好	<ul style="list-style-type: none"><li>■ RDBMS在任何時候都以保證資料一致性為優先目標，其次才是最佳化系統性能。隨著資料規模的增大，RDBMS為了保證嚴格的一致性，只能提供<b>相對較弱的可用性</b></li><li>■ 大多數NoSQL都能提供較高的可用性</li></ul>



比較標準	RDBMS	NoSQL	備註
標準化	是	否	<ul style="list-style-type: none"><li>■ RDBMS已經標準化 ( 有ANSI SQL )</li><li>■ NoSQL還沒有業界標準，不同的NoSQL資料庫都有自己的查詢語言，很難規範應用程式介面</li><li>■ Stone Braker認為：NoSQL缺乏統一查詢語言，將會拖慢NoSQL發展</li></ul>
技術支援	高	低	<ul style="list-style-type: none"><li>■ RDBMS經過幾十年的發展，已經非常成熟，Oracle等大型廠商都可以提供很好的技術支援</li><li>■ NoSQL在技術支援方面仍然處於起步階段，還不成熟，缺乏有力的技術支援</li></ul>
可維護性	複雜	複雜	<ul style="list-style-type: none"><li>■ RDBMS需要專門的資料庫管理員(DBA)維護</li><li>■ NoSQL資料庫雖然沒有DBMS複雜，卻也難以維護</li></ul>



## 小結

### (1) 關聯式資料庫

**優勢：**以完善的關聯代數理論作為基礎，有嚴格的標準，支援交易ACID四特性，借助索引機制可以實現高效的查詢，技術成熟，有專業公司的技術支援

**劣勢：**可擴展性較差(尤其是橫向擴展性)，無法較好地支援海量資料儲存，資料模型過於死板、無法較好地支援Web2.0應用，交易機制影響了系統的整體性能等

### (2) NoSQL資料庫

**優勢：**可以支援超大規模資料儲存，靈活的資料模型可以很好地支援Web2.0應用，具有強大的橫向擴展能力等

**劣勢：**缺乏數學理論基礎，複雜查詢性能不高，大都不能實現交易強一致性，很難實現資料完整性，企業關鍵任務不能採用。技術尚不成熟，缺乏專業團隊的技術支援，維護較困難等



關聯式資料庫和NoSQL資料庫各有優缺點，彼此無法取代

- **關聯式資料庫應用場域**：電信、銀行等領域的**關鍵任務**，需要保證強交易一致性
- **NoSQL資料庫應用場域**：互聯網企業、傳統企業的**非關鍵任務**（例如：資料分析）

### 採用混合架構

- 如：亞馬遜公司就使用不同類型的資料庫來支撐電子商務應用
  - 對於“購物籃”這種臨時性資料，採用鍵值儲存會更加有效率
  - 當前的產品和訂單資訊則適合存放在關聯式資料庫(屬關鍵任務)
  - 大量的歷史訂單資訊則適合保存在類似MongoDB的文件資料庫中

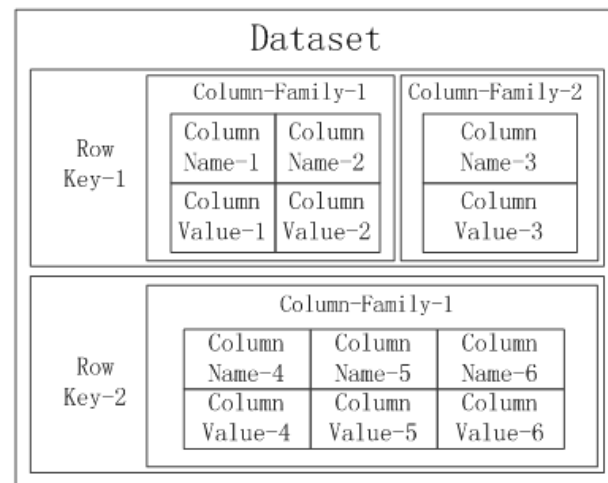


# NoSQL的四大類型

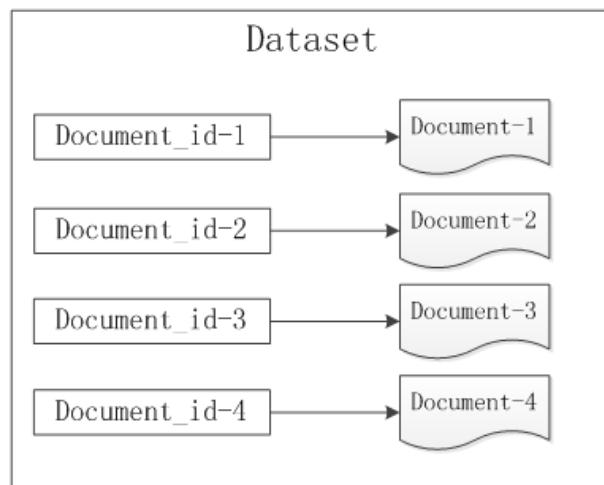
- NoSQL資料庫雖然數量眾多，但是，歸結起來，典型的NoSQL資料庫通常包括鍵值資料庫、欄族資料庫、文件資料庫和圖形資料庫

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

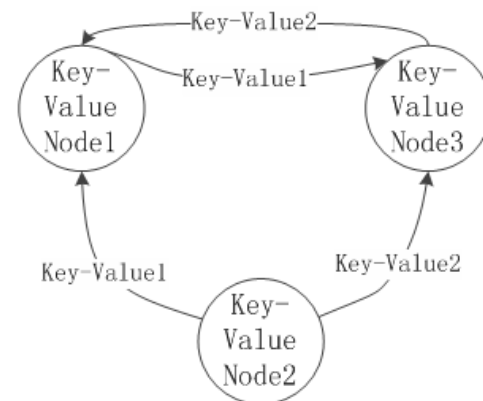
鍵值資料庫 (Key Value)



欄族資料庫 (Column Family)











文件資料庫 (Document)



圖形資料庫 (Graph)



Type	Example	
Key-Value Store	 redis	
Wide Column Store	 H-BASE	
Document Store	 mongoDB	 CouchDB relax
Graph Store	 Neo4j	 InfiniteGraph The Distributed Graph Database

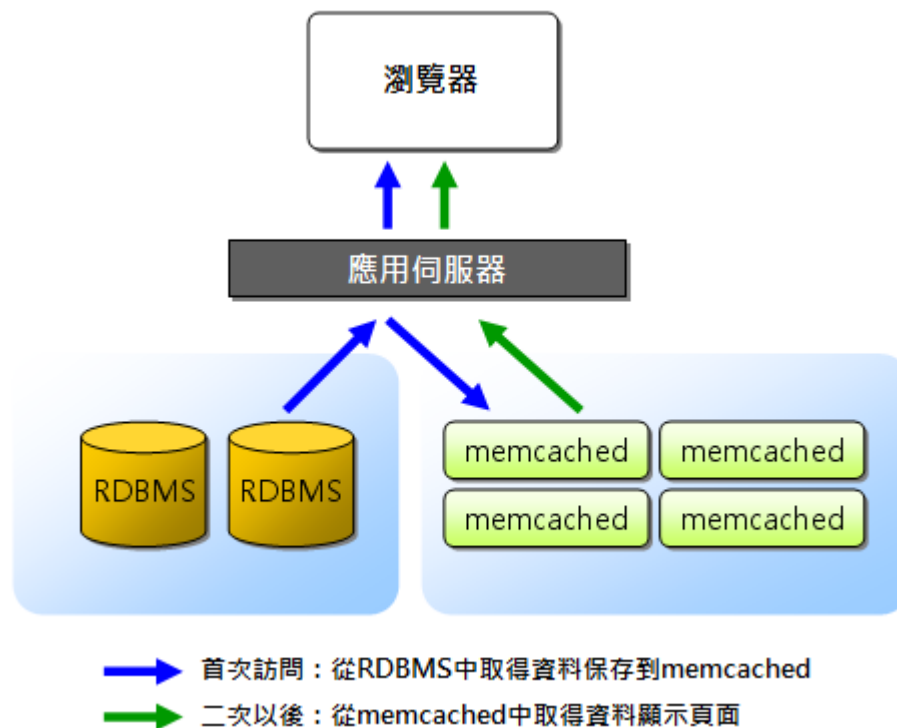


# 鍵值資料庫 (Key-value)

相關產品	■ Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached
資料模型	■ 鍵/值對 <key, value> ■ 鍵是一個字串物件 ■ 值可以是任意類型的資料，例如：整數、字元、陣列、清單、集合等
典型應用	■ 涉及頻繁讀寫、擁有簡單資料模型的應用 ■ 內容暫時性存放，例如：會話、設定檔、參數、購物車等
優點	■ 擴展性好，靈活性好，大量寫操作時性能高
缺點	■ 無法儲存結構化資訊，條件查詢效率較低
不適用情形	■ 不是透過鍵而是透過值來查詢：鍵值資料庫根本沒有透過值查詢的途徑 ■ 需要儲存資料之間的關係：在鍵值資料庫中，不能透過兩個或兩個以上的鍵來關聯資料 ■ 需要交易的支援：在一些鍵值資料庫中，產生故障時，不可以退回
使用者	■ GitHub ( Riak )、BestBuy ( Riak )、Twitter ( Redis和Memcached )、StackOverFlow ( Redis )、Instagram ( Redis )、Youtube ( Memcached )、Wikipedia ( Memcached )



