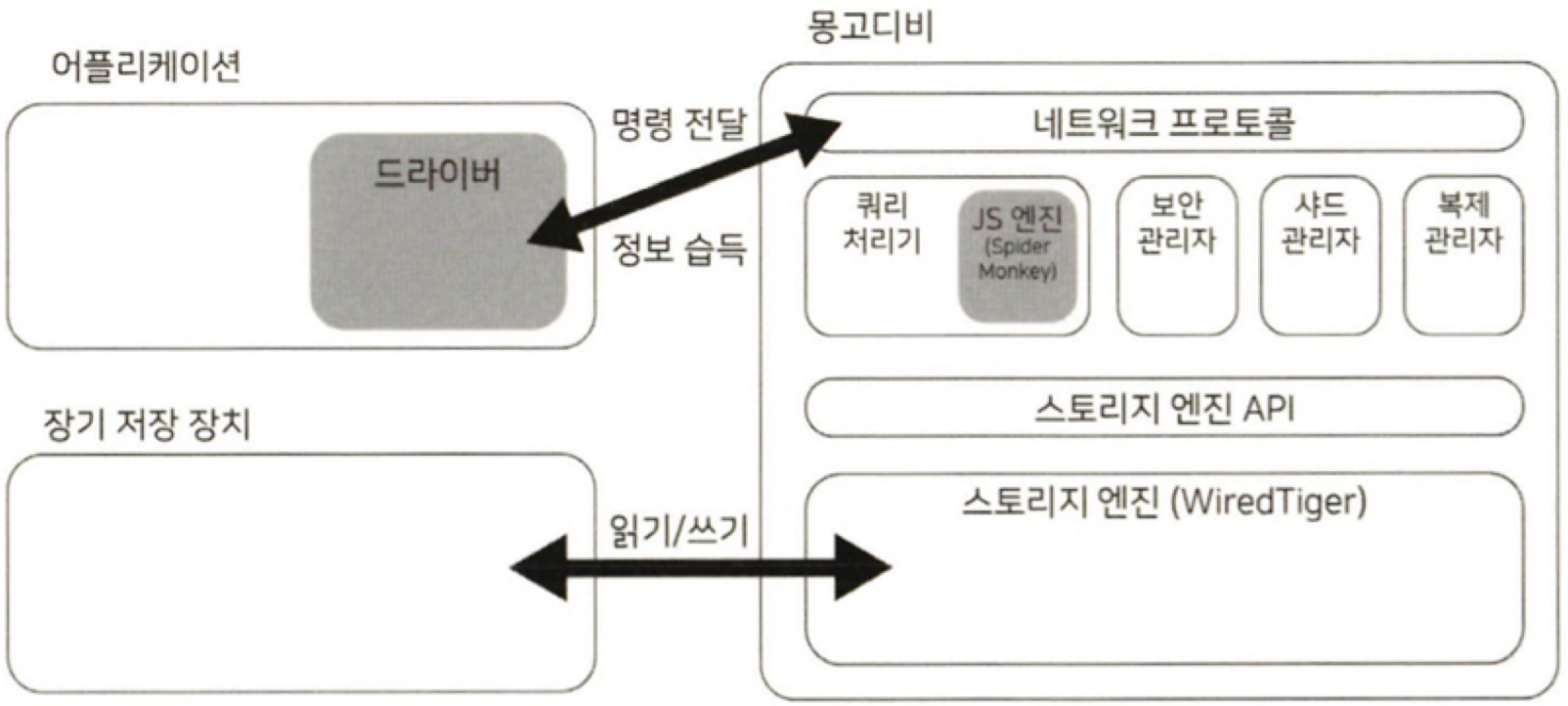
Mongo DB Architecture

Mongo DB

❖ Mongo DB Architecture



Mongo DB

- Mongo DB JSON Document – Binary JSON
  - ✓ 장점
    - Light Weight
      - 각 필드의 값을 단순히 문자열만으로 저장하는 것이 아니라 정수와 부동 소수점 그리고 날짜 등과 같이 이진 데이터 타입을 이용해서 데이터를 저장
      - 문자열 대신에 숫자나 날짜로 저장하면 메모리를 절약할 수 있어서 데이터를 디스크 파일로 저장하거나 네트워크로 전송할 때 가볍고 효율적으로 처리 가능
    - Traversable: 각 필드는 항상 데이터 타입과 필드 값의 길이가 먼저 저장돼 있기 때문에 복잡한 처리 과정 없이 불필요한 필드는 건너뛰고 필요한 필드만 빠르게 찾아갈 수 있게 저장
    - Efficient: 기본 데이터 타입으로 C언어의 Primitive 타입을 사용하고 있기 때문에 어떤 개발 언어에서도 매우 빠르게 Encoding 및 Decoding

Mongo DB

- Mongo DB JSON Document – Binary JSON
  - ✓ 데이터 타입
    - Double
    - String
    - Object
    - Array
    - Binary Data
    - ObjectId: 12Byte 의 Binary Data 로 Mongo DB의 Primary Key 인 _id 필드의 값으로 주로 사용
    - Boolean
    - Date, Timestamp
    - Null
    - Regular Expression
    - JavaScript, JavaScript(with scope)
    - Symbol
    - 32bit Integer, 64bit Integer
    - Decimal128: 정확한 숫자(3.4 버전에서 도입)
    - Min Key, Max Key

