MongoDB数据访问完整指南

🎯 数据存储总览

📊 需要存储在MongoDB的数据

数据类型	集合名称	存储原因	访问模式
🏢 **建筑物数据**	`buildings`	地理空间查询、缓存优化	高频读取
🗺️ **DEM瓦片元数据**	`dem_tiles`	缓存管理、统计分析	中频读写
🚶 **用户轨迹**	`user_tracks`	出行分析、历史记录	读写平衡
👤 **用户数据**	`users`	个性化设置、统计	低频读写
🌤️ **天气缓存**	`weather_cache`	性能优化、历史分析	高频读取
🛣️ **路线推荐**	`route_recommendations`	智能推荐、缓存	中频读取

📁 不存储在MongoDB的数据

数据类型	存储位置	原因
🖼️ **DEM瓦片文件**	文件系统 + CDN	二进制数据、适合CDN
🎨 **前端资源**	CDN	静态内容、全球分发
⚡ **实时计算结果**	内存缓存 (Redis)	临时数据、快速访问

🌐 API接口使用示例

1. 建筑物数据访问

```
# 获取瓦片建筑物数据
GET /api/buildings/15/26976/13487.json

# 响应示例
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[116.397, 39.908], ...]]
      },
      "properties": {
        "id": "way_227192080",
        "buildingType": "residential",
        "height": 25.5,
        "levels": 8
```

```
      }
    }
  ],
  "bbox": [116.394, 39.905, 116.400, 39.909],
  "tileInfo": { "z": 15, "x": 26976, "y": 13487 },
  "cached": true,
  "fromDatabase": true
}

# 服务信息
GET /api/buildings/info

# 批量预加载
POST /api/buildings/preload
{
  "tiles": [
    {"z": 15, "x": 26976, "y": 13487},
    {"z": 15, "x": 26977, "y": 13487}
  ]
}
```

2. 用户轨迹数据访问

```
# 创建用户轨迹
POST /api/tracks
{
  "user_id": "user_123",
  "name": "晨跑路线",
  "gps_points": [
    {
      "lng": 116.397,
      "lat": 39.908,
      "timestamp": "2025-01-15T06:30:00Z",
      "accuracy": 5.0
    },
    // ... 更多GPS点
  ],
  "metadata": {
    "activity_type": "running"
  }
}

# 获取用户轨迹列表
GET /api/tracks/user/user_123?activity=running&limit=20

# 响应示例
{
  "tracks": [
    {
      "id": "ObjectId",
      "name": "晨跑路线",
      "activity_type": "running",
      "distance": 2500,
      "duration": 18,
      "comfort_score": 7.5,
```

```
      "created_at": "2025-01-15T06:30:00Z"
    }
  ],
  "pagination": {
    "has_more": true,
    "next_cursor": "ObjectId"
  }
}

# 获取公开轨迹
GET /api/tracks/public?activity=walking&min_comfort=7

# 分析轨迹阴影
POST /api/tracks/{trackId}/analyze

# 用户统计
GET /api/tracks/user/{userId}/stats
```

3. 天气数据访问

```
# 获取当前天气
GET /api/weather/current?lat=39.908&lng=116.397

# 响应示例
{
  "location": {
    "latitude": 39.908,
    "longitude": 116.397
  },
  "timestamp": "2025-01-15T12:00:00Z",
  "weather": {
    "temperature": 23.5,
    "humidity": 65,
    "cloud_cover": 0.3,
    "uv_index": 6,
    "wind_speed": 2.2,
    "wind_direction": 180,
    "visibility": 10000,
    "precipitation": 0,
    "pressure": 1013
  }
}

# 批量获取天气
POST /api/weather/batch
{
  "locations": [
    {"lat": 39.908, "lng": 116.397},
    {"lat": 39.910, "lng": 116.400}
  ],
  "timestamp": "2025-01-15T12:00:00Z"
}

# 预加载区域天气
POST /api/weather/preload
```

```
{
  "bounds": {
    "west": 116.3,
    "south": 39.9,
    "east": 116.4,
    "north": 40.0
  }
}

# 缓存统计
GET /api/weather/cache/stats
```

🔍 数据查询模式

1. 地理空间查询

```
// MongoDB查询示例

// 1. 瓦片查询（最常用）
db.buildings.find({
  "tile.z": 15,
  "tile.x": 26976,
  "tile.y": 13487
})

// 2. 边界框查询
db.buildings.find({
  "geometry": {
    $geoIntersects: {
      $geometry: {
        type: "Polygon",
        coordinates: [[
          [116.3, 39.9], [116.4, 39.9],
          [116.4, 40.0], [116.3, 40.0],
          [116.3, 39.9]
        ]]
      }
    }
  }
})

// 3. 附近查询
db.user_tracks.find({
  "route": {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [116.397, 39.908]
      },
      $maxDistance: 1000 // 1公里内
    }
  }
})
```

2. 聚合查询

```
// 用户活动统计
db.user_tracks.aggregate([
  { $match: { user_id: "user_123" } },
  { $group: {
    _id: "$metadata.activity_type",
    count: { $sum: 1 },
    total_distance: { $sum: "$metadata.total_distance" },
    avg_comfort: { $avg: "$analysis.comfort_score" }
  }},
  { $sort: { count: -1 } }
])

// 建筑物类型分布
db.buildings.aggregate([
  { $group: {
    _id: "$properties.buildingType",
    count: { $sum: 1 },
    avg_height: { $avg: "$properties.height" }
  }},
  { $sort: { count: -1 } },
  { $limit: 10 }
])

// 天气数据趋势
db.weather_cache.aggregate([
  { $match: {
    grid_cell: "39.9_116.4",
    timestamp: { $gte: new Date("2025-01-01") }
  }},
  { $group: {
    _id: {
      year: { $year: "$timestamp" },
      month: { $month: "$timestamp" },
      day: { $dayOfMonth: "$timestamp" }
    },
    avg_temp: { $avg: "$data.temperature" },
    avg_uv: { $avg: "$data.uv_index" }
  }},
  { $sort: { "_id.year": 1, "_id.month": 1, "_id.day": 1 } }
])
```

⚡ 性能优化策略

1. 索引设计

```
// 地理空间索引（必须）
db.buildings.createIndex({ "geometry": "2dsphere" })
db.user_tracks.createIndex({ "route": "2dsphere" })
db.weather_cache.createIndex({ "location": "2dsphere" })

// 瓦片查询优化
db.buildings.createIndex({
```

```
  "tile.z": 1,
  "tile.x": 1,
  "tile.y": 1,
  "properties.height": -1
})

// 时间序列优化
db.user_tracks.createIndex({ "user_id": 1, "created_at": -1 })
db.weather_cache.createIndex({ "grid_cell": 1, "timestamp": -1 })

// TTL索引（自动清理）
db.weather_cache.createIndex(
  { "expires_at": 1 },
  { expireAfterSeconds: 0 }
)
```

2. 查询优化

```
// 使用lean()跳过Mongoose对象转换
const buildings = await Building.find(query).lean();

// 限制返回字段
const tracks = await UserTrack.find(query)
  .select('user_id metadata.distance analysis.comfort_score')
  .lean();

// 批量插入优化
await Building.insertMany(buildings, {
  ordered: false,  // 允许部分失败继续
  lean: true       // 跳过验证提升性能
});

// 分页使用cursor而非offset
const tracks = await UserTrack.find({
  _id: { $lt: new ObjectId(cursor) }
}).limit(20);
```

3. 缓存策略

```
// 多级缓存架构
const cacheHierarchy = {
  L1_Memory: {
    hot_tiles: "5 minutes",
    current_weather: "2 minutes"
  },
  L2_MongoDB: {
    buildings: "30 days",
    weather_cache: "6 hours",
    user_tracks: "permanent"
  },
  L3_External_API: {
    osm_api: "fallback only",
    weather_api: "fallback only"
  }
```

```
};
```

📊 数据监控和维护

1. 定期维护任务

```
# 清理过期天气缓存
DELETE /api/weather/cache/cleanup

# 清理过期建筑物数据
DELETE /api/buildings/cleanup?maxAge=2592000000

# 获取数据库统计
GET /api/buildings/stats
GET /api/weather/cache/stats
```

2. 性能监控

```
// MongoDB性能监控
db.runCommand({ "serverStatus": 1 })
db.buildings.getIndexes()
db.buildings.stats()

// 慢查询分析
db.setProfilingLevel(1, { slowms: 100 })
db.system.profile.find().limit(5).sort({ ts: -1 })
```

3. 容量规划

```
// 数据增长预估
const growthProjection = {
  buildings: {
    current: "20GB",
    yearly_growth: "10%",
    5_year_projection: "32GB"
  },
  user_tracks: {
    daily_new: "5MB",
    yearly_growth: "1.8GB",
    retention: "永久保存"
  },
  weather_cache: {
    active_data: "10GB",
    auto_cleanup: "6小时TTL",
    stable_size: "是"
  }
};
```

🔧 故障排除

常见问题

1. **连接超时**

```
# 检查Atlas连接
npx ts-node src/utils/testAtlasConnection.ts
```

2. **查询性能差**

```
// 检查索引使用
db.buildings.find(query).explain("executionStats")
```

3. **内存使用高**

```
// 使用流式处理
const cursor = Building.find(query).cursor();
for (let doc = await cursor.next(); doc != null; doc = await cursor.next()) {
  // 处理单个文档
}
```

🎯 最佳实践总结

1. **数据模型设计**
   - ✅ 合理使用地理空间索引
   - ✅ 设计有效的复合索引
   - ✅ 控制文档大小（< 16MB）
2. **查询优化**
   - ✅ 使用lean()查询
   - ✅ 限制返回字段
   - ✅ 使用cursor分页
3. **缓存策略**
   - ✅ 多级缓存架构
   - ✅ 合理的TTL设置
   - ✅ 智能预加载
4. **监控维护**
   - ✅ 定期性能分析
   - ✅ 自动化清理任务
   - ✅ 容量规划监控

这个完整的MongoDB数据存储方案为ShadowMap项目提供了强大的数据管理能力，支持高性能的地理空间查询、智能缓存管理，以及完整的用户轨迹分析功能！