- 鍵值資料庫可成為理想的緩衝層解決方案。
  - 網頁要訪問伺服器底層的關聯式資料庫硬碟上的資料時，若每次都直接訪問效能會很差；若以鍵值資料庫做為緩衝(如：Memcached或Redis)，經常使用的資料放在鍵值資料庫上，就可提升網頁訪問的效能。
- Redis有時候會被稱為“強化版的Memcached”，它支援持久化、資料恢復、更多資料型態





# 欄族資料庫

相關產品	■ BigTable、HBase、Cassandra、HadoopDB、GreenPlum、PNUTS
資料模型	■ 欄族
典型應用	<ul style="list-style-type: none"><li>■ 分散式資料儲存與管理</li><li>■ 資料在地理上分散於多個資料中心的應用程式</li><li>■ 可以容忍副本中存在短期不一致情況的應用程式</li><li>■ 擁有動態欄位的應用程式</li><li>■ 擁有潛在大量資料的應用程式，大到幾百TB的資料</li></ul>
優點	■ 尋找速度快，可擴展性強，容易進行分散式擴展，複雜性低
缺點	■ 功能較少，大都不支援強交易一致性
不適用情形	■ 需要ACID交易支援的情形，Cassandra等產品就不適用
使用者	■ ebay ( Cassandra )、Instagram ( Cassandra )、NASA ( Cassandra )、Twitter ( Cassandra and HBase )、Facebook ( HBase )、Yahoo! ( HBase )



# 文件資料庫

- “文件”其實是一個資料記錄，這個記錄能夠對包含的資料型態和內容進行“自我描述”。XML文件、HTML文件和JSON文件就屬於這一類，它儲存的資料是這樣的：

Document 1	Document 2	Document 3
<pre>{   "id": "1",   "name": "John Smith",   "isActive": true,   "dob": "1964-30-08" }</pre>	<pre>{   "id": "2",   "fullName": "Sarah Jones",   "isActive": false,   "dob": "2002-02-18" }</pre>	<pre>{   "id": "3",   "fullName":     {       "first": "Adam",       "last": "Stark"     },   "isActive": true,   "dob": "2015-04-19" }</pre>

