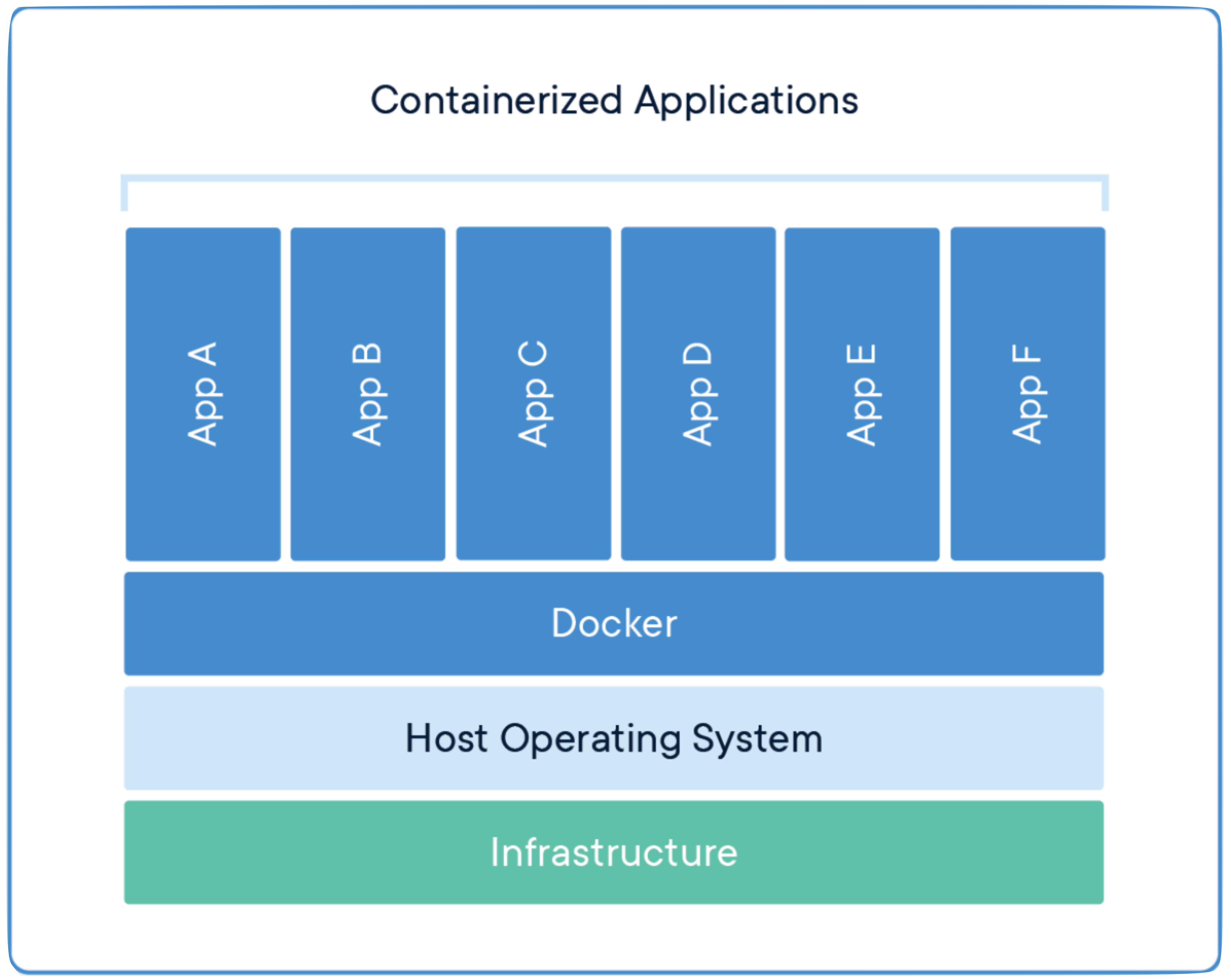
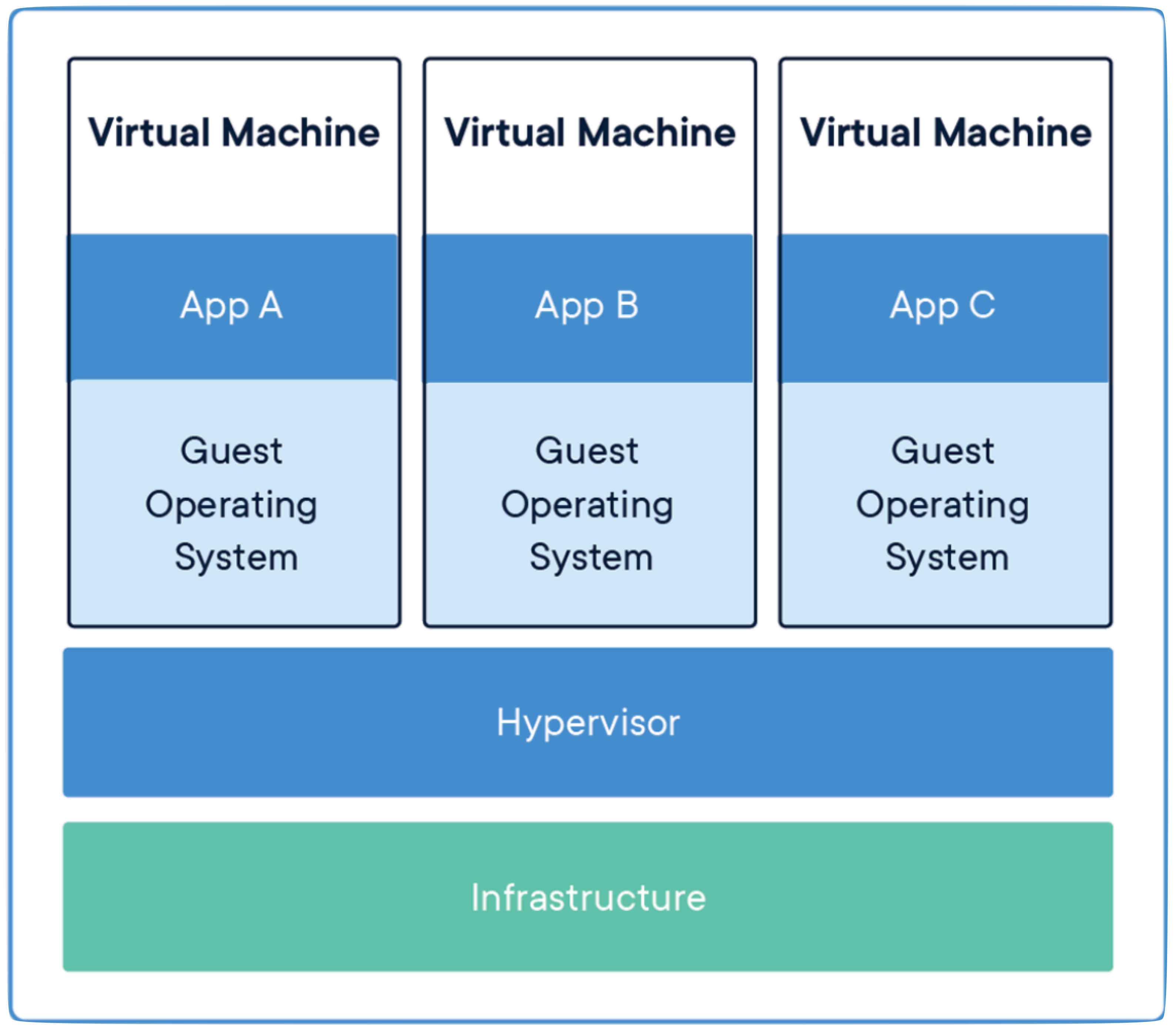
NoSQL 实践

