Redis OM

❖Introduction to Redis OM

- What is Redis OM?
  - Redis OM (pronounced "ohm") stands for **Object Mapping (and more) for Redis.** It's a family of high-level client libraries that provide powerful abstractions for working with Redis Stack, making it as easy as possible to use Redis as a document database with modern application frameworks.
  - Redis OM transforms Redis from a simple key-value store into a sophisticated document database by leveraging Redis Stack modules, particularly RedisJSON and RediSearch, to provide:
    - **Object Mapping:** Seamlessly map domain objects to Redis data structures
    - **Fluent Querying:** Query your data using language-native, type-safe APIs
    - **Automatic Indexing:** Efficient secondary indexes created automatically
    - **Full-Text Search:** Built-in search capabilities across your data
    - **Vector Similarity Search:** Modern AI/ML capabilities for semantic search

- Why Use Redis OM?
  - Traditional Redis usage requires developers to manually compose abstractions using core data structures. While powerful, this approach demands significant time and expertise. Redis OM solves this by providing:
    - **Developer Productivity:** Focus on business logic, not Redis internals
    - **Type Safety:** Compile-time guarantees and IDE support
    - **Performance:** Always-indexed queries for optimal performance
    - **Modern Features:** Vector search, geospatial queries, and full-text search
    - **Familiar APIs:** Language-idiomatic interfaces that feel natural

- Benefits and Use Cases
  - **Primary Benefits:**
    - Reduced boilerplate code for data access
    - Automatic query optimization through indexing
    - Type-safe operations with compile-time validation
    - Rich querying capabilities out of the box
    - High performance with Redis's in-memory architecture
  - **Common Use Cases:**
    - **Document Databases:** Store and query hierarchical data
    - **Search Applications:** Full-text search across content
    - **Real-time Analytics:** Fast aggregations and filtering
    - **Recommendation Systems:** Vector similarity for AI-powered recommendations
    - **Geospatial Applications:** Location-based queries and filtering
    - **Caching with Querying:** More than simple key-value caching

❖Core Concepts

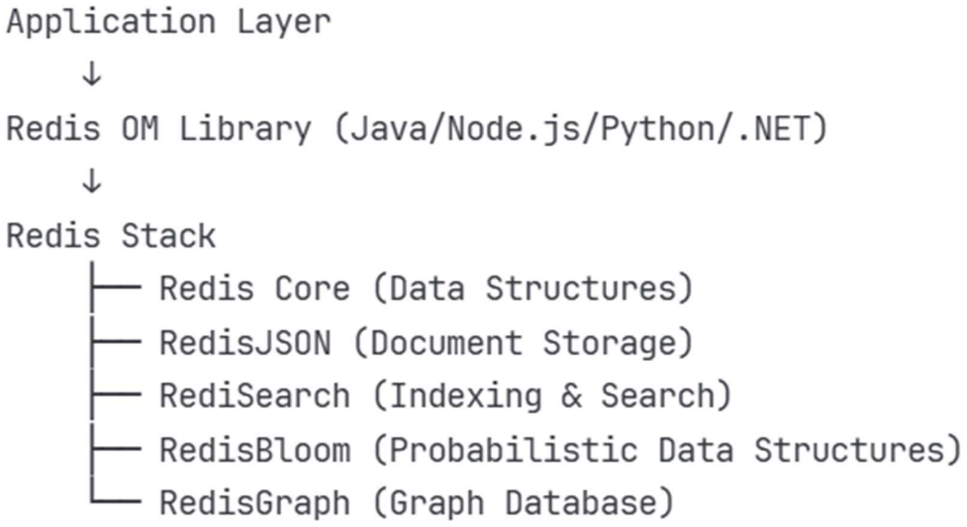
- Object Mapping and Schema Definition
  - Redis OM automatically maps your domain objects to Redis data structures. Objects can be stored as:
    - **Redis Hashes:** For simple, flat structures
    - **Redis JSON Documents:** For complex, nested data (requires RedisJSON)
  - **Schema Definition Process:**
    - Annotate your domain classes with mapping annotations
    - Define field types and indexing strategies
    - Configure relationships and nested objects
    - Generate indexes automatically at runtime

- Indexing and Search Capabilities
  - Redis OM leverages RediSearch to provide powerful indexing:
  - **Index Types:**
    - **Tag Indexes:** Exact-match searches (strings, enums)
    - **Text Indexes:** Full-text search with stemming and fuzzy matching
    - **Numeric Indexes:** Range queries and sorting
    - **Geo Indexes:** Geospatial queries within radius or bounding box
    - **Vector Indexes:** Similarity search for AI/ML applications
  - **Automatic Index Creation:**
    - Indexes are created automatically based on annotations
    - Field-level control over indexing behavior
    - Support for nested object indexing
    - Composite indexes for multi-field queries