- 關聯式資料庫：

必須有schema資訊才能理解資料的含義

學生（學號，姓名，性別，年齡，系，年級）

（1001，張三，男，20，電腦，2002）

- 一個XML文件：

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:9000/hbase
  </value>
  </property>
</configuration>
```



- 資料是不規則的，每一條記錄包含了所有的有關“John Smith”的資訊而沒有任何其它外部文件的引用，這條記錄就是“自我包含”的
- 這使得記錄很容易完全移動到其他伺服器，因為這條記錄的所有資訊都包含在裡面了，不需要考慮還有其它相關資訊在別的表沒有一起遷移走
- 同時，因為在移動過程中，只有被移動的那一條記錄（文件）需要操作，而不像關聯式中每個有關聯的表都需要鎖定來保證一致性，這樣一來讀寫的速度也會有很大的提升

#### Document 1

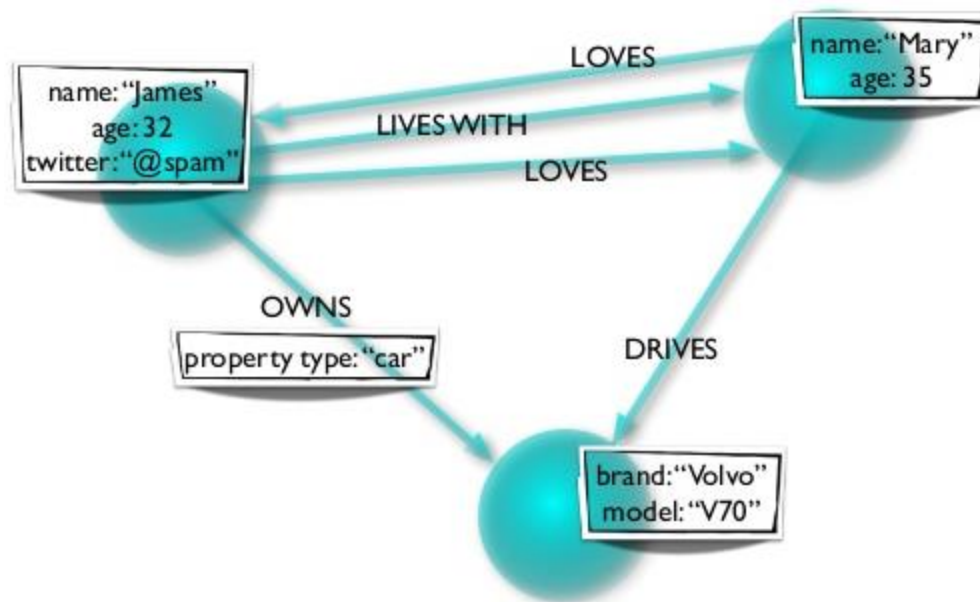
```
{  
  "id": "1",  
  "name": "John Smith",  
  "isActive": true,  
  "dob": "1964-30-08"  
}
```



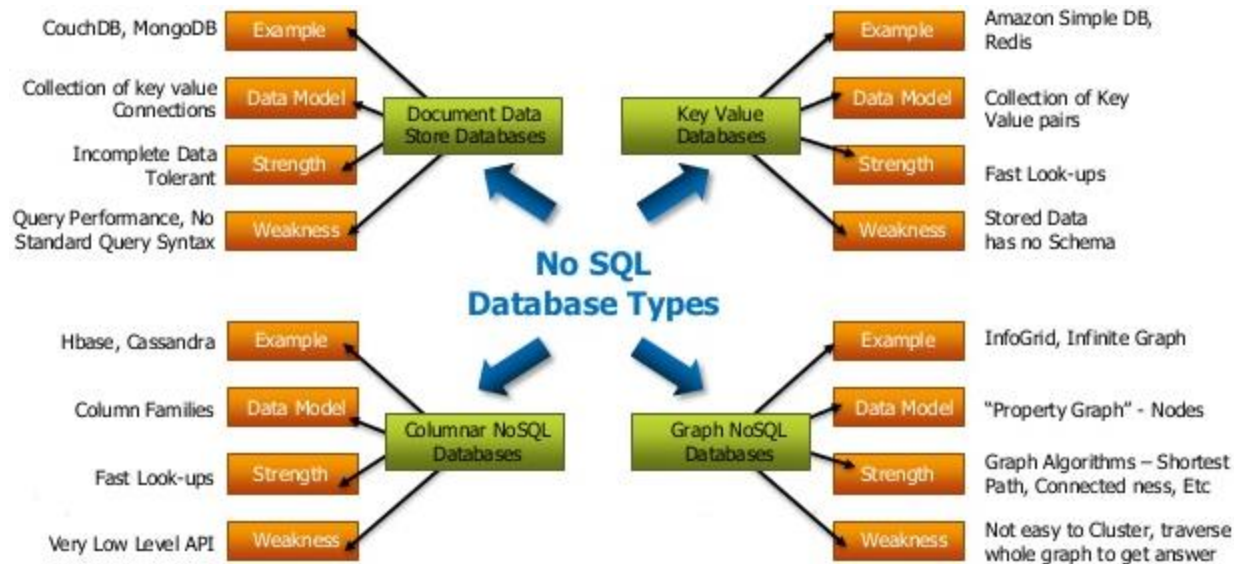
相關產品	■ MongoDB、CouchDB、Terrastore、ThruDB、RavenDB、SisoDB、RaptorDB、CloudKit、Perservere、Jackrabbit
資料模型	■ 鍵/值 ■ 值 ( value ) 是版本化的文件
典型應用	■ 專門處理以文件為主的資料或者類似的半結構化資料 ■ 比如，用於後臺具有大量讀寫操作的網站、使用JSON資料結構的應用、使用嵌套結構等非正規化資料的應用程式
優點	■ 性能好，靈活性高，複雜性低，資料結構靈活 ■ 提供嵌入式文件功能，將經常查詢的資料儲存在同一個文件中 ■ 既可以根據鍵來建構索引，也可以根據內容建構索引
缺點	■ 缺乏統一的查詢語法
不適用情形	■ 在不同的文件上執行交易。文件資料庫並不支援文件間的交易，如果對這方面有需求則不應該選用這個解決方案
使用者	■ SAP ( MongoDB )、Codecademy ( MongoDB )、Foursquare ( MongoDB )、NBC News ( RavenDB )

# 圖形資料庫

相關產品	■ Neo4J、OrientDB、InfoGrid、Infinite Graph、GraphDB
資料模型	■ 圖形結構
典型應用	■ 專門處理具有高度相互關聯關係的資料，比較適合於社交網路、模式識別、依賴分析、推薦系統以及路徑尋找等問題
優點	■ 靈活性高，支援複雜的圖形演算法，可用於建構複雜的關係圖譜
缺點	■ 複雜性高，只能支援一定的資料規模
使用者	■ Adobe ( Neo4J )、Cisco ( Neo4J )、T-Mobile ( Neo4J )



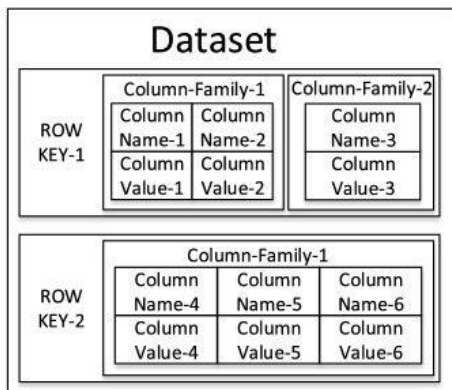
# 不同類型資料庫簡易匯整



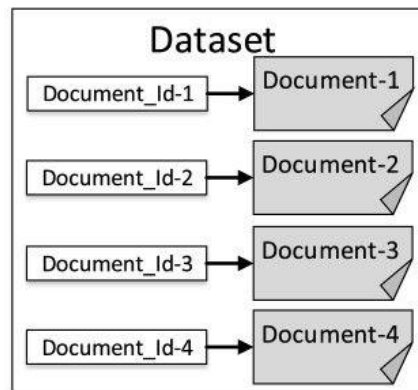
- 鍵值資料庫可視為其它類NoSQL資料庫的原型。

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

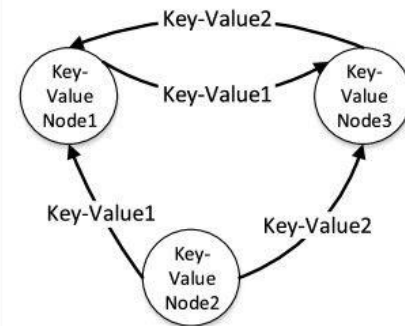
Key-Value Store



Column-Family Store



Document Store



Graph Databases





- HBase中需要根據列鍵、欄族、欄限定詞和時間戳記 “四維座標” 來確定一個儲存格。

鍵	值
[ "key001" , "column-family-1" , "column-B" , "t1" ]	"high"
[ "key002" , "column-family-2" , "column-A" , "t3" ]	"smile"

RowKey	column-family-1		column-family-2			column-family-3
	column-A	column-B	column-A	column-B	column-C	column-A