Mongo DB

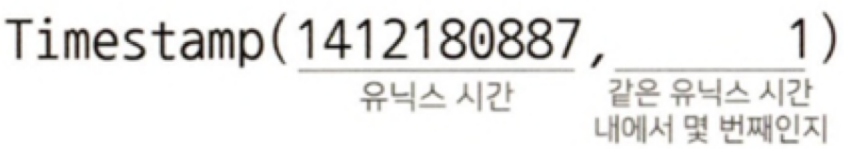
❖ Mongo DB JSON Document – Binary JSON

✓ 데이터 타입

타입	설명	종류	예시
Null	"아무것도 없다"는 의미	null	null
정수형	정수 값을 가진다.	int, long	1, 1004
실수형	실수 값을 가진다.	double	1.24, 3.14
문자열	문자 정보를 가진다.	string	"hello", 'hello'
Object	정보를 필드와 값의 형태로 가진다.	object	{field: 'value', number: 1}
Array	정보를 배열의 형태로 가진다.	array	[1, 3.14, null, {x: 1}, true]
Boolean	참과 거짓에 대한 정보를 나타낸다.	boolean	true, false

Mongo DB

- ❖ Mongo DB JSON Document – Binary JSON
  - ✓ 데이터 타입
    - Timestamp
      - ◆ 하나의 mongod 인스턴스내에서 값이 항상 유일하게 설계



      - ◆ 타임스탬프 값은 그림과 같이 표시
      - ◆ 유닉스 시간은 UTC(1970년 1월 1일 00:00:00 협정 시) 부터 시작해서 지금까지 몇 초가 지났는지를 나타내는 정수
      - ◆ 하나의 mongod 인스턴스에서 같은 유닉스 시간에 타임스탬프가 여러 개 생성되면 생성된 순서에 따라 숫자를 매겨서 두 번째 인자로 가지게 됨

        {ts: Timestamp(1573822330, 1)}

        {ts: Timestamp(1573822330, 2)}

Mongo DB

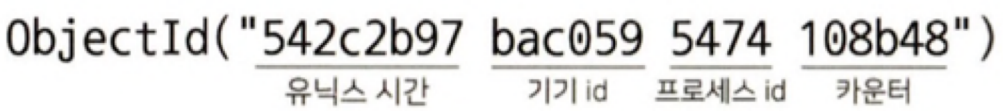
- Mongo DB JSON Document – Binary JSON
  - ✓ 데이터 타입
    - Date

```
ISODate("2017-10-24T05:02:46.395Z")
        날짜          시각
```

  - ◆ 날짜 타입은 64비트 정수형 숫자를 저장해서 기록하는데 저장된 숫자는 유닉스 시간을 의미
  - ◆ Mongo DB 셸에서는 ISODate 라는 객체로 표시
  - ◆ ISODate 객체는 JavaScript에서 Date 객체와 동일하게 사용

Mongo DB

- Mongo DB JSON Document – Binary JSON
  - ✓ 데이터 타입
    - ObjectId
      - RDBMS에서는 auto_Increment 또는 Sequence를 이용해서 일련번호를 제공
      - Mongo DB는 Collection 안에 도큐먼트 생성할 때마다 Primary key 로 _id 필드를 자동적으로 생성하는데 _id 필드는 서로 겹치지 않는 ObjectId 타입으로 값을 할당하는데 이 때문에 Collection 내에서 _id 값을 알게 되면 하나의 도큐먼트를 특정 지을 수 있음
      - 12byte 로 구성



Mongo DB

- Mongo DB JSON Document – Binary JSON
  - ✓ 데이터 타입
    - Array
      - 배열의 경우 여러 개의 요소를 내부에 담을 수 있는데 이 요소의 값으로 여러 가지 종류의 데이터를 가질 수 있으며 오브젝트도 가질 수 있어서 배열 안에 여러 개의 도큐먼트를 저장하는 것이 가능
      - 배열은 [ ]안에 표기

        [“apple”, “pear”, “banana”]

        [{“name”:”park”, ”age”:40}, {“name”:”kim”, ”age”:30}]
    - 내장 도큐먼트
      - 도큐먼트는 부모 도큐먼트의 값으로 내장된 도큐먼트 전체를 포함할 수 있다

        {"x" : {"foo" : "bar"}}
    - 이진 데이터
      - 이진 데이터는 임의의 바이트 문자열이며 셸에서는 조작이 불가능
      - 데이터베이스에 UTF-8이 아닌 문자열을 저장하는 유일한 방법
    - 코드
      - 쿼리와 도큐먼트는 임의의 자바스크립트 코드를 포함할 수 있음

        {"x" : function() { /* ... */ }}

Mongo DB

- Mongo DB JSON Document – Binary JSON
  - ✓ Document
    - JSON 형태로 데이터가 저장됨 – 키(Key)와 값(Value)을 쌍으로 저장하는 구조

      {"greeting" : "Hello, World"}

      {"greeting" : "Hello, World", "foo" : 3}
    - 문서의 키/값 쌍은 정렬되는데 아래 두 문서는 서로 다른 문서

      {"greeting" : "Hello, World", "foo" : 3}

      {"foo" : 3, "greeting" : "Hello, World"}
    - 문서의 Key는 문자열로 하고 Value는 어느 값이든 사용 가능
    - Key는 중복될 수 없음
    - 데이터 타입 과 대소문자를 구분함

Mongo DB

- Mongo DB JSON Document – Binary JSON
  - ✓ Document
    - Key는 예약어를 제외하고 사용 가능
      - Key는 ₩0(null 문자)을 포함하지 않음
      - .과 $문자는 사용할 수 없음
      - _로 시작하는 문자는 사용하지 않음 (예약어 일 가능성 높음)
      - 대소문자 및 데이터 형을 정확히 구분하며 대소문자가 다르거나 데이터 형이 다르면 서로 다른 문서로 인식
      - Key는 중복 될 수 없음
      - 예약어
        - https://www.mongodb.com/docs/manual/reference/command/

Mongo DB

- Mongo DB JSON Document
  - ✓ 도큐먼트: 하나의 데이터를 의미
  - ✓ 스키마(테이블 구조)가 없기 때문에 어떠한 형태의 데이터라도 하나의 Collection에 저장 가능하기 때문에 아래 문서들은 하나의 Collection에 같이 저장될 수 있음

    { " greeting " : " Hello, World " }

    { " foo " : 5}
  - ✓ 제한 사항
    - 하나의 도큐먼트는 { 로 시작해서 }로 종료
    - 도큐먼트의 모든 원소는 반드시 키와 값의 쌍으로 구성해야 함
    - 중첩된 도큐먼트의 깊이는 100 레벨까지 지원
    - 도큐먼트의 전체 크기는 16MB 까지 지원

Collection

- Database
  - ✓ database는 서비스나 데이터의 그룹을 만들기 위해서 사용하는 물리적인 개념
  - ✓ 데이터베이스는 동시 처리 성능과 연관됨
  - ✓ Collection이나 인덱스를 추가 또는 변경하는 경우에 데이터베이스 수준의 잠금이 적용됨
  - ✓ Map Reduce 작업도 데이터베이스 수준의 잠금을 필요로 함
  - ✓ 데이터베이스 목록 확인

    show dbs
  - ✓ 데이터베이스 생성 및 사용 설정

    use 데이터베이스이름
    - 데이터베이스가 없으면 생성하고 있으면 데이터베이스를 사용
    - 데이터베이스만 만든 상태에서는 데이터베이스 목록에 확인되지 않음
    - adam 이라는 데이터베이스 생성 및 사용

      use adam

      'switched to db adam'

Collection

- Database
  - ✓ 직접 접근할 수는 있지만 특별한 의미를 갖는 예약된 데이터베이스 이름
    - ✓ admin
      - 인증 과 권한 부여 역할을 수행
    - ✓ local
      - local 데이터베이스는 단일 서버에 대한 데이터를 저장
      - replica set에서 local은 replication 프로세스에 사용된 데이터를 저장
      - local 데이터베이스 자체는 복제되지 않음
    - ✓ config
      - sharding된 몽고DB 클러스터는 config 데이터베이스를 사용해 각 shard의 정보를 저장

Collection

- Database
  - ✓ 현재 데이터베이스 확인

    db
  - ✓ 사용 중인 데이터베이스 삭제: 이 작업 중에는 Lock 이 걸림

    db.dropDatabase()
  - ✓ 데이터베이스를 생성하고 데이터를 추가한 후 확인

    use adam

    show dbs

    admin 41 kB

    config 73.7 kB

    local 106 kB

Collection

- Database
  - ✓ 데이터베이스를 생성하고 데이터를 추가한 후 확인

```
> db.mycollection.insertOne({name:1})
{ acknowledged: true, insertedId: ObjectId("6258d86252380857ce5a33c5") }

> show dbs
adam     8.19 kB
admin      41 kB
config   73.7 kB
local     106 kB
```

Collection

- Database
  - ✓ 상태 조회
    - db.getCollectionInfos(): 현재 데이터베이스의 Collection들의 정보를 리스트로 반환하는데 이름과 타입, UUID 정보를 리턴
    - db.serverStatus(): 호스트, 프로세스 id, Lock 옵션, 스토리지 엔진 이름, 스토리지 엔진 통계 와 같은 정보를 제공
    - db.stats(): 데이터베이스 내의 Collection, 뷰, 오브젝트의 개수 와 크기에 대한 통계를 제공

Collection

- Collection
  - ✓ RDBMS에서 테이블이라고 부르는 개체와 유사한 개념을 Mongo DB에서는 Collection 이라고 함
  - ✓ 도큐먼트의 모음
  - ✓ 저장되는 데이터가 정규화 된 데이터가 아니라 응용 프로그램으로부터 직렬화된 비 정규화된 데이터를 저장하기 때문에 테이블이라는 표현을 사용하지 않지만 실무에서는 2개의 용어를 혼용
  - ✓ 동적 스키마를 가지지만 실제로는 같은 종류의 데이터를 하나의 컬렉션에 저장
  - ✓ Mongo DB에서는 Join을 지원하지 않기 때문에 하나의 Collection에 최대한 많은 양의 데이터를 저장하는 것을 권장하지만 성능 측면에서 보면 하나의 Collection에 너무 많은 양의 데이터를 저장하면 Collection의 크기가 너무 커지게 되서 디스크 읽기 오퍼레이션이 많이 필요하게 되고 메모리의 캐시 효율이 떨어지게 되므로 여러 개의 Collection을 만들어서 데이터를 나누어서 저장하기를 권장

Collection

- Collection
  - ✓ 명명 규칙
    - ₩0(null 문자)은 컬렉션명의 끝을 나타내는 문자이므로 컬렉션명에 사용할수 없음
    - system.으로 시작하는 컬렉션명은 시스템 컬렉션에서 사용하는 예약어이므로 사용할 수 없는데 예를 들어 system.users 컬렉션에는 데이터베이스 사용자 정보가 system.namespaces 컬렉션에는 데이터베이스내 모든 컬렉션의 정보가 들어 있음
    - 사용자가 만든 컬렉션은 이름에 예약어인 $를 포함할 수 없는데 시스템에서 생성한 몇몇 컬렉션에서 $ 문자를 사용
  - ✓ 서브 컬렉션
    - collection의 네임스페이스namespace에 .(마침표) 문자를 사용해 컬렉션을 체계화 할 수 있음
    - 예를 들어 블로그 기능이 있는 애플리케이션은 blog.posts와 blog.authors라는 컬렉션을 가질 수 있는데 이는 단지 체계화를 위함이며 blog 컬렉션이나 자식 컬렉션과는 아무런 관계가 없는데 blog 컬렉션은 없어도 가능
    - 서브 컬렉션은 특별한 속성은 없지만 여러 몽고DB 툴에서 지원하므로 유용
    - 큰 파일을 저장하는 프로토콜인 GridFS는 콘텐츠 데이터와 별도로 메타데이터를 저장하는 데 서브컬렉션을 사용
    - 서브컬렉션은 몽고DB의 데이터를 체계화하는 방법 중 하나
  - ✓ SQL 과 Mongo DB 의 BSON 쿼리를 비교한 맵

    https://www.mongodb.com/docs/manual/reference/sql-comparison/

Collection

- Collection
  - ✓ 관련 명령
    - Collection을 생성

      db.createCollection(“Collection이름”)
    - 데이터베이스의 Collection 조회

      db.getCollectionNames()

      show collections
    - Collection 제거 – 파라미터로 WriteConcern을 전달 할 수 있음

      db.Collection이름.drop()
    - Collection 이름 변경

      db.collection이름.renameCollection(변경할 이름)

Collection

- Collection
  - ✓ Capped Collection
    - 일반 Collection과 다르게 정해진 크기를 초과하게 되면 자동으로 가장 오래된 데이터를 삭제하는 collection
    - 생성
      - db.createCollection(<Collection 이름>, { capped: true, size: <제한할 크기>})
        - size 매개변수에 오는 부분은 바이트 단위의 숫자를 적어주게 되면 크기가 다 찼을 경우 자동으로 오래된 도큐먼트부터 삭제
        - 제한하는 크기 값이 4096보다 작거나 같으면 Collection의 크기는 4096 바이트가 되는데 기본적으로 차지하는 크기가 있기 때문이며 4096보다 더 큰 값을 넣으면 Mongo DB는 받은 크기를 늘려 256의 정수 배수로 생성
    - Capped Collection은 로그 데이터나 일정 시간 동안만 보관하는 통계 데이터를 보관하고자 하는 경우 유용하게 사용