- CRUD Operations
  - All Redis OM libraries provide consistent CRUD interfaces:
    - **Create:** Save new entities with auto-generated IDs
    - **Read:** Fetch by ID or query with complex criteria
    - **Update:** Modify existing entities with optimistic concurrency
    - **Delete:** Remove individual entities or bulk operations
  - **Advanced Operations:**
    - Bulk operations for better performance
    - Upsert capabilities
    - Partial updates
    - Transactional operations where supported

❖Architecture Overview

- How Redis OM Fits into Redis Ecosystem



```
Application Layer
    ↓
Redis OM Library (Java/Node.js/Python/.NET)
    ↓
Redis Stack
    ├── Redis Core (Data Structures)
    ├── RedisJSON (Document Storage)
    ├── RediSearch (Indexing & Search)
    ├── RedisBloom (Probabilistic Data Structures)
    └── RedisGraph (Graph Database)
```

- Client-Server Interactions
  - **Data Flow:**
    - **Entity Definition:** Domain objects defined with annotations
    - **Index Creation:** Automatic schema generation and index creation
    - **Data Persistence:** Objects serialized to JSON/Hash format
    - **Query Processing:** High-level queries translated to Redis commands
    - **Result Mapping:** Redis responses mapped back to domain objects
  - **Connection Management:**
    - Connection pooling for optimal performance
    - Automatic retry and failover capabilities
    - Support for Redis Cluster and Sentinel configurations
    - SSL/TLS encryption support

❖Supported Languages and SDKs

- ### Overview of Language-Specific Implementations
  - ➢ Redis OM currently supports four major programming ecosystems:
    - **Redis OM Spring (Java):** Deep integration with Spring Framework
    - **Redis OM Node.js:** TypeScript-first with JavaScript support
    - **Redis OM Python:** Async/sync support with FastAPI integration
    - **Redis OM .NET:** LINQ support for C# developers

- ### Key Similarities Across SDKs
  - ➢ **Common Features:**
    - Consistent annotation-based mapping
    - Repository pattern implementation
    - Fluent query APIs
    - Automatic index management
    - ULID-based ID generation
    - Vector similarity search support
  - ➢ **Shared Concepts:**
    - Entity/Document annotations
    - Field-level indexing control
    - Relationship mapping
    - Query builders
    - Connection management

- ### Key Differences
  - ➢ **Language-Specific Adaptations:**
    - **Spring:** Leverages Spring Data patterns and annotations
    - **Node.js:** TypeScript-first design with modern async/await
    - **Python:** Pydantic integration and FastAPI compatibility
    - **.NET:** LINQ query support and Entity Framework-like experience

❖Common Features Across SDKs

- Data Modelling

➢ Entity Definition:

```
// Conceptual structure - syntax varies by language
@Document
class Product {
    @Id
    String id;

    @Indexed
    String name;

    @Searchable
    String description;

    @Indexed
    Double price;

    @Indexed
    Set<String> tags;
}
```

➢ Supported Field Types:

- Primitive types (string, number, boolean)
- Collections (arrays, sets, lists)
- Nested objects and documents
- Geospatial coordinates
- Date/time types
- Binary data (in some implementations)

- Querying
  - Query Methods:
    - **Equality:** Exact matches on indexed fields
    - **Range:** Numeric and date range queries
    - **Full-Text:** Search across text fields with ranking
    - **Geospatial:** Distance-based and bounding box queries
    - **Tag Matching:** Set membership and intersection queries
    - **Vector Similarity:** Semantic search with embeddings
  - Query Building:
    - Fluent API for complex query construction
    - Method chaining for multiple conditions
    - Sorting and pagination support
    - Aggregation capabilities

- Repository Pattern
  - Standard Operations:
    - save(entity): Persist entity to Redis
    - findById(id): Retrieve by primary key
    - findAll(): Get all entities of type
    - delete(entity): Remove entity
    - count(): Get total entity count
  - Custom Queries:
    - Method name-based query derivation
    - Annotation-based query definitions
    - Native Redis query support
    - Stream-based processing

- Indexing
  - Automatic Index Management:
    - Indexes created on application startup
    - Field-level index configuration
    - Composite index support
    - Index rebuilding capabilities

- Index Types Configuration:
  - Text indexes with language-specific stemming
  - Tag indexes for exact matching
  - Numeric indexes with range support
  - Geo indexes for location queries
  - Vector indexes for similarity search