key001	t2:hk t1:jy		t4:ipad t3:ipod t1:iphone			
key002	t3:style t1:wk		t3:smile t2:jk	t2:wtf	t1:hw	
key003		t1:high				t4:powerful

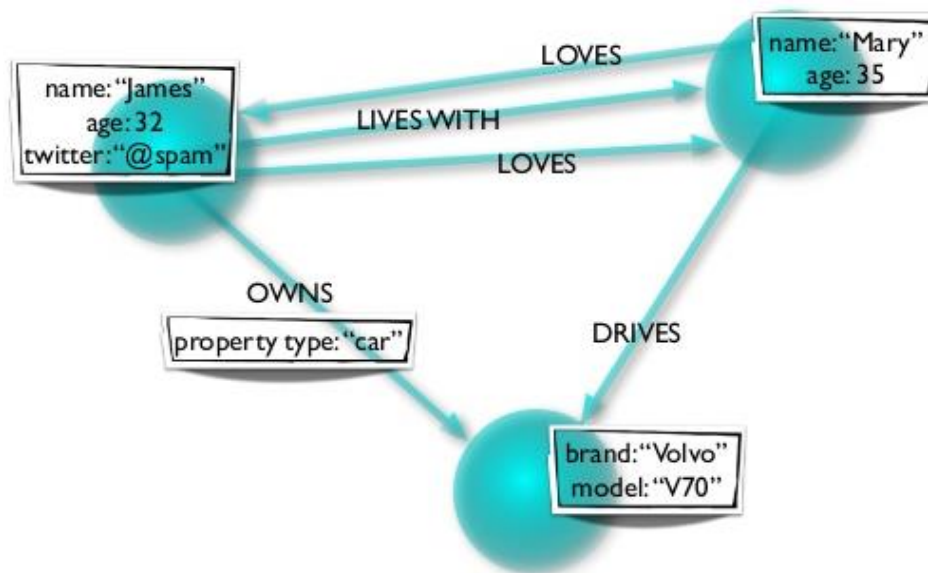


## ■ Document Database

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

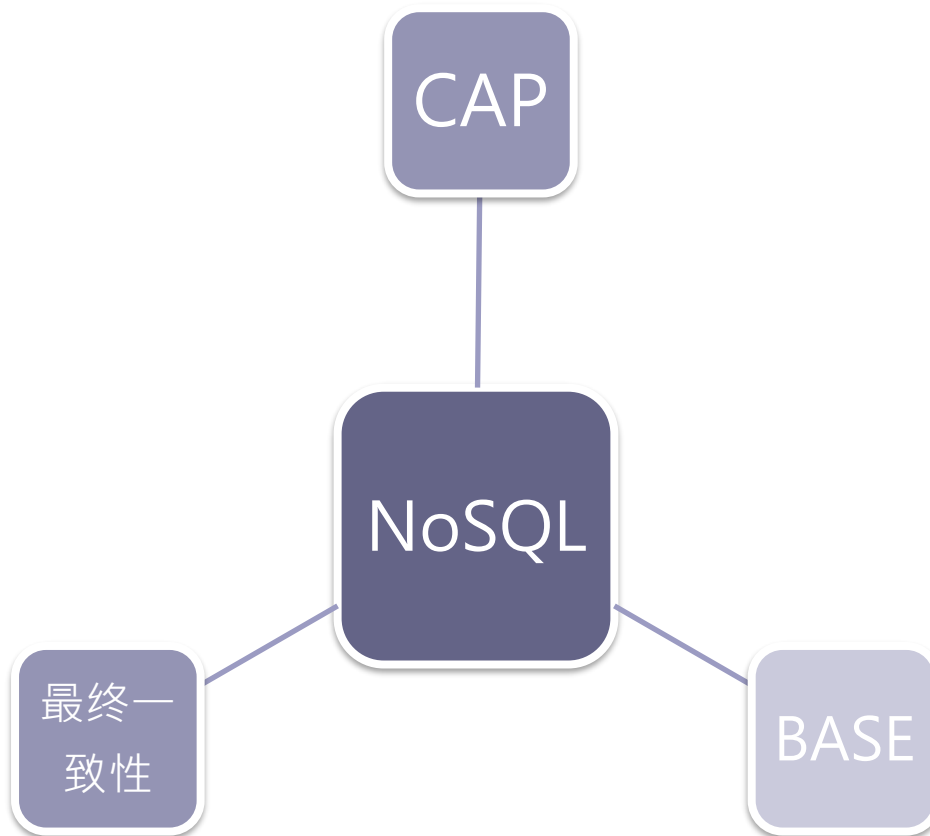
← field: value  
← field: value  
← field: value  
← field: value

## ■ Graph Database





# ■ NoSQL的三大基礎





- 談到NoSQL的特性，則不得不先談關聯式資料庫系統的ACID。一個關聯式資料庫系統的交易操作，需具備ACID四特性：

- A ( Atomicity ) : 單元性

- 交易是不可分割的完整個體，它不是全部執行，就是全部不執行。

- C ( Consistency ) : 一致性

- 資料庫的一致狀態 (Consistent State)：是指資料庫所有被儲存的資料(不論是在交易前後)，必須皆滿足資料庫所設定的相關限制，以及具有正確的結果。
  - 如果交易是全部執行，能讓資料庫從某個一致狀態，轉變到另一個一致狀態。我們則稱此次交易具有一致性。



- I (Isolation) : 孤立性

- 某交易執行期間所用的資料或中間結果，不容許其它交易讀取或寫入，直到此交易被確認 (Commit，即：成功結束) 為止。也就是說，它不應被同時執行的其它交易所干擾。

- D (Durability) : 永久性

- 一旦交易全部執行，且經過確認 (Commit) 後，其對資料庫所做的變更則永遠有效，即使未來系統當機或毀損。





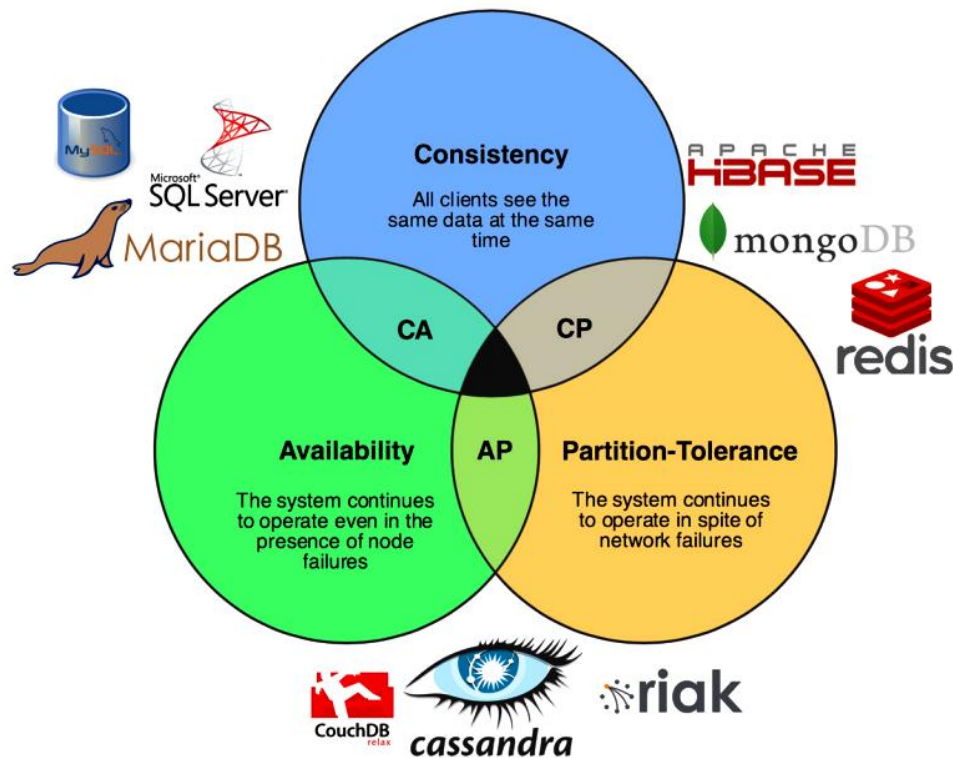
# CAP

所謂的CAP指的是：

- **C (Consistency)**：一致性，是指任何一個讀操作總是能夠讀到**正常完成之寫操作結果**。而在分散式環境下，多個節點的資料副本是否一致的，或者說，**使用者讀取某資料的任一副本是否皆相同**
- **A: (Availability)**：可用性，是指**快速獲取資料**，可以在**有限時間內返回操作結果**，保證每個請求不管成功或者失敗都有回應；
- **P (Tolerance of Network Partition)**：分區容忍性，是指當出現系統中的一部分節點無法正常營運、和其他節點進行通信時，分離的系統也能夠正常運行。也就是說，**系統中任意資訊的遺失或節點失敗不會影響系統整體的繼續運作**。



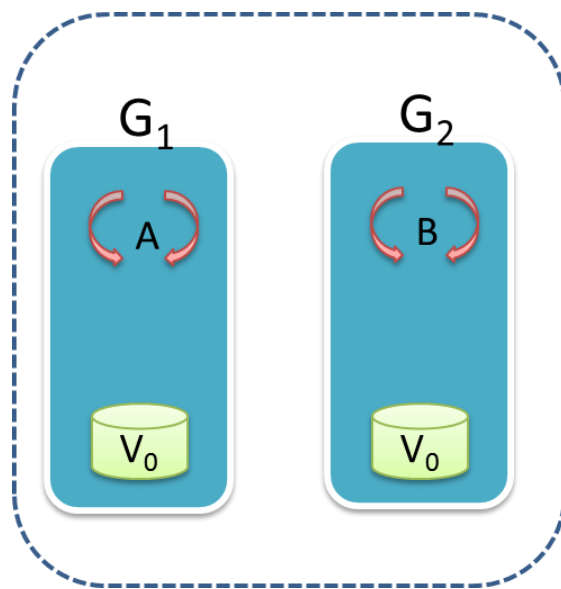
- CAP理論告訴我們，一個分散式系統不可能同時滿足一致性、可用性和分區容忍性這三個需求，最多只能同時滿足其中兩個，正所謂“魚和熊掌不可兼得”。
- 在分散式系統環境下，**P** 一定要被滿足。





## • 一個犧牲一致性來換取可用性的實例

- a) 在網路中有兩個節點分別為 $G_1$ 和 $G_2$ ，這兩個節點上存儲著同一資料的不同副本，現在資料是一致的，其值皆為 $V_0$ ，A、B分別是運行在 $G_1$ 、 $G_2$ 上與資料互動的應用程式。

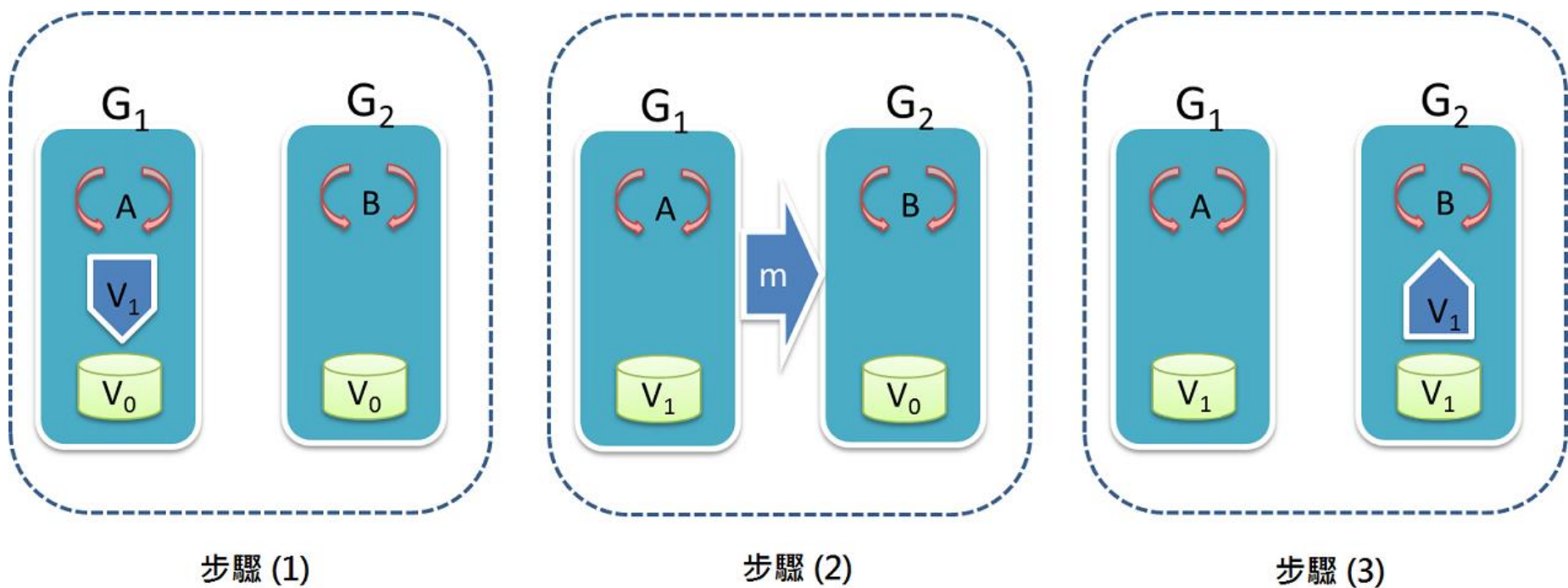


a) 初始狀態



b) 在正常情況下，操作過程如下(如下圖所示)：

- (1) A將 $V_0$ 更新，資料值為 $V_1$ ;
- (2)  $G_1$ 發送更新消息 $m$ 給副本 $G_2$ ，資料 $V_0$ 更新為 $V_1$ ;
- (3) B讀取到 $G_2$ 中的資料 $V_1$ 。

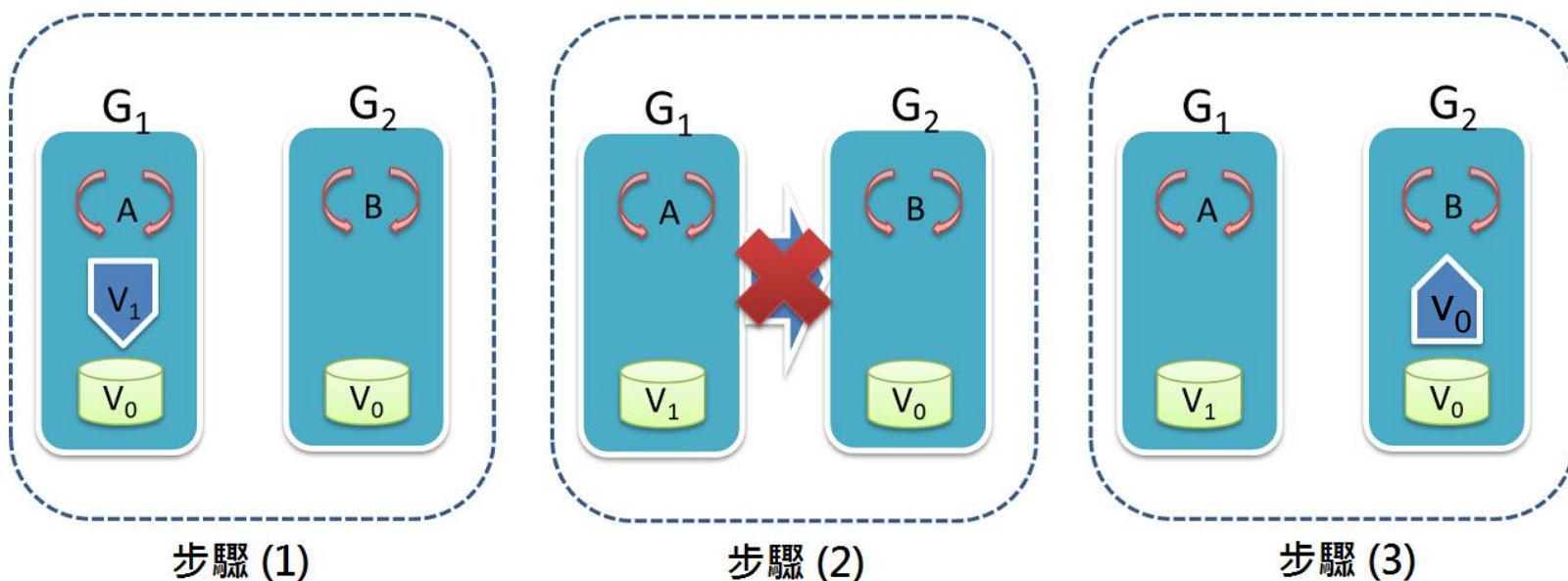


b) 正常執行過程



c) 如果發生網路故障：

- 那麼在操作的步驟(2)將發生錯誤： $G_1$ 發送的消息不能傳送到 $G_2$ 上。
- 這樣資料就處於不一致的狀態，B讀取到的就不是最新的資料。
- 如果我們採用一些技術，如：阻塞、加鎖、集中控制...等，來保證資料的一致，那麼必然會影響到系統的可用性。
- 所以無法同時滿足三點，總是需要放棄一部分。



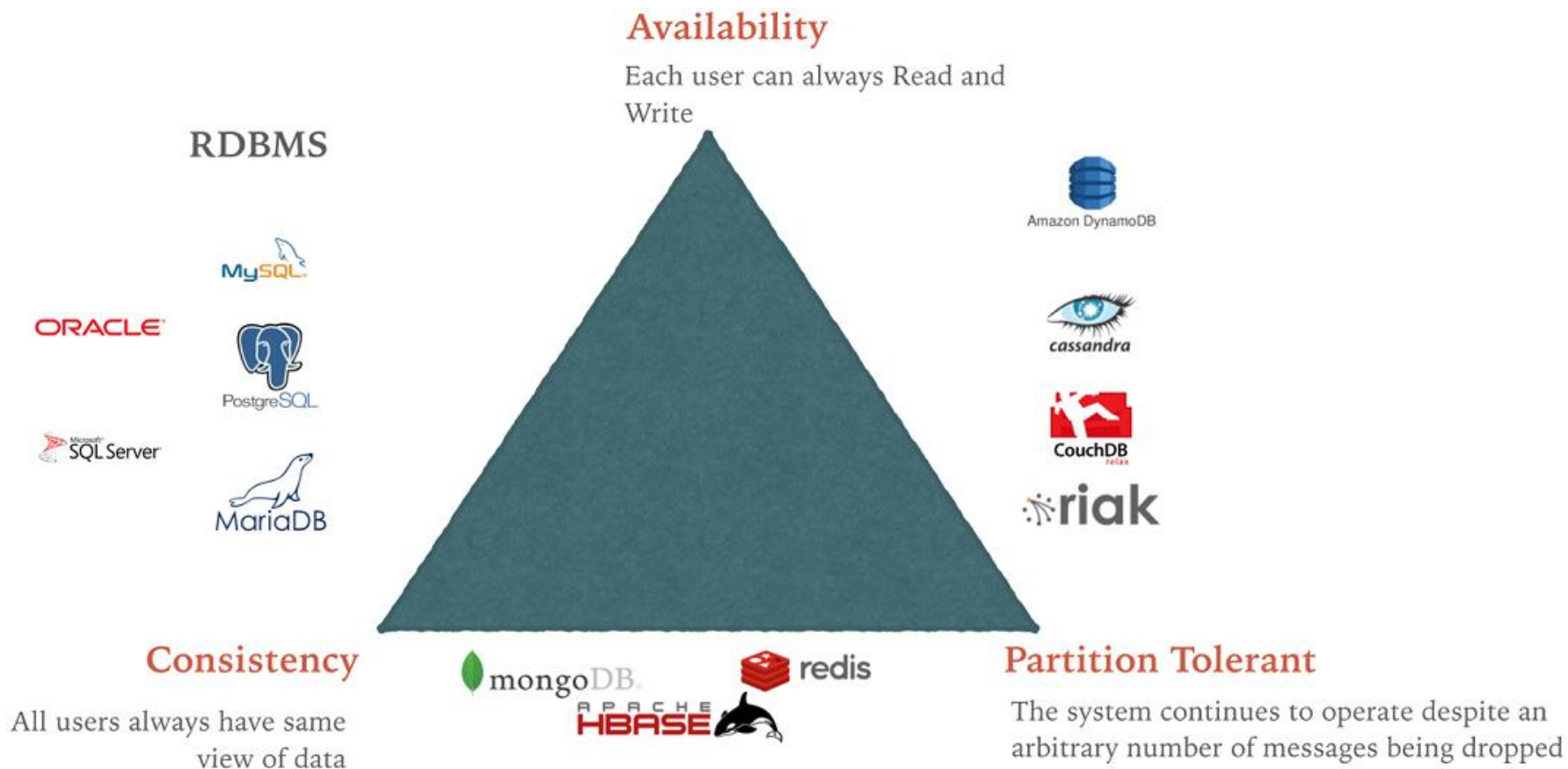
c) 傳送失敗的執行過程



當處理CAP的問題時，可以有幾個明顯的選擇：

1. **CA**：也就是強調**一致性 (C)** 和 **可用性 (A)**，放棄分區容忍性 (P)，最簡單的做法是把所有與交易相關的內容都**放到同一台機器上**。很顯然，這種做法會嚴重影響系統的可擴展性。傳統的關聯式資料庫 (MySQL、SQL Server 和 PostgreSQL)，都採用了這種設計原則，因此，擴展性都比較差
2. **CP**：也就是強調**一致性 (C)** 和 **分區容忍性 (P)**，放棄可用性 (A)，當出現網路磁碟分割的情況時，受影響的服務需要**等待資料一致**，因此在等待期間就無法對外提供服務
3. **AP**：也就是強調**可用性 (A)** 和 **分區容忍性 (P)**，放棄一致性 (C)，允許系統返回不一致的資料
  - AP 為多數商業網站的需求





不同產品在CAP理論下的不同設計原則



# BASE

- BASE (Basically Available, Soft-state, Eventual consistency) 是對 CAP 中 C 和 A 的延伸：
  - **Basically Available**：基本可用；
  - **Soft-state**：軟狀態/柔性交易，即狀態可以有一段時間的不同步；
  - **Eventual consistency**：最終一致性；
- BASE 是基於 CAP 理論逐步演化而來，核心思想是：即便不能達到「強一致性」(Strong consistency)，但可以根據應用特點採用適當的方式來達到「最終一致性」(Eventual consistency) 的效果。



BASE的基本含義是基本可用（Basically Available）、軟狀態（Soft-state）和最終一致性（Eventual consistency）：

- 基本可用

基本可用，是指一個分散式系統的一部分發生問題變得不可用時，其他部分仍然可以正常使用，也就是允許分區失敗的情形出現

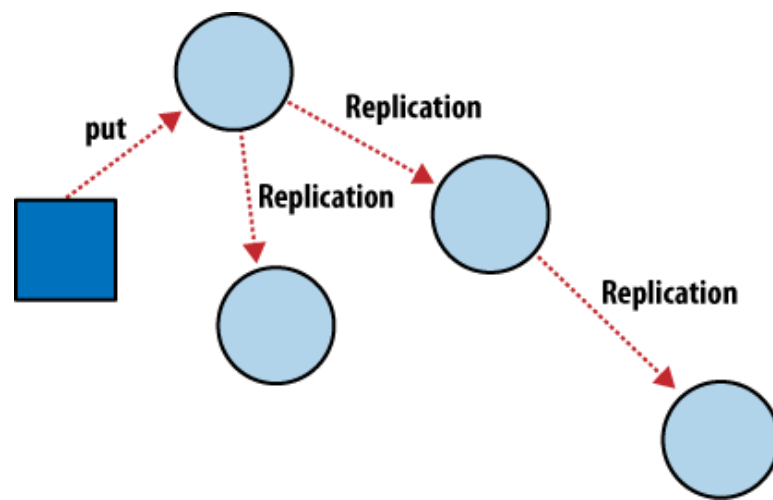
- 軟狀態

“軟狀態（soft-state）”是與“硬狀態（hard-state）”相對應的一種說法。資料庫保存的資料是“硬狀態”時，可以保證資料一致性，即保證資料一直是正確的。“軟狀態”是指狀態可以有一段時間不同步，具有一定的滯後性



## ● 最終一致性

- 一致性的類型包括強一致性和弱一致性，二者的主要區別在於分散式多副本的資料更新操作下，後續操作是否能夠獲取最新的資料。
  - 對於強一致性而言，當執行完一次更新操作後，可以保證後續的其他讀操作不論是讀到哪一個資料副本，都是可立即讀到更新後的最新資料；
  - 反之，如果不能保證後續訪問可立即讀到更新後的最新資料，就是弱一致性。
- 最終一致性只不過是弱一致性的一種特例，允許後續的訪問操作可以暫時讀不到更新後的資料，但是經過一段時間之後，必須最終讀到更新後的資料。





# 最終一致性

- 探討各類型最終一致性之前，先假設一個接下來會用到的對應範例：
  - 假設老闆要幫全公司的員工逐一進行不同程度的加薪，在老闆逐一加薪完、尚未公告在公佈欄之前 (即：**滯後性**)，公司所有人是不知道自己的調薪情況，但是**最終**大家還是都會知道的。
  - 此公司只會加薪/不調薪，不會減薪...
- 最終一致性根據更新資料後各行程訪問到資料的**時間**和**方式**的不同，又可以區分為：
  - 因果一致性(Causal consistency)
  - 讀己之所寫一致性(Read-your-writes consistency)
  - 單調讀一致性 (Monotonic read consistency)
  - 會話一致性 (Session consistency)
  - 單調寫一致性(Monotonic write consistency)



## ●因果一致性：

- 如果行程A通知行程B它已更新了一個資料項目，那麼行程B的後續訪問將獲得A寫入的最新值