Collection

- ❖ Collection
  - ✓ View
    - 데이터베이스 안에 쓸 수는 없고 읽을 수만 있는 데이터베이스 개체
    - 설정한 내용에 의해 뷰를 불러올 때 마다 실제로 데이터를 저장한 Collection 으로부터 데이터를 모아서 데이터를 출력하는 개체
    - 회원에 대한 Collection과 회원의 포인트에 대한 Collection이 각각 따로 있는 경우 애플리케이션의 많은 부분이 회원 정보와 포인트에 대한 정보를 동시에 접근하게 되는 경우 코드를 간결하게 유지하고 가독성을 높이기 위해 회원 정보와 포인트를 한번에 불러올 수 있는 뷰를 만들어서 활용할 수 있음
    - 뷰를 생성한다 하더라도 물리적인 데이터베이스 내부에는 회원 정보와 포인트 정보가 합쳐지지 않은 채로 존재하며 뷰에 저장된 회원 Collection 과 포인트 Collection을 합쳐서 출력하라 는 논리적인 명령이 실행되면서 기존에 존재하던 데이터를 가공하여 하나의 Collection인 것처럼 결과를 출력
    - 뷰는 실제로 데이터를 저장해서 불러오지 않기 때문에 사용할 수 있는 명령어에 제약이 있음

Collection

- Collection 상태 조회
  - ✓ show collections: 현재 DB의 컬렉션 리스트 확인
  - ✓ db.[컬렉션명].stats() : 컬렉션의 크기, 도큐먼트 개수, 스토리지 엔진의 통계 제공
  - ✓ db.[컬렉션명].isCapped() : Capped 컬렉션일 경우 true 반환
  - ✓ db.[컬렉션명].latencyStats() : 컬렉션의 지연 시간 통계 반환
  - ✓ db.[컬렉션명].storageSize() : 컬렉션 스토리지 크기 반환
  - ✓ db.[컬렉션명].totalIndexSize() : 컬렉션의 인덱스 크기 반환
  - ✓ db.[컬렉션명].totalSize() : 컬렉션 스토리지 + 인덱스 크기 반환

Collection

❖ Capped Collection 의 동작

```
> use adam
> db.createCollection('cappedCollection', {capped:true, size:10000})
{ ok: 1 }

> db.cappedCollection.insertOne({x:1})
{ acknowledged: true,
  insertedId: ObjectId("625cdb5b88993df9910866b2") }

> db.cappedCollection.find()
{ _id: ObjectId("625cdb5b88993df9910866b2"), x: 1 }
```

Collection

❖ Capped Collection 의 동작

```
> db.cappedCollection.stats()
{
  ns: 'adam.cappedCollection',
  size: 29,
  count: 1,
  avgObjSize: 29,
  storageSize: 20480,
  freeStorageSize: 0,
  capped: true,
  max: 0,
  maxSize: 10240,
```

Collection

- Capped Collection 의 동작

  > for(i=0; i<1000; i++){ db.cappedCollection.insertOne({x:i})}

  { acknowledged: true,

  insertedId: ObjectId("625d022b88993df991086a9a") }

Collection

❖ Capped Collection 의 동작

> db.cappedCollection.find()

```
{ _id: ObjectId("625d022688993df99108693a"), x: 647 }
{ _id: ObjectId("625d022688993df99108693b"), x: 648 }
{ _id: ObjectId("625d022688993df99108693c"), x: 649 }
{ _id: ObjectId("625d022688993df99108693d"), x: 650 }
{ _id: ObjectId("625d022688993df99108693e"), x: 651 }
{ _id: ObjectId("625d022688993df99108693f"), x: 652 }
{ _id: ObjectId("625d022688993df991086940"), x: 653 }
{ _id: ObjectId("625d022688993df991086941"), x: 654 }
{ _id: ObjectId("625d022688993df991086942"), x: 655 }
{ _id: ObjectId("625d022688993df991086943"), x: 656 }
{ _id: ObjectId("625d022688993df991086944"), x: 657 }
{ _id: ObjectId("625d022688993df991086945"), x: 658 }
{ _id: ObjectId("625d022688993df991086946"), x: 659 }
{ _id: ObjectId("625d022688993df991086947"), x: 660 }
{ _id: ObjectId("625d022688993df991086948"), x: 661 }
{ _id: ObjectId("625d022688993df991086949"), x: 662 }
{ _id: ObjectId("625d022688993df99108694a"), x: 663 }
```