通过 Docker 辅助开发

什么是容器

应用层的抽象



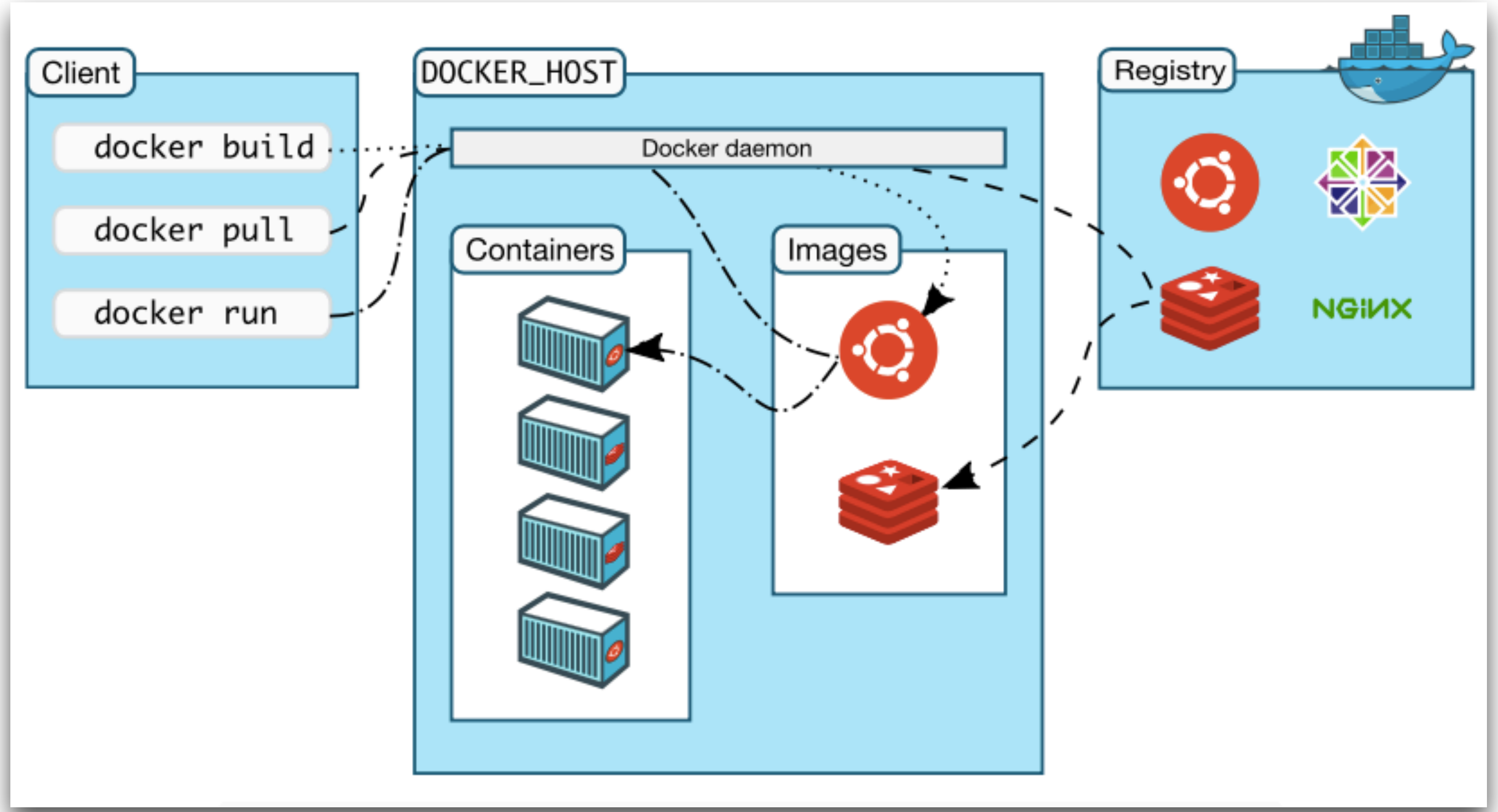


容器与虚拟机不同，不包含操作系统相关的细节和内容

所以容器更加轻量，部署更快

使用Docker创建容器，在容器里
创建一些基础设施

认识 Docker



不同人眼中的 Docker

开发眼中的 Docker

- 简化了重复搭建开发环境的工作

运维眼中的 Docker

- 交付系统更为流畅
- 伸缩性更好

Docker 常用命令

镜像相关

- docker pull <image>   Hub上下载镜像
- docker search <image>

容器相关

- docker run
- docker start/stop <容器名>
- docker ps <容器名>
- docker logs <容器名>

docker run 的常用选项

docker run [OPTIONS] IMAGE [COMMAND] [ARG...] 镜像运行起来，变成容器

选项说明

- -d，后台运行容器
- -e，设置环境变量
- --expose / -p 宿主端口:容器端口
- --name，指定容器名称
- --link，链接不同容器
- -v 宿主目录:容器目录，挂载磁盘卷

国内 Docker 镜像配置

官方 Docker Hub

- https://hub.docker.com

官方镜像

- 镜像 https://www.docker-cn.com/registry-mirror
- 下载 https://www.docker-cn.com/get-docker

阿里云镜像

- https://dev.aliyun.com

通过 Docker 启动 MongoDB

官方指引

- https://hub.docker.com/_/mongo

获取镜像

- `docker pull mongo`

运行 MongoDB 镜像

-
  ```
  docker run --name mongo -p 27017:27017 -v ~/docker-data/mongo:/data/db -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin -d mongo
  ```

What is MongoDB?

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata. MongoDB is developed by MongoDB Inc., and is published under a combination of the Server Side Public License and the Apache License.

First developed by the software company 10gen (now MongoDB Inc.) in October 2007 as a component of a planned platform as a service product, the company shifted to an open source development model in 2009, with 10gen offering commercial support and other services. Since then, MongoDB has been adopted as backend software by a number of major websites and services, including MetLife, Barclays, ADP, UPS, Viacom, and the New York Times, among others. MongoDB is the most popular NoSQL database system.

> wikipedia.org/wiki/MongoDB

How to use this image

Start a `mongo` server instance

```
$ docker run --name some-mongo -d mongo:tag
```

... where `some-mongo` is the name you want to assign to your container and `tag` is the tag specifying the MongoDB version you want. See the list above for relevant tags.

Connect to MongoDB from another Docker container

The MongoDB server in the image listens on the standard MongoDB port, `27017`, so connecting via Docker networks will be the same as connecting to a remote `mongod`. The following example starts another MongoDB container instance and runs the `mongo` command line client against the original MongoDB container from the example above, allowing you to execute MongoDB statements against your database instance:

```
$ docker run -it --network some-network --rm mongo mongo --host some-mongo test
```

通过 Docker 启动 MongoDB

登录到 MongoDB 容器中

- `docker exec -it mongo bash`

通过 Shell 连接 MongoDB

- `mongo -u admin -p admin`

Show dbs

在 Spring 中访问 MongoDB

Spring 对 MongoDB 的支持

MongoDB 是一款开源的文档型数据库

- https://www.mongodb.com

Spring 对 MongoDB 的支持

- Spring Data MongoDB
  - MongoTemplate
  - Repository 支持

Spring 对底层的添加一层封装

NoSQL:
1.key-value: Redis, Memcached

2.文档型：MongoDB,CouchBase

3.列存储：HBase, Cassandra

4. 图数据库： Neo4J

5. NewSQL: TiDB, CosmosDB 都是走 Google Spanner的技术路线发展

Spring Data MongoDB 的基本用法

注解

- `@Document` 等同于SQL的 @Entity，标注哪个文档
- `@Id` 用于处理数据分片

MongoTemplate

- `save / remove`
- `Criteria / Query / Update`

极客时间

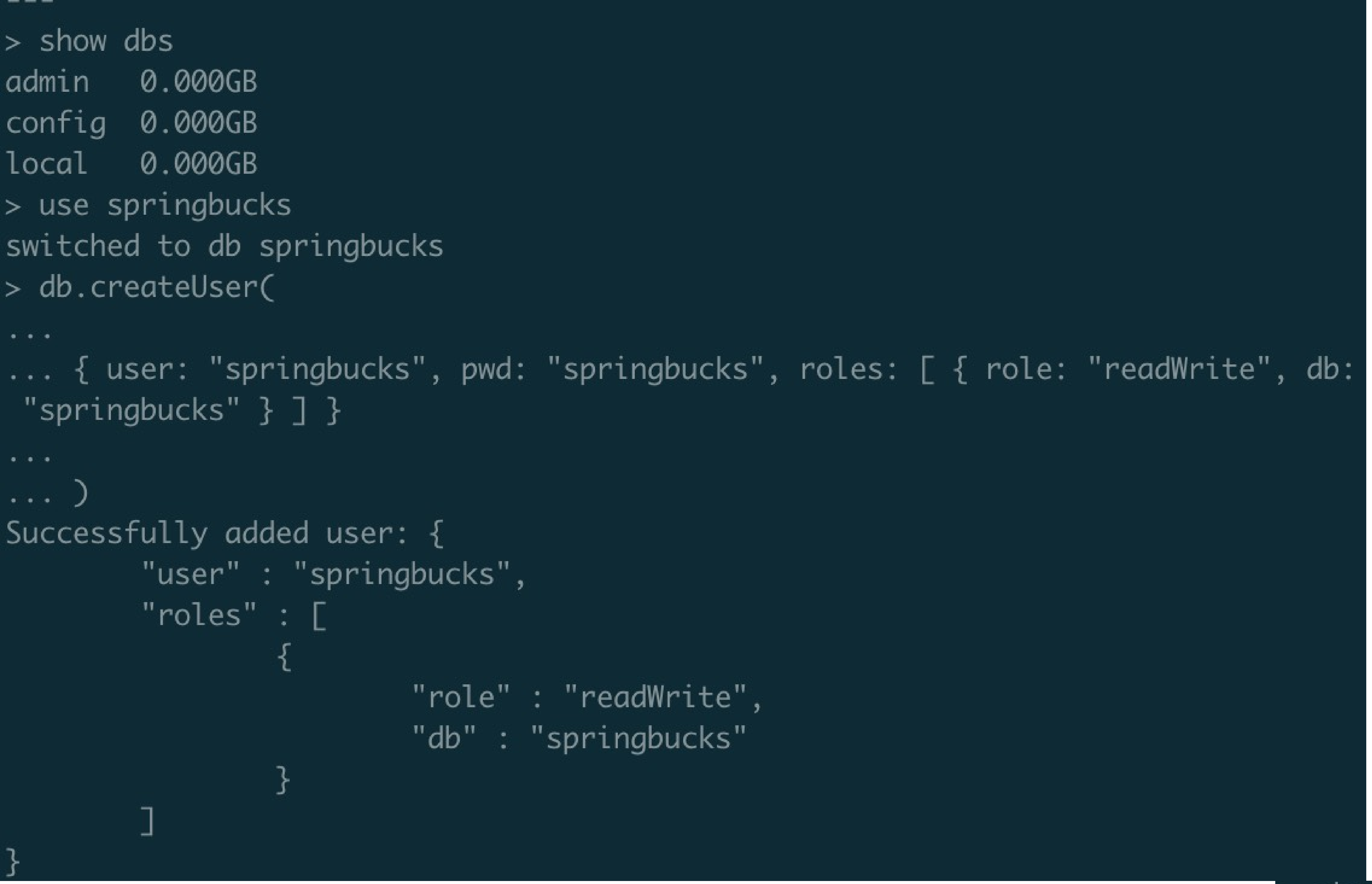
初始化 MongoDB 的库及权限

创建库

```
use springbucks;
```

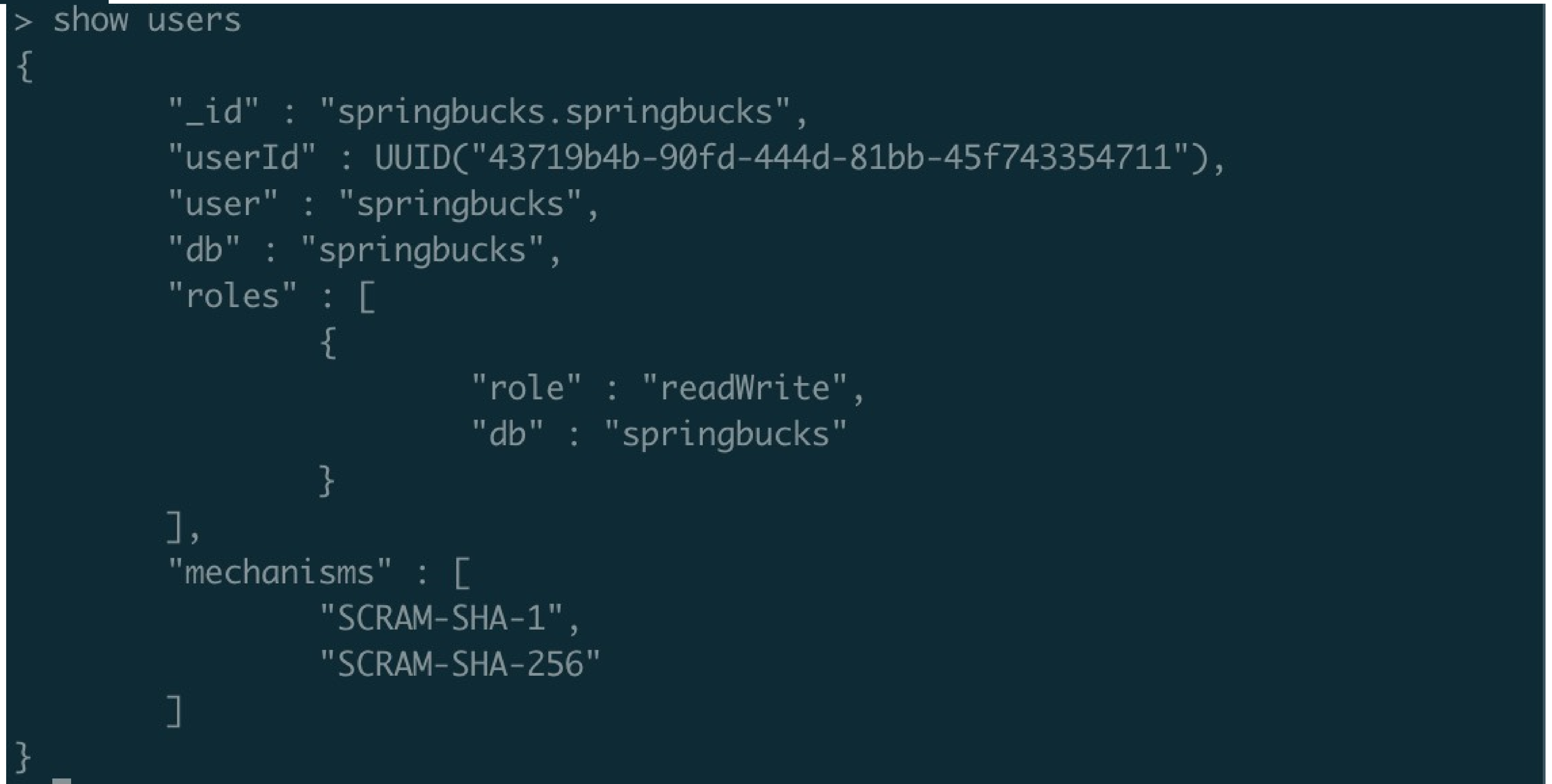
创建用户

```
db.createUser(
  {
    user: "springbucks",
    pwd: "springbucks",
    roles: [
       { role: "readWrite", db: "springbucks" }
    ]
  }
)
```



```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use springbucks
switched to db springbucks
> db.createUser(
...
... { user: "springbucks", pwd: "springbucks", roles: [ { role: "readWrite", db:
 "springbucks" } ] }
...
... )
Successfully added user: {
        "user" : "springbucks",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "springbucks"
                }
        ]
}
```

先创建一个Mongo数据库



```
> show users
{
        "_id" : "springbucks.springbucks",
        "userId" : UUID("43719b4b-90fd-444d-81bb-45f743354711"),
        "user" : "springbucks",
        "db" : "springbucks",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "springbucks"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
```

“Talk is cheap, show me the code.”

Chapter 4 / mongo-demo

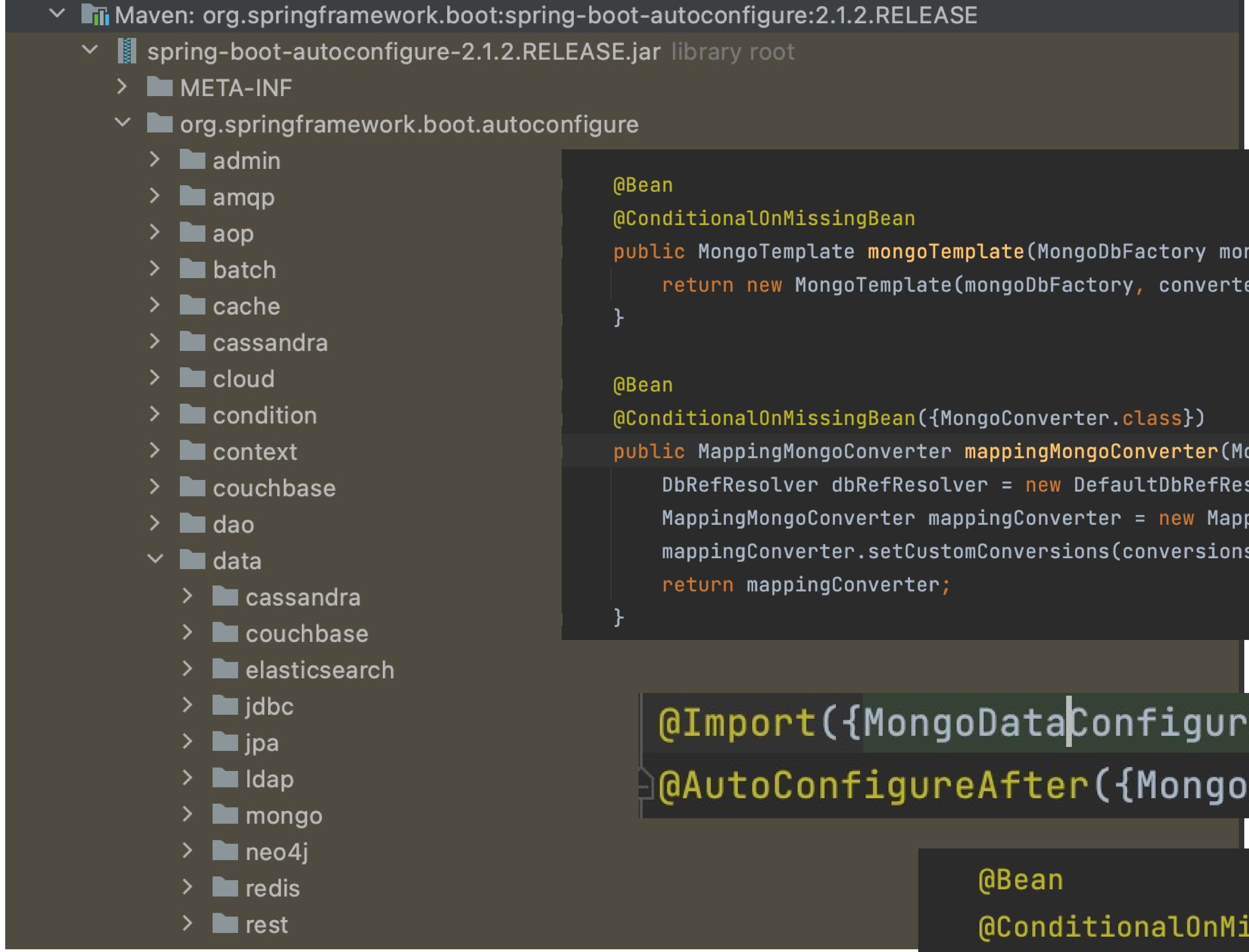
Spring Data MongoDB 的 Repository

@Document 对应数据库的一个document

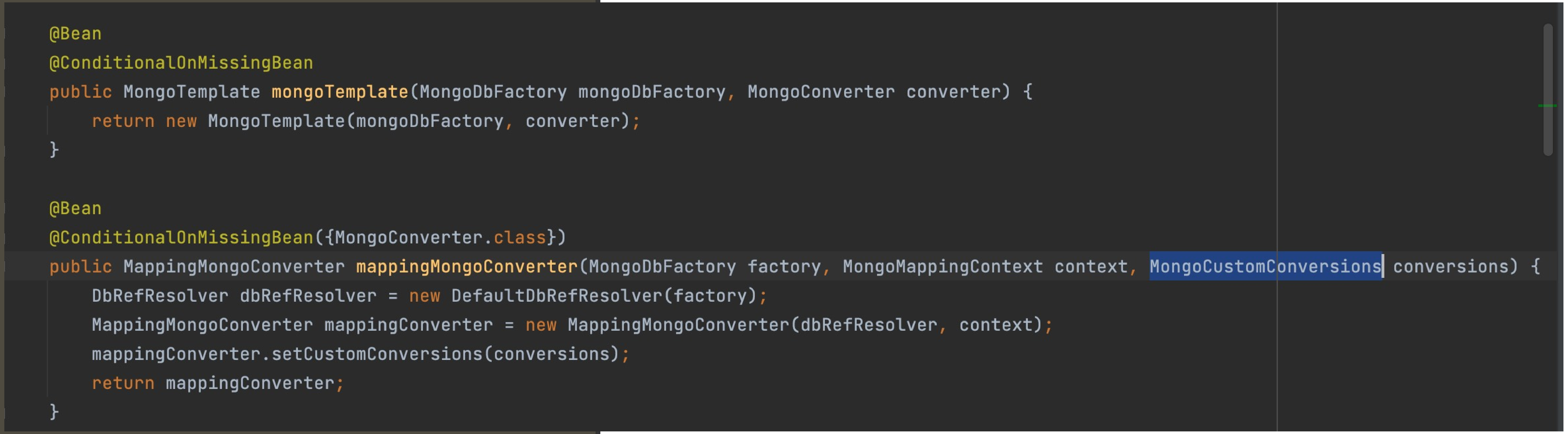
@EnableMongoRepositories

对应接口

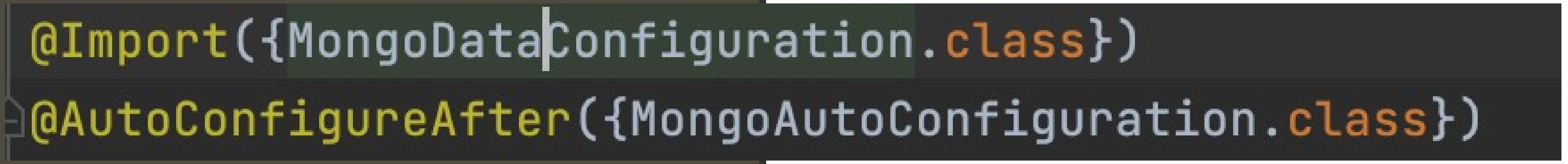
- `MongoRepository<T, ID>`
- `PagingAndSortingRepository<T, ID>`
- `CrudRepository<T, ID>`



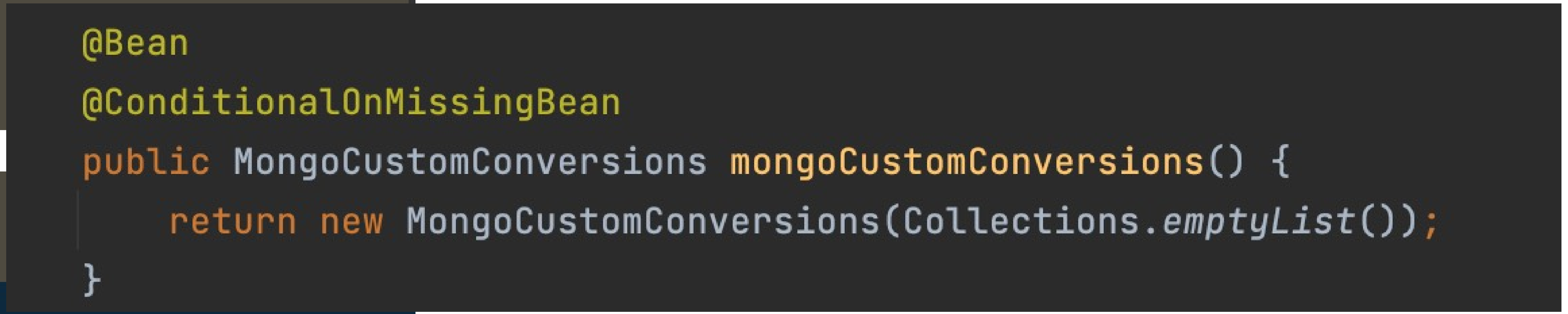
```
Maven: org.springframework.boot:spring-boot-autoconfigure:2.1.2.RELEASE
  spring-boot-autoconfigure-2.1.2.RELEASE.jar library root
    META-INF
    org.springframework.boot.autoconfigure
      admin
      amqp
      aop
      batch
      cache
      cassandra
      cloud
      condition
      context
      couchbase
      dao
      data
        cassandra
        couchbase
        elasticsearch
        jdbc
        jpa
        ldap
        mongo
        neo4j
        redis
        rest
```



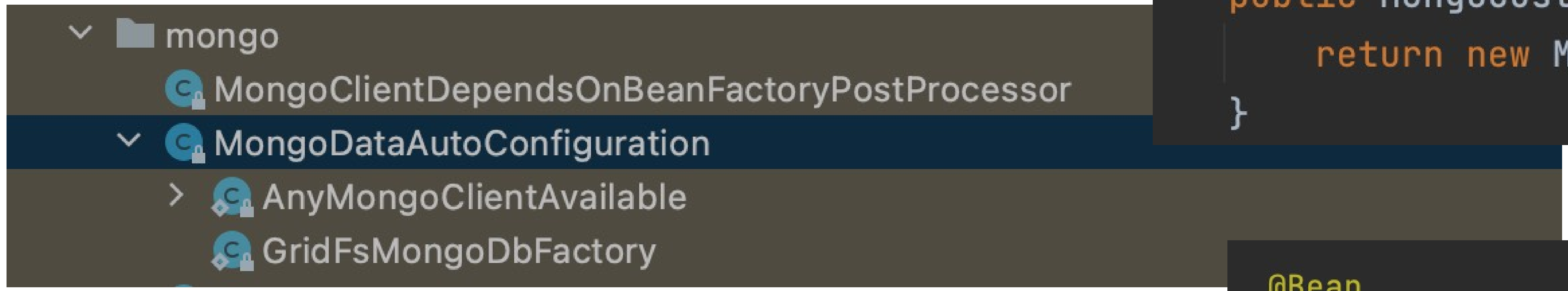
```java
@Bean
@ConditionalOnMissingBean
public MongoTemplate mongoTemplate(MongoDbFactory mongoDbFactory, MongoConverter converter) {
    return new MongoTemplate(mongoDbFactory, converter);
}

@Bean
@ConditionalOnMissingBean({MongoConverter.class})
public MappingMongoConverter mappingMongoConverter(MongoDbFactory factory, MongoMappingContext context, MongoCustomConversions conversions) {
    DbRefResolver dbRefResolver = new DefaultDbRefResolver(factory);
    MappingMongoConverter mappingConverter = new MappingMongoConverter(dbRefResolver, context);
    mappingConverter.setCustomConversions(conversions);
    return mappingConverter;
}
```



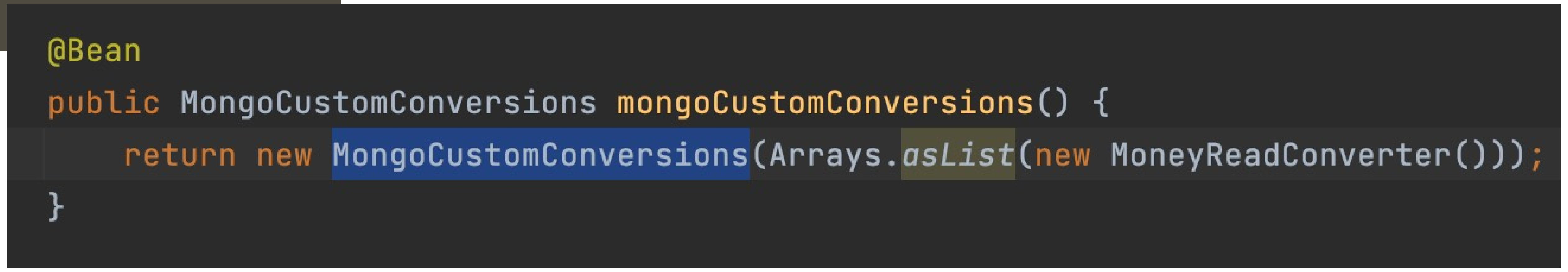
```java
@Import({MongoDataConfiguration.class})
@AutoConfigureAfter({MongoAutoConfiguration.class})
```



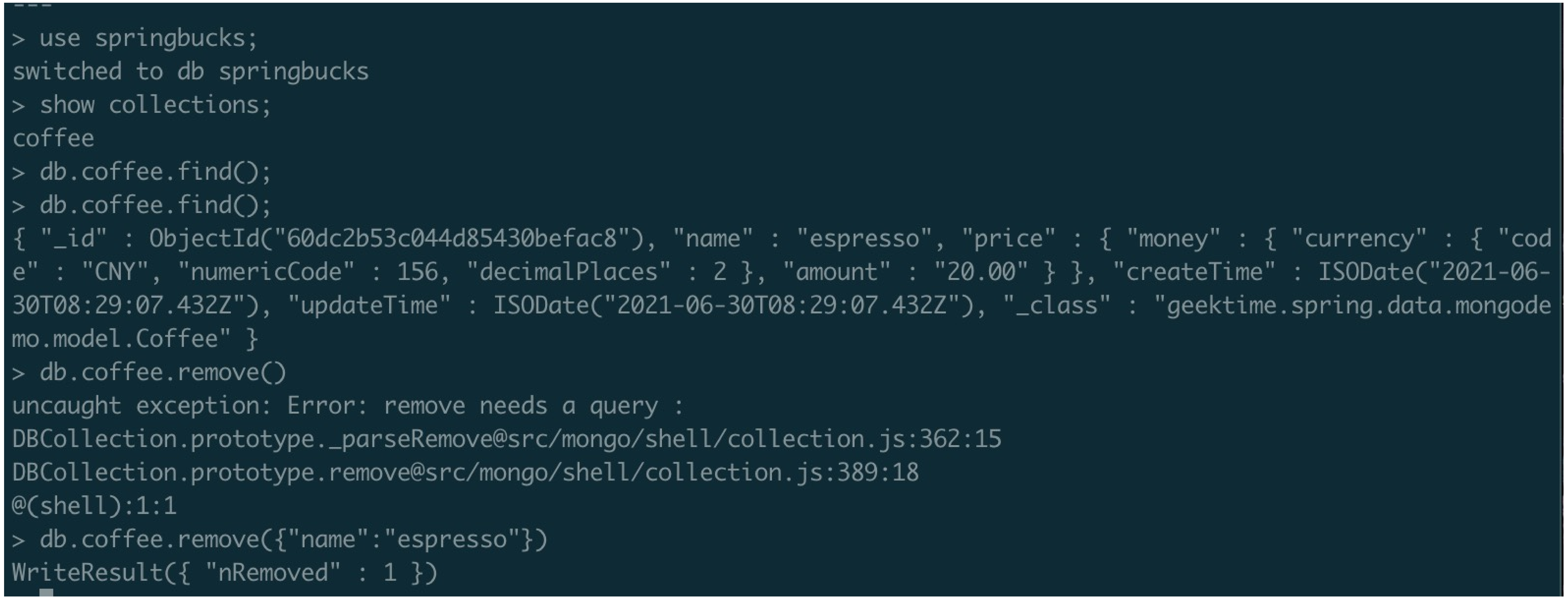
```java
@Bean
@ConditionalOnMissingBean
public MongoCustomConversions mongoCustomConversions() {
    return new MongoCustomConversions(Collections.emptyList());
}
```



```
mongo
  MongoClientDependsOnBeanFactoryPostProcessor
  MongoDataAutoConfiguration
    AnyMongoClientAvailable
    GridFsMongoDbFactory
```



```java
@Bean
public MongoCustomConversions mongoCustomConversions() {
    return new MongoCustomConversions(Arrays.asList(new MoneyReadConverter()));
}
```



```
> use springbucks;
switched to db springbucks
> show collections;
coffee
> db.coffee.find();
> db.coffee.find();
{ "_id" : ObjectId("60dc2b53c044d85430befac8"), "name" : "espresso", "price" : { "money" : { "currency" : { "cod
e" : "CNY", "numericCode" : 156, "decimalPlaces" : 2 }, "amount" : "20.00" } }, "createTime" : ISODate("2021-06-
30T08:29:07.432Z"), "updateTime" : ISODate("2021-06-30T08:29:07.432Z"), "_class" : "geektime.spring.data.mongode
mo.model.Coffee" }
> db.coffee.remove()
uncaught exception: Error: remove needs a query :
DBCollection.prototype._parseRemove@src/mongo/shell/collection.js:362:15
DBCollection.prototype.remove@src/mongo/shell/collection.js:389:18
@(shell):1:1
> db.coffee.remove({"name":"espresso"})
WriteResult({ "nRemoved" : 1 })
```

Redis两种部署方式

在 Spring 中访问 Redis

Spring 对 Redis 的支持

Redis 是一款开源的内存 KV 存储，支持多种数据结构

- https://redis.io

Spring 对 Redis 的支持

- Spring Data Redis
    - 支持的客户端 Jedis / Lettuce
    - RedisTemplate
    - Repository 支持

通常当做缓存来用，不作为持久化

Jedis 客户端的简单使用

- Jedis 不是线程安全的　也就是说不通线程不能共享Jedis实例
- 通过 JedisPool 获得 Jedis 实例
- 直接使用 Jedis 中的方法

Jedis 客户端的简单使用

```java
@Bean
@ConfigurationProperties("redis")
public JedisPoolConfig jedisPoolConfig() {
        return new JedisPoolConfig();
}

@Bean(destroyMethod = "close")
public JedisPool jedisPool(@Value("${redis.host}") String host) {
        return new JedisPool(jedisPoolConfig(), host);
}
```

通过 Docker 启动 Redis

官方指引

- https://hub.docker.com/_/redis

获取镜像

- `docker pull redis`

启动 Redis

- `docker run --name redis -d -p 6379:6379 redis`

“Talk is cheap, show me the code.”

Chapter 4 / jedis-demo

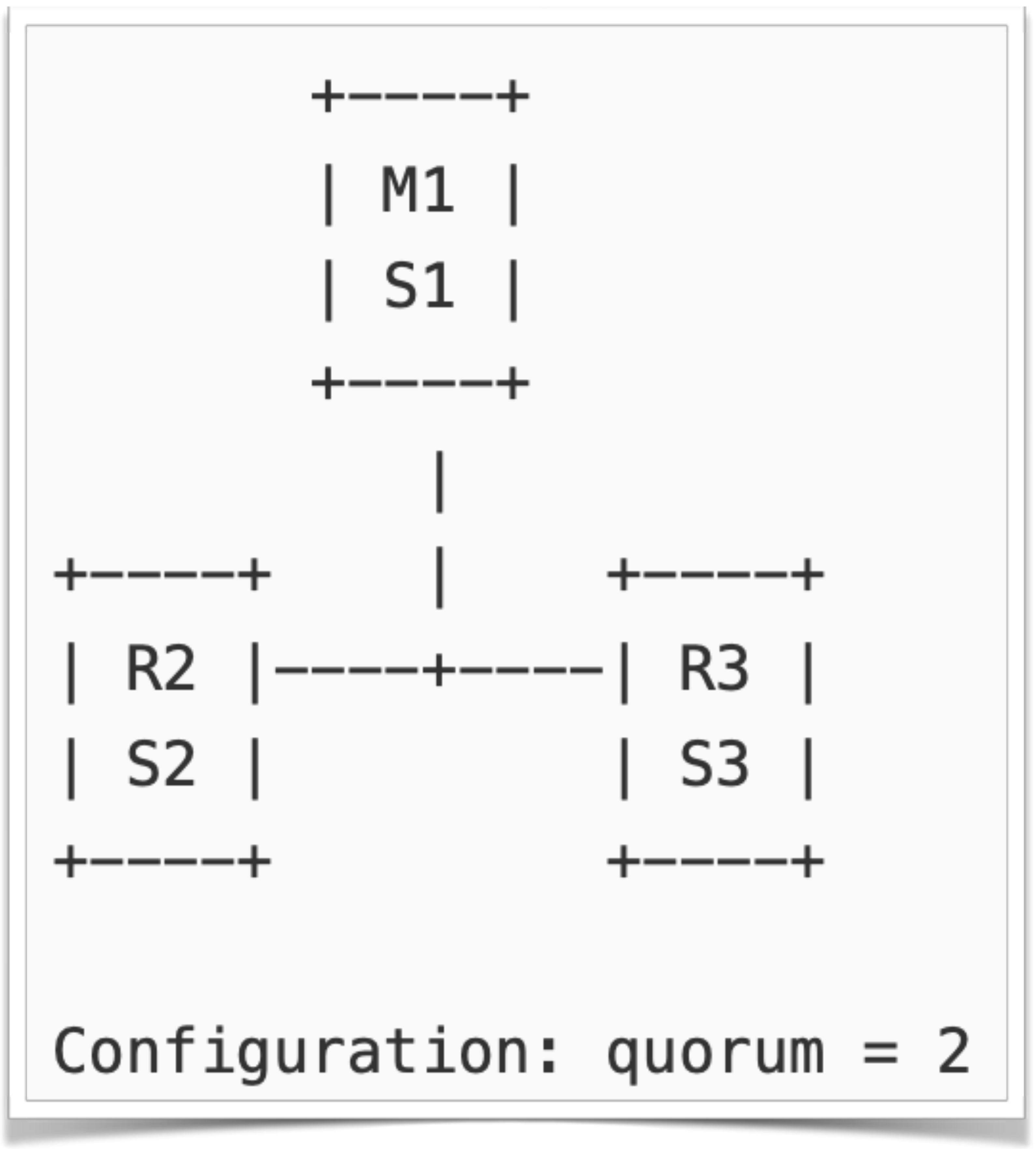
Redis 的哨兵与集群模式

Redis 的哨兵模式

Redis Sentinel 是 Redis 的一种高可用方案

- 监控、通知、自动故障转移、服务发现

JedisSentinelPool



```
       +----+
       | M1 |
       | S1 |
       +----+
          |
+----+    |    +----+
| R2 |----+----| R3 |
| S2 |         | S3 |
+----+         +----+

Configuration: quorum = 2
```

Redis 的集群模式

单节点时候可以对key进行批量操作，但是到集群上就不行了，因为key可能分散在不同cluster上

Redis Cluster

- 数据自动分片（分成16384个 Hash Slot ）
- 在部分节点失效时有一定可用性

Lettuce可以满足你的需求，另外Lettuce是基于netty的，性能比较好，是一个不错的redis client

JedisCluster

Lettuce是可以的

- Jedis 只从 Master 读数据，如果想要自动读写分离，可以定制

RedisCluster that can use ReadOnly slaves

了解 Spring 的缓存抽象

Spring 的缓存抽象

为不同的缓存提供一层抽象

- 为 Java 方法增加缓存，缓存执行结果
- 支持ConcurrentMap、EhCache、Caffeine、JCache（JSR-107）
- 接口
  - `org.springframework.cache.Cache`
  - `org.springframework.cache.CacheManager`

1.数据很长时间不变，可以接受变化有一定延时，那么可以放到界面内部，设置一个过期时间，过期后再去后端捞取
2.如果希望缓存具有一致性的，可以使用分布式缓存Redis
3.写一次读一次的，没有必要缓存。如果基本都是读，请务必使用缓存

基于注解的缓存

@EnableCaching

- @Cacheable
- @CacheEvict 缓存清理
- @CachePut
- @Caching 缓存清理、设置等进行打包
- @CacheConfig

“Talk is cheap, show me the code.”

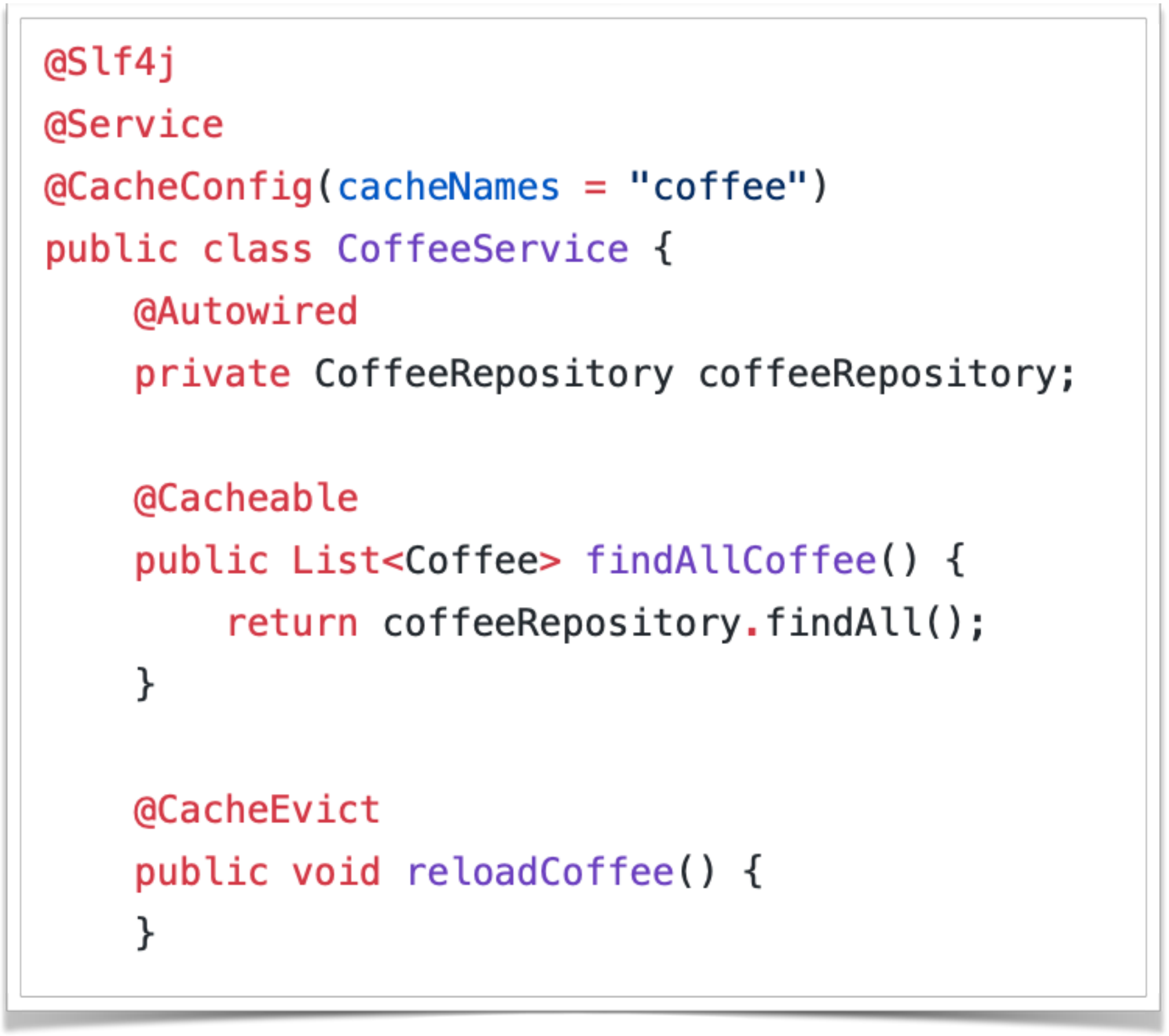
Chapter 4 / cache-demo

通过 Spring Boot 配置 Redis 缓存

```xml
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

```properties
spring.cache.type=redis
spring.cache.cache-names=coffee
spring.cache.redis.time-to-live=5000
spring.cache.redis.cache-null-values=false

spring.redis.host=localhost
```

通过 Spring Boot 配置 Redis 缓存



```
@Slf4j
@Service
@CacheConfig(cacheNames = "coffee")
public class CoffeeService {
    @Autowired
    private CoffeeRepository coffeeRepository;

    @Cacheable
    public List<Coffee> findAllCoffee() {
        return coffeeRepository.findAll();
    }

    @CacheEvict
    public void reloadCoffee() {
    }
```

“Talk is cheap, show me the code.”

Chapter 4 / cache-with-redis-demo

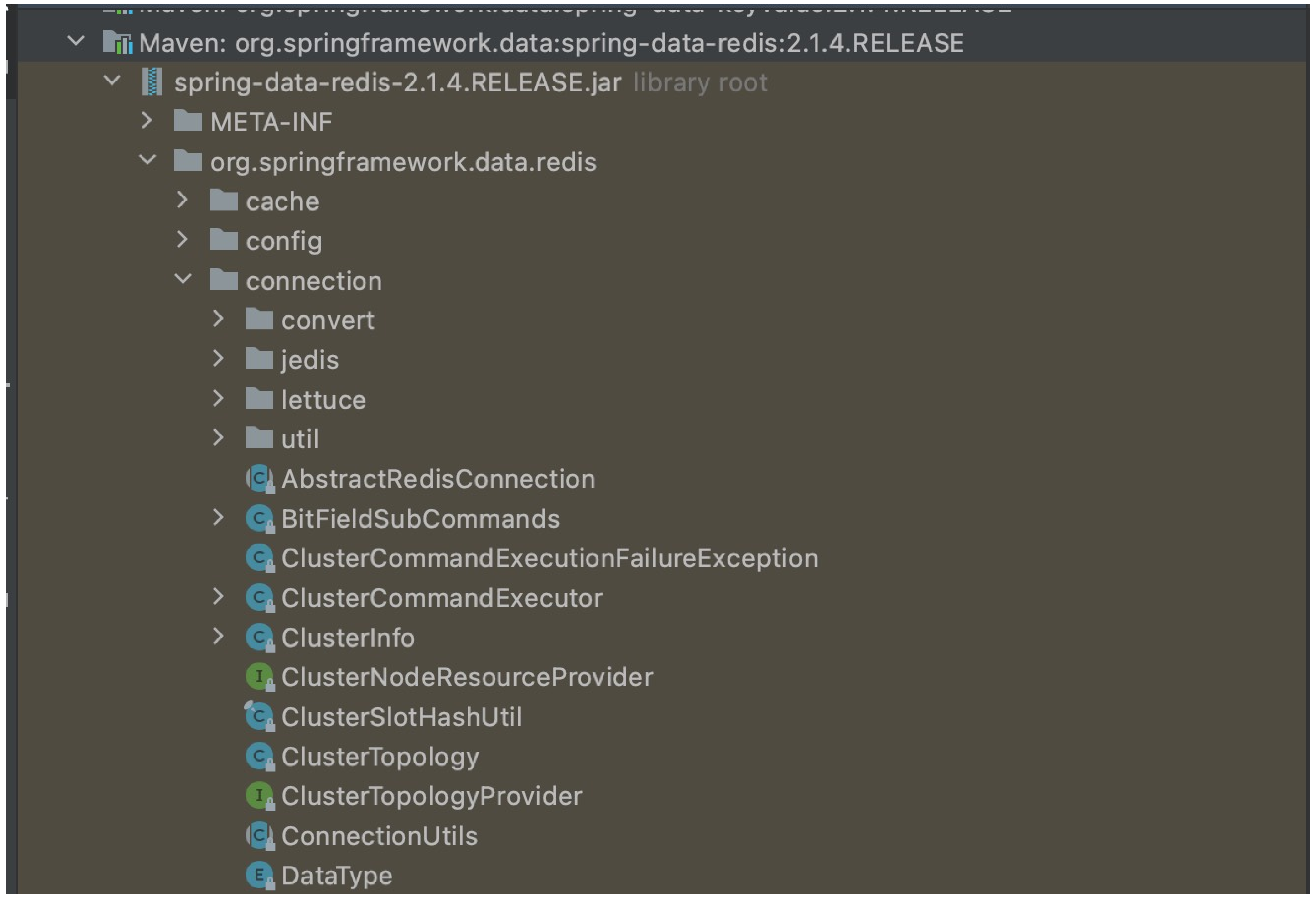
Redis 在 Spring 中的其他用法

与 Redis 建立连接

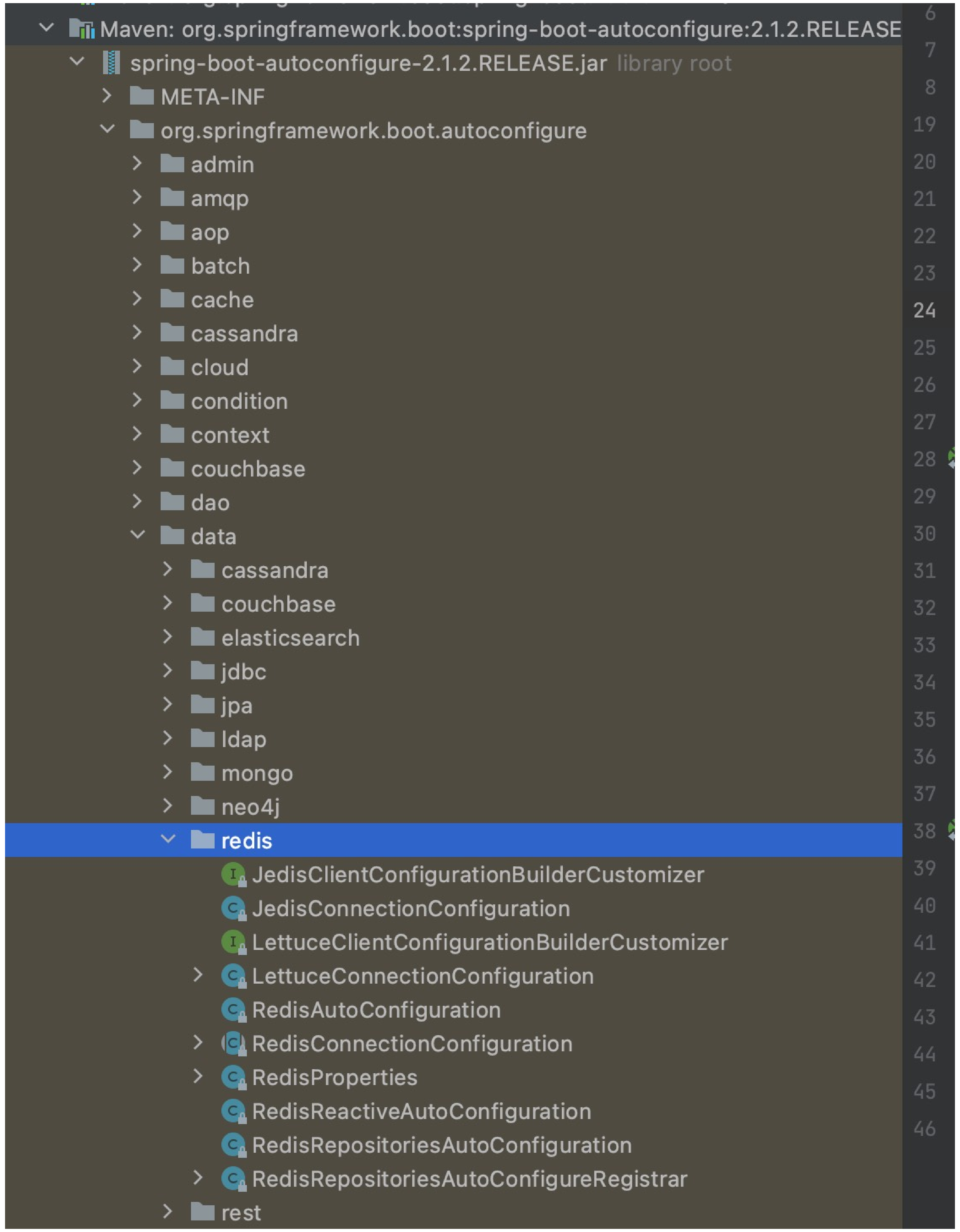
新版本已经使用lettuce作为默认客户端

配置连接工厂

- LettuceConnectionFactory 与 JedisConnectionFactory
  - RedisStandaloneConfiguration
  - RedisSentinelConfiguration
  - RedisClusterConfiguration



```
Maven: org.springframework.data:spring-data-redis:2.1.4.RELEASE
  spring-data-redis-2.1.4.RELEASE.jar library root
    META-INF
    org.springframework.data.redis
      cache
      config
      connection
        convert
        jedis
        lettuce
        util
        AbstractRedisConnection
        BitFieldSubCommands
        ClusterCommandExecutionFailureException
        ClusterCommandExecutor
        ClusterInfo
        ClusterNodeResourceProvider
        ClusterSlotHashUtil
        ClusterTopology
        ClusterTopologyProvider
        ConnectionUtils
        DataType
```



```
Maven: org.springframework.boot:spring-boot-autoconfigure:2.1.2.RELEASE
  spring-boot-autoconfigure-2.1.2.RELEASE.jar library root
    META-INF
    org.springframework.boot.autoconfigure
      admin
      amqp
      aop
      batch
      cache
      cassandra
      cloud
      condition
      context
      couchbase
      dao
      data
        cassandra
        couchbase
        elasticsearch
        jdbc
        jpa
        ldap
        mongo
        neo4j
        redis
          JedisClientConfigurationBuilderCustomizer
          JedisConnectionConfiguration
          LettuceClientConfigurationBuilderCustomizer
          LettuceConnectionConfiguration
          RedisAutoConfiguration
          RedisConnectionConfiguration
          RedisProperties
          RedisReactiveAutoConfiguration
          RedisRepositoriesAutoConfiguration
          RedisRepositoriesAutoConfigureRegistrar
        rest
```

读写分离

Lettuce **内置支持读写分离**

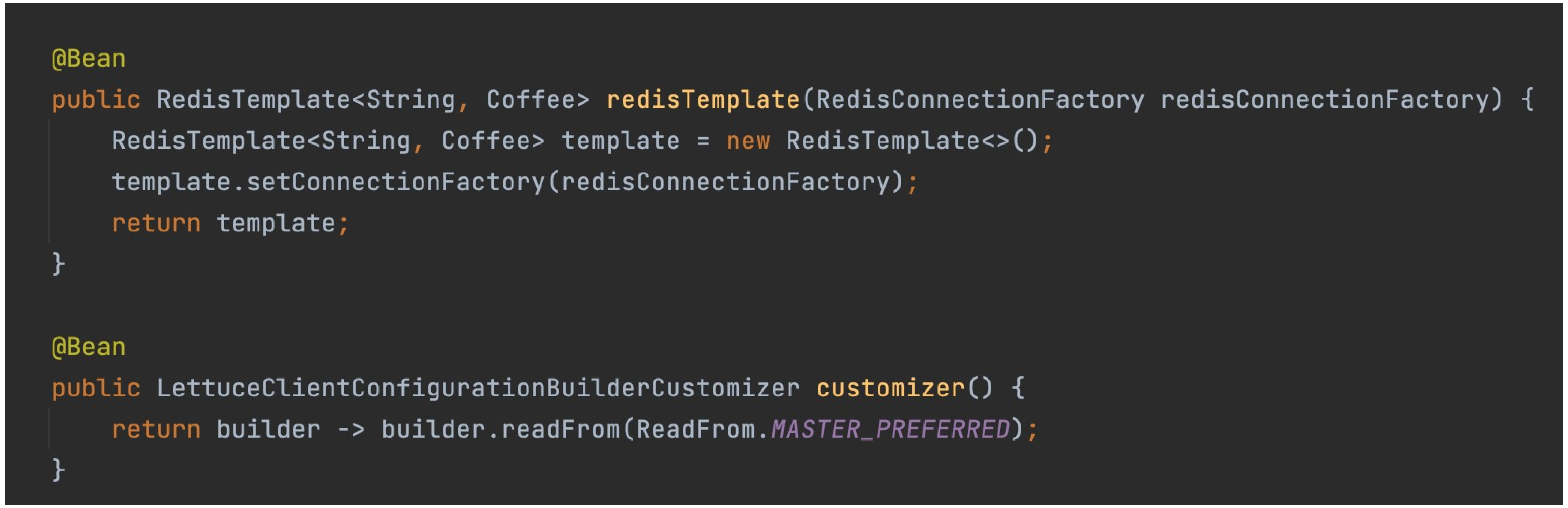
- 只读主、只读从
- 优先读主、优先读从

LettuceClientConfiguration

LettucePoolingClientConfiguration

LettuceClientConfigurationBuilderCustomizer 回调配置

先定义一个自己需要的类型的redisTemplate
然后通过LettuceClientConfigurationBuilderCustomizer回调



```java
@Bean
public RedisTemplate<String, Coffee> redisTemplate(RedisConnectionFactory redisConnectionFactory) {
    RedisTemplate<String, Coffee> template = new RedisTemplate<>();
    template.setConnectionFactory(redisConnectionFactory);
    return template;
}

@Bean
public LettuceClientConfigurationBuilderCustomizer customizer() {
    return builder -> builder.readFrom(ReadFrom.MASTER_PREFERRED);
}
```

这里优先读主节点的

RedisTemplate

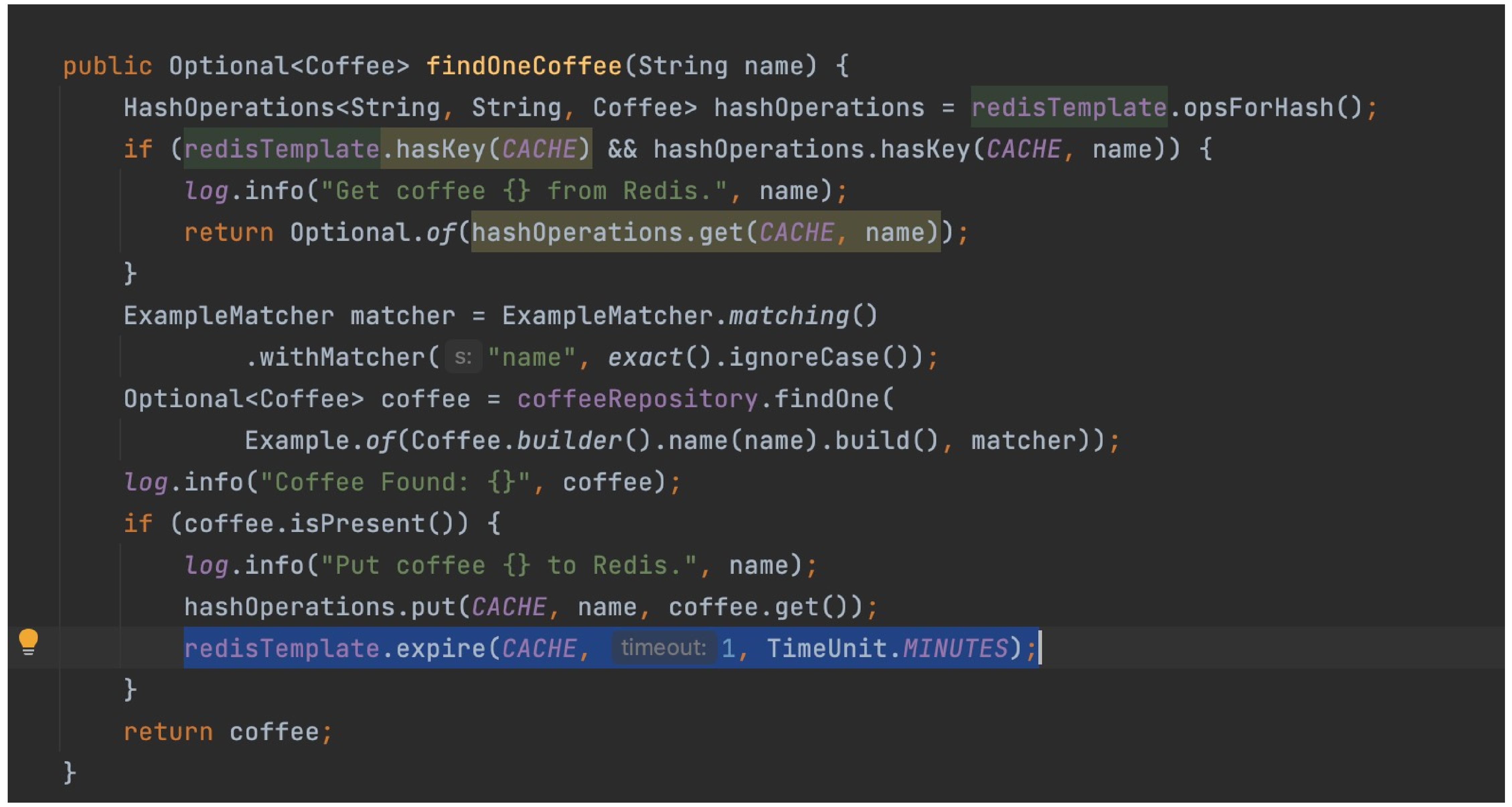
key,value的类型

RedisTemplate<K, V>

- opsForXxx()

  后面接类型，例如opsForHash

StringRedisTemplate



```java
public Optional<Coffee> findOneCoffee(String name) {
    HashOperations<String, String, Coffee> hashOperations = redisTemplate.opsForHash();
    if (redisTemplate.hasKey(CACHE) && hashOperations.hasKey(CACHE, name)) {
        log.info("Get coffee {} from Redis.", name);
        return Optional.of(hashOperations.get(CACHE, name));
    }
    ExampleMatcher matcher = ExