

# 组织架构

---

## 功能描述

- 提供组织机构的查询和修改
- 组织架构变化支持订阅和发布

## 性能和扩展性要求

- 读写都复杂
- 能够扩展
- 读的频率更多

## 领域模型（待完成）

- Company
  - id
  - name
  - departments
  - members
- Department
- SubCompany
- Employee

以 部门 作为聚合根

## API（待完成）

## 参考CQRS / EventSourcing 架构模式

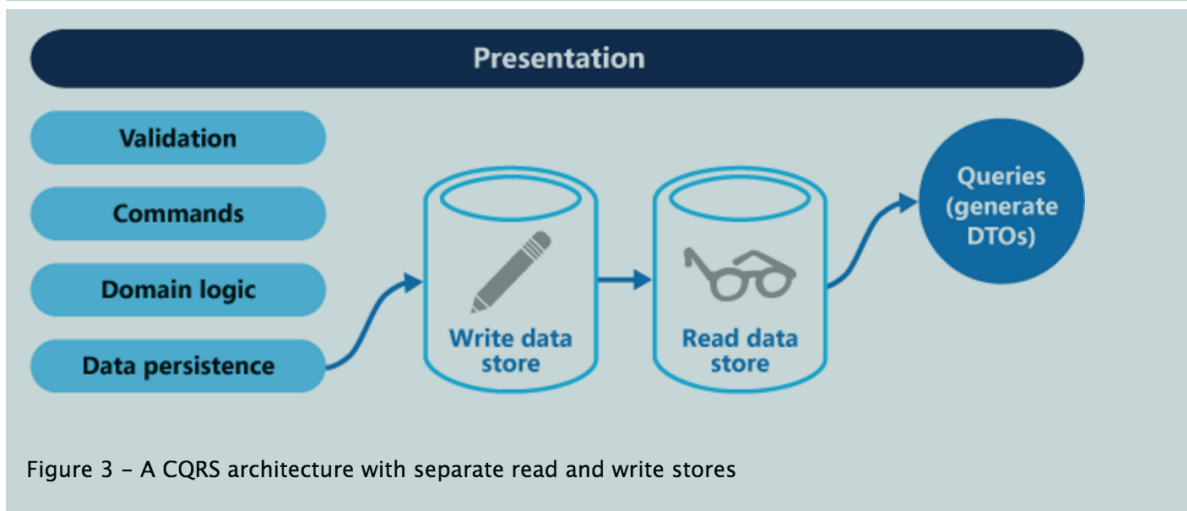
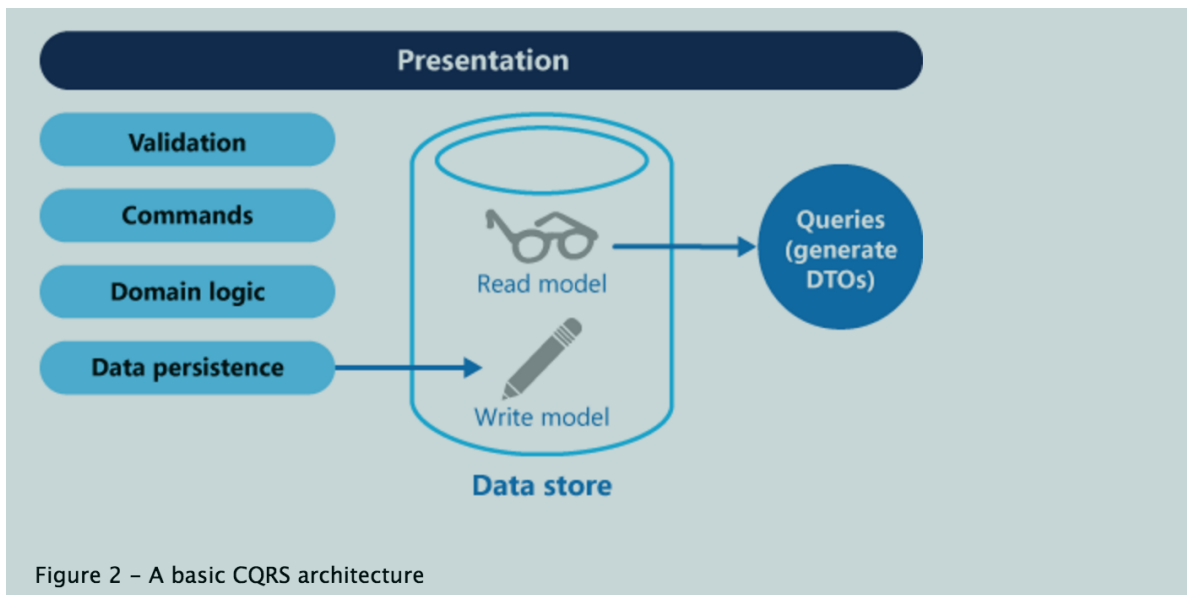
CQRS 的介绍网上有许多，这里采用微软MSDN上的文档：链接如下：

[Command and Query Responsibility Segregation \(CQRS\) Pattern](#)

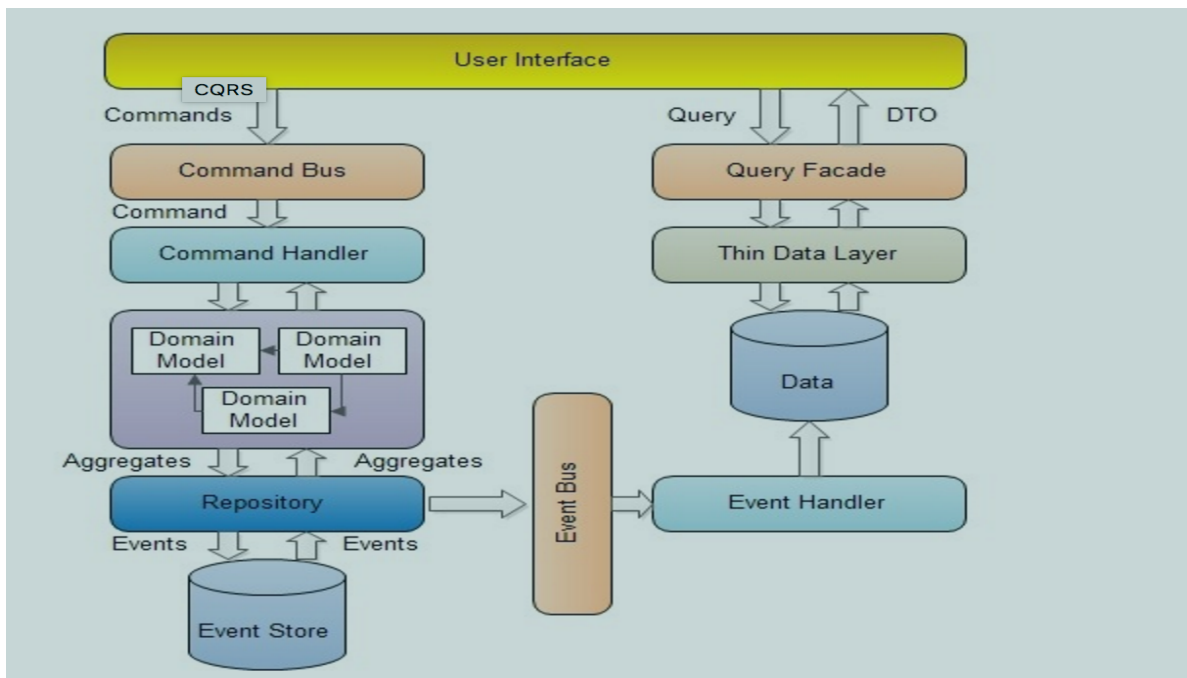
CQRS有好几种变种，例如这份文档就给出两种：

**Figure 2** 这种读写模式共享数据库。在代码层面，将读和写的逻辑剥离，好处是能够降低业务处理的复杂度。

**Figure 3** 这种模式是分离了读库和写库，这种模式，读库（WriteDB）和写库（ReadDB）分开部署，可以部署不通的数据（NoSQL + RDBMS 可以混合部署），扩展性好。读和写的差异更大。



后一种模式，完整的CQRS架构如下。在JVM平台上，大部分组件都是现成的，并且使用比较广泛。



## 并发，Actor模型和AKKA

并发处理方面，有个比较成熟的Actor模型（JVM平台上），akka。

这个akka对于我们来说有几个优势：

- 很好的并发。akka 官网宣布的性能是

50 million msg/sec on a single machine. Small memory footprint; ~2.5 million actors per GB of heap.

- Actor 也是比较熟悉的模型（不过以前是Erlang的Actor模型）
- akka有个组件，叫做persistence,这种方式实现EventSourcing和CQRS比较自然。

## 性能模型

### 写入性能分析

总得代价为校验加上事件发布的代价。

- 校验
- 事件（日志-leveldb\cassandra(LSM)）
  - DepartmentDisabled
  - DepartmentMoved
  - EmployeeJoined
  - ..

### 读取性能评估

读取的方式有多种

- 读取部门信息
- 读取员工信息
- 查询部门员工集合
- 查询满足特定调节的岗位员工

可以看出大部分，查询可以用一个SQL，高效查询出。但是有部分查询，则不太高效，例如：

- 查询一个部门的所有子(父)部门
- 查询一个员工直属的子公司

这些查询，使用SQL，则不太高效，所以最好，做个冗余的检索信息结构。在查询这类信息时，用这种结构做检索，再辅助SQL。例如

查询一个部门的所有子孙部门的人员列表

这个查询，可以分下面两个阶段的 **查询计划**

1. 从检索结构中查出相关的拓扑关系（例如，子部门ID）
2. SQL查询（例如，`select * from employee where department_id in (子部门id列表)`）

## 部门拓扑关系高效查询的可能方式有

### 内存建模

在内存中自己维护这个拓扑关系。这里需要注意内存的使用，**防止使用过多的内存**。

可以有下面的一些措施减少内存占用：

- **succinct** 信息结构
- 定期失效，移除内存

### **neo4j** 等图数据存储拓扑关系

使用图数据库来维护这个关系。

## 数据切分

以公司做水平划分。需要避免跨公司查询。

## 读表结构（待完成）

- company
  - id
  - name
  - parent(?)
- department
  - id
  - name
  - parent (?)
- employee
- user\_company
- user\_department

## 一致性分析（待完成）

## 业务迁移（待完成）