- 而與行程A無因果關係的行程C之資料訪問，仍然遵守一般的最終一致性規則
- 對應範例：
  - 老闆在所有人加薪過程完成、尚未公佈在公告欄之前，要先告訴老闆娘其加薪幅度。而公司其他人還是要等到最終公告在公佈欄那天，才會得知自己的薪資最新情況...



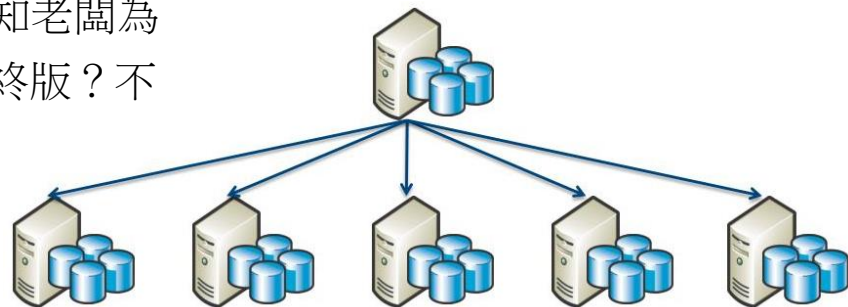


## ●讀己之所寫一致性：

- 當行程A自己執行一個更新操作之後，它自己總是可以訪問到更新過的值，絕不會看到舊值
- 對應範例：
  - 老闆自己當然看得到他幫員工加薪後的最新員工薪資情況

## ●單調讀一致性：

- 如果行程已經看到過資料物件的某個值，那麼任何後續訪問都不會返回在那個值之前的值(不會讀到更舊的值...)
- 對應範例：
  - 某員工C (小舅子?)不小心從側面得知老闆為自己的調薪狀態 (但是不是真正的最終版? 不一定!!)





## ●會話一致性：

- 當行程A在某一次的會話（session）進行資料更改，只要該會話還存在，系統就保證其**讀己之所寫一致性**。但如果因某些失敗情況令該會話終止，則行程A就要建立新的會話，而先前的保證也不會延續到新的會話

### ●對應範例：

- 老闆在此次登入系統修改員工薪資時，自己當然看得到他幫員工加薪後的最新員工薪資情況；但若是不小心手殘或是跳電造成系統跳脫，重新登入後一切就重新來吧...

## ●單調寫一致性：

- 系統保證**來自同一個行程的寫操作循序執行**。系統必須保證這種程度的一致性，否則就非常難以程式設計了

### ●對應範例：

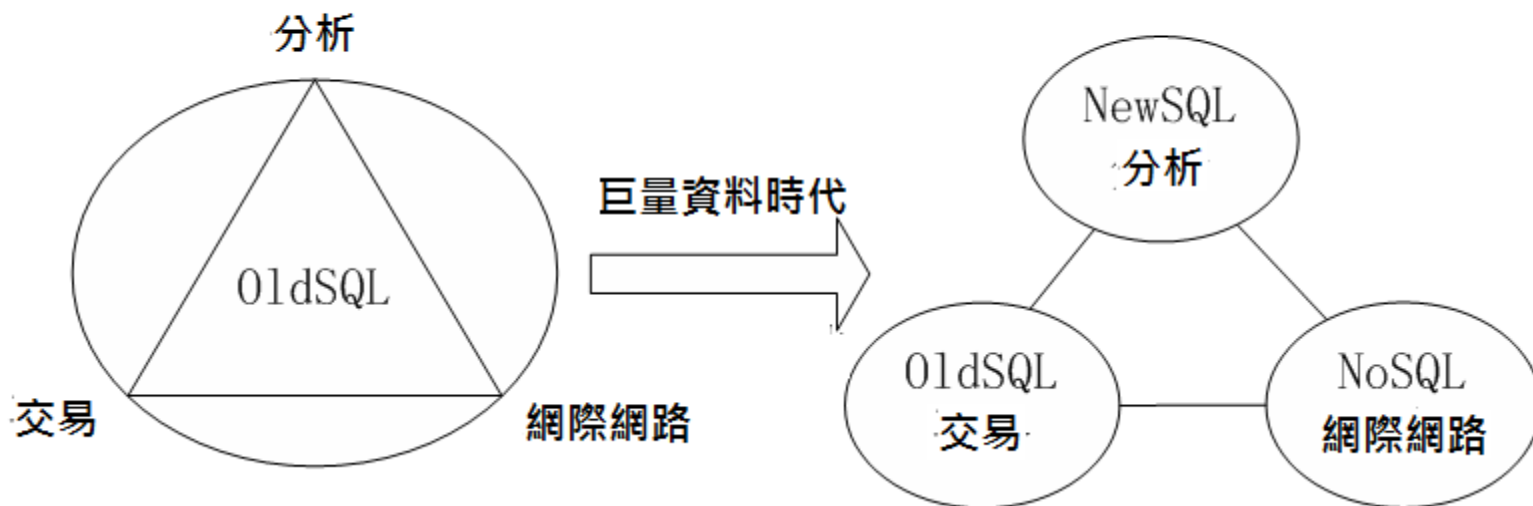
- 每個老闆都有自己固定的調薪方式 (如：自己上調50% -> 老闆娘上調20% -> 小舅子等重量級人士上調2% -> 其他中高階主管上調1% -> 其餘人等0.01%)



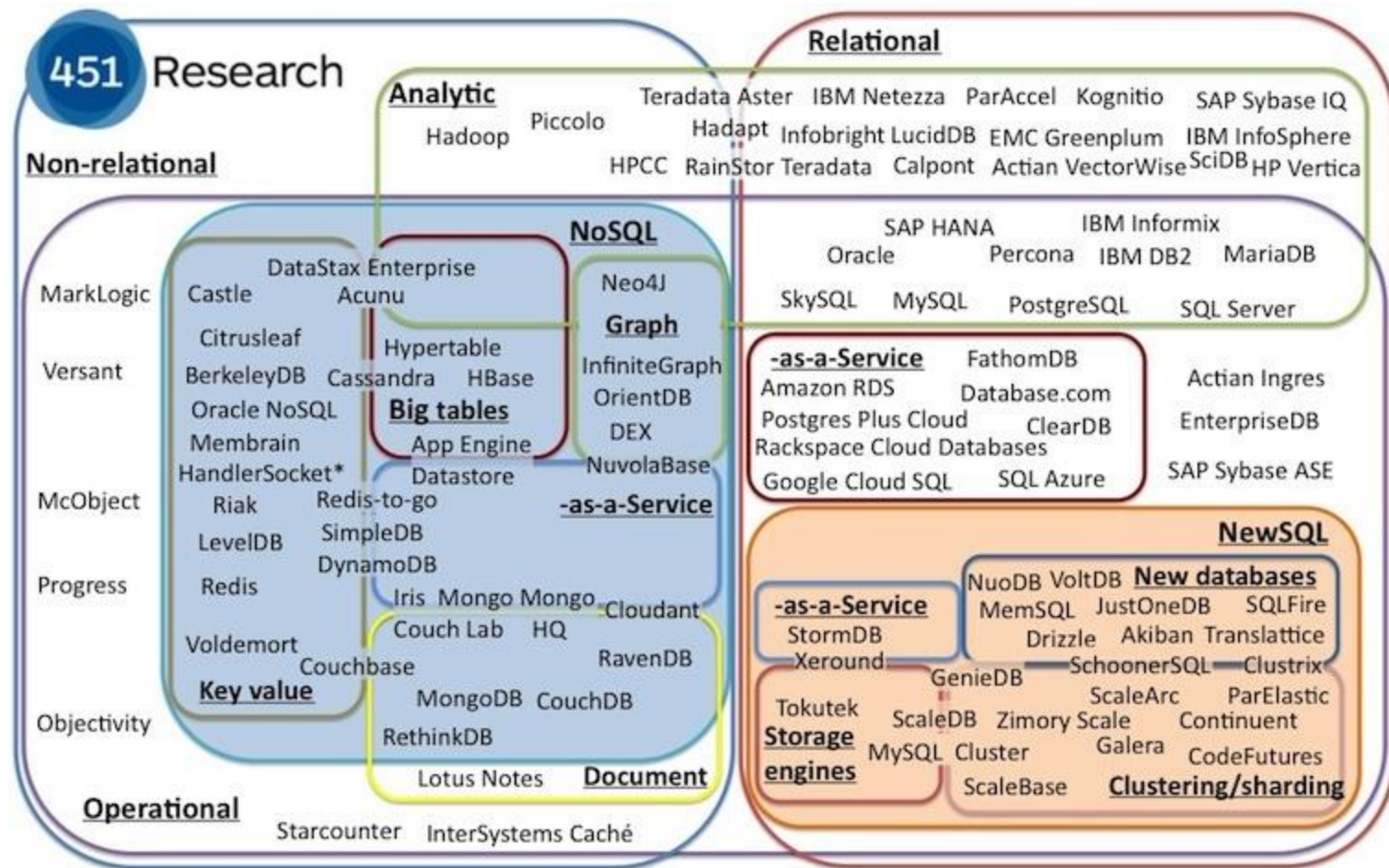
# 從NoSQL到NewSQL資料庫

(One Size Fits All)

多重架構支援多類應用



	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No



關聯式資料庫、NoSQL和NewSQL資料庫產品分類圖



# ■ Redis DB與MongoDB

- Redis簡介、安裝與簡易操作提點
- MongoDB簡介、安裝與簡易操作提點

在此僅做簡要說明，詳細安裝與使用請見本教材線上講義：

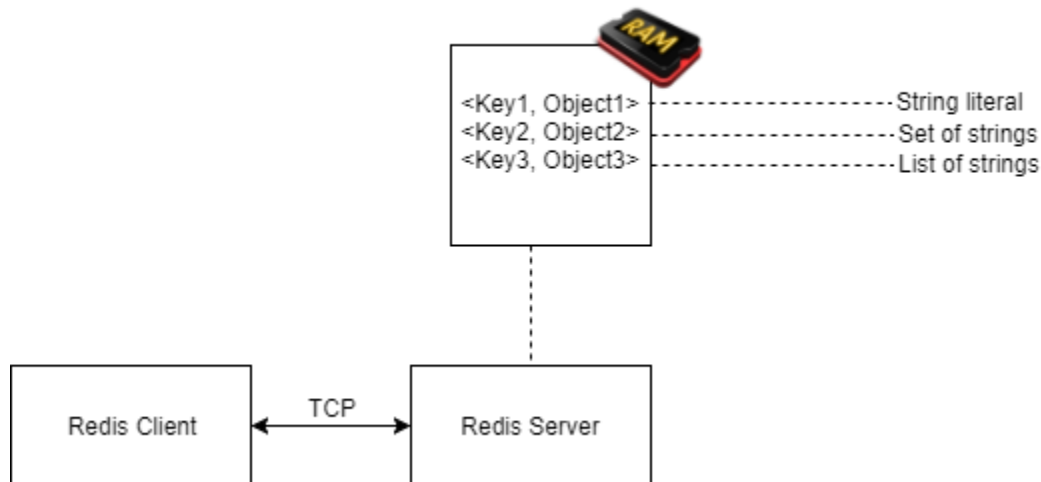
- 環境設定-單元04：其它NoSQL資料庫之安裝配置
- 實務操作-單元05：其它NoSQL資料庫系統簡易操作

課程網頁：[http://debussy.im.nuu.edu.tw/sjchen/BigData\\_final.html](http://debussy.im.nuu.edu.tw/sjchen/BigData_final.html)

教師網頁：<http://web.nuu.edu.tw/~sjchen>

# Redis DB簡介

- Redis 是一個 in-memory 的 key-value database，因此常常被用在需要快取 (Cache) 的一些場合，可以減輕許多後端資料庫的壓力。
  - 性能極高 – Redis能讀的速度是110000次/s,寫的速度是81000次/s。
  - 豐富的資料型態– Redis支援二進制的Strings, Lists, Hashes, Sets 及 Ordered Sets資料型態操作。
  - 單元性– Redis的所有操作都是具單元性的，意思就是要麼成功執行要麼失敗完全不執行。單一個操作是單元性。多個操作也支援交易，即單元性，透過MULTI和EXEC指令包起來。








# 安裝Redis

- 在Ubuntu作業系統對所下載的檔案redis-4.0.2.tar.gz進行解壓縮，並將解壓縮後的結果存放至/usr/local這個目錄之下。
- 將解壓縮後的Redis檔案夾名稱，由redis-4.0.2更名為redis，以方便後續的使用。
- 接著將redis目錄的權限賦予給hadoop使用者。
- 進入/usr/local/redis目錄以編輯與安裝Redis。
- 透過以下的命令可開啟Redis伺服器，以確認Redis是否安裝成功。

```
hadoop@master: /usr/local/redis
hadoop@master:/usr/local/redis$ cd /usr/local/redis
hadoop@master:/usr/local/redis$ ./src/redis-server
8909:C 07 Jan 00:18:14.225 # 000000000000 Redis is starting 000000000000
8909:C 07 Jan 00:18:14.225 # Redis version=4.0.2, bits=64, commit=00000000, modified=0, pid=8909, just started
8909:C 07 Jan 00:18:14.225 # Warning: no config file specified, using the default config. In order to specify a config file use ./src/redis-server /path/to/redis.conf
8909:M 07 Jan 00:18:14.227 * Increased maximum number of open files to 10032 (it was originally set to 1024).
```



```
Redis 4.0.2 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 8909

http://redis.io
```



# 使用Redis Shell訪問Redis

- 使用Redis Shell相關指令操作資料之前，需要先啟動Redis伺服器。

```
$ redis-server
```

- 接著，再開啟Redis Shell的操作環境。

```
$ redis-cli
```

- 上述兩指令執行後，會得到以下畫面：

The screenshot shows a terminal window with two overlapping windows. The background window is titled 'hadoop@master: ~' and displays the Redis server startup process, including the Redis logo and the URL 'http://redis.io'. The foreground window is also titled 'hadoop@master: ~' and shows the Redis CLI prompt '127.0.0.1:6379>'. A red label 'Redis伺服器端' points to the background window, and another red label 'Redis客戶端' points to the foreground window.

```
hadoop@master: ~  
12032:M 0  
enforced  
12032:M 0  
12032:M 0  
save may  
emory = 1  
rcommit_m  
12032:M 0  
ort enabl  
h Redis.  
ent hugep  
in the se  
12032:M 0
```



- Redis資料庫系統是以<key, value>的形式儲存資料。key和value的表示格式如下：

key = 表格名:主鍵名:欄名

value = 欲插入的欄值

- 在終端機畫面輸入set指令，以插入資料到某一個欄

```
> set key value
```

範例如下：

```
hadoop@master: ~  
hadoop@master:~$ redis-cli  
127.0.0.1:6379> set student:s96113106:name Ball  
OK  
127.0.0.1:6379> set student:s96113106:chinese 30  
OK  
127.0.0.1:6379> set student:s96113106:english 40  
OK  
127.0.0.1:6379> set student:s96113106:math 50  
OK  
127.0.0.1:6379> 
```



# MongoDB簡介

- MongoDB是由C++語言編寫的，是一種強大，靈活、且易於擴展的文件導向式(document-oriented)資料庫，與傳統的關聯式導向資料庫相比，它不再有row的概念，取而代之的是document的概念。
- MongoDB將資料儲存為一個文件，資料結構由鍵值(key-value)對組成。MongoDB文件類似於JSON對象。欄位值可以包含其他文件，陣列及文件陣列。

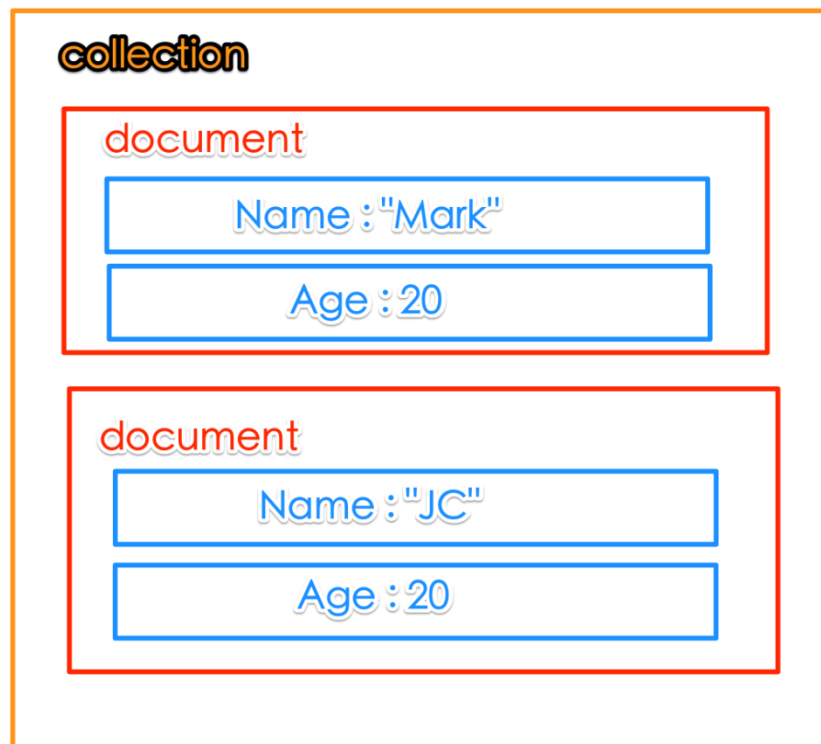
```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value  
← field: value  
← field: value  
← field: value



## • MongoDB的組成：Document與Collection

- Document是mongodb的核心，它就是Key對應個Value組合
- Collection就是一組Document，如果把它用來與關聯式資料庫比較，他就是Table裡面存放了很多Row





- 透過下圖實例，我們也可以更直觀的的瞭解MongoDB中的一些概念：

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas



```
{
  "_id": ObjectId("5146bb52d8524270060001f3"),
  "age": 25,
  "city": "Los Angeles",
  "email": "mark@abc.com",
  "user_name": "Mark Hanks "
}
{
  "_id": ObjectId("5146bb52d8524270060001f2"),
  "age": 31,
  "city": "Dallas",
  "email": "richard@abc.com",
  "user_name": "Richard Peter "
}
```



SQL術語/概念	MongoDB術語/概念	解釋/說明
database	database	資料庫
table	collection	資料庫表/集合
row	document	資料記錄列/文件
column	field	資料欄位/域
index	index	索引
table joins		表連接,MongoDB不支援
primary key	primary key	主鍵,MongoDB自動將_id欄位設置為主鍵





## MongoDB資料型態

資料型態	描述
String	字串。儲存資料常用的資料型態。在MongoDB中，UTF-8編碼的字串才是合法的。
Integer	整型數值。用於儲存數值。根據你所採用的伺服器，可分為32位或64位。
Boolean	布林值。用於儲存布林值 ( 真/假 ) 。
Double	雙精度浮點值。用於儲存浮點值。
Min/Max keys	將一個值與BSON ( 二進位的JSON ) 元素的最低值和最高值相對比。
Arrays	用於將陣列或清單或多個值儲存為一個鍵。
Timestamp	時間戳記。記錄文件修改或添加的具體時間。
Object	用於內嵌文件。
Null	用於建立空值。
Symbol	符號。該資料型態基本上等同於字串類型，但不同的是，它一般用於採用特殊符號類型的語言。
Date	日期時間。用UNIX時間格式來儲存當前日期或時間。你可以指定自己的日期時間：建立Date物件，傳入年月日資訊。
Object ID	對象ID。用於建立文件的ID。
Binary Data	二進位資料。用於儲存二進位資料。
Code	程式碼類型。用於在文件中儲存JavaScript程式碼。
Regular expression	規則運算式類型。用於儲存規則運算式。



# 安裝MongoDB

- MongoDB在Linux環境下的安裝很簡單，可以不需要下載安裝檔
- 不同版本的Ubuntu，安裝過程中，所用到的部份安裝指令也略有不同，請參考MongoDB官網([Ubuntu版安裝過程](#))，本講義使用Ubuntu 16.04版的指令。
- 由於apt-get會在新版本的MongoDB可用時逕行升級，為了防止意外的升級而造成不可預期的問題，需執行一些命令，以固定當前安裝的版本做為MongoDB的版本。
- 可透過查詢版本的方式檢視是否安裝成功：

```
hadoop@master: ~  
hadoop@master:~$ mongo -version  
MongoDB shell version v3.6.1  
git version: 025d4f4fe61efd1fb6f0005be20cb45a004093d1  
OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016  
allocator: tcmalloc  
modules: none  
build environment:  
  distmod: ubuntu1604  
  distarch: x86_64  
  target_arch: x86_64  
hadoop@master:~$
```



# 使用MongoDBshell訪問MongoDB

- 使用MongoDB的Shell相關指令操作資料之前，需要先啟動MongoDB伺服器。

```
$ sudo service mongod start
```

- 接著，再開啟MongoDB Shell的操作環境。

```
$ mongo
```

- 上述兩指令執行後，會得到以下畫面：

```
hadoop@master:~$ sudo service mongod start
[sudo] password for hadoop:
hadoop@master:~$ mongo
MongoDB shell version v3.6.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.1
Server has startup warnings:
2018-02-01T19:04:20.995+0800 I STORAGE [initandlisten]
2018-02-01T19:04:20.995+0800 I STORAGE [initandlisten] ** WARNING: Using the XFS
filesystem is strongly recommended with the WiredTiger storage engine
2018-02-01T19:04:20.995+0800 I STORAGE [initandlisten] ** See http://d
ochub.mongodb.org/core/prodnotes-filesystem
2018-02-01T19:04:21.390+0800 I CONTROL [initandlisten]
2018-02-01T19:04:21.390+0800 I CONTROL [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2018-02-01T19:04:21.390+0800 I CONTROL [initandlisten] ** Read and wri
te access to data and configuration is unrestricted.
2018-02-01T19:04:21.390+0800 I CONTROL [initandlisten]
> □
```



- MongoDB建立資料庫

MongoDB建立資料庫的語法格式如下：

**use** DATABASE\_NAME

如果資料庫不存在，則建立資料庫，否則切換到指定資料庫。

- 如果你想查看所有資料庫，可以使用 **show dbs** 命令

- 建立集合(Collection, 即：表格)

MongoDB沒有單獨建立集合名的shell命令，在插入資料的時候，MongoDB會自動建立對應的集合。



## • MongoDB 插入文件

- 文件的資料結構和JSON基本一樣。
- 所有儲存在集合中的資料都是BSON格式。
  - BSON是一種類JSON的一種二進位形式的儲存格式,簡稱Binary JSON。
- MongoDB使用insert()或save()方法向集合中插入文件，語法如下：

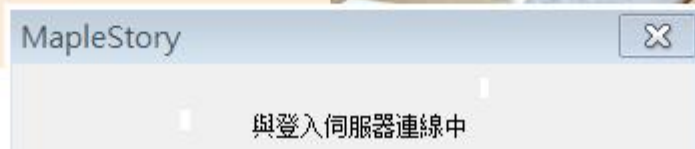
`db.COLLECTION_NAME.insert(document)`

```
hadoop@master: ~  
> db.student.insert({_id:'s96113106', name:'Ball', chinese:30, english:40, math:50})  
WriteResult({ "nInserted" : 1 })  
> db.student.insert({_id:'s96113107', name:'Tom', chinese:60})  
WriteResult({ "nInserted" : 1 })  
> db.student.insert({_id:'s96111234', name:'EQ', english:95, math:100})  
WriteResult({ "nInserted" : 1 })  
>
```



## ■ 本章小結

- 本章介紹了NoSQL資料庫的相關知識
- NoSQL資料庫較好地滿足了巨量資料時代的各種非結構化資料的儲存需求，開始得到越來越廣泛的應用。但是，需要指出的是，傳統的關聯式資料庫和NoSQL資料庫各有所長，彼此都有各自的市場空間，不存在一方完全取代另一方的問題，在很長的一段時期內，二者都會共同存在，滿足不同應用的差異化需求
- NoSQL資料庫主要包括鍵值資料庫、欄族資料庫、文件型資料庫和圖形資料庫等四種類型，不同產品都有各自的應用場合。CAP、BASE和最終一致性是NoSQL資料庫的三大理論基礎，是理解NoSQL資料庫的基礎
- 介紹了融合傳統關聯式資料庫和NoSQL優點的NewSQL資料庫
- 本章最後介紹了Redis DB和MongoDB



-本單元結束-  
感謝您的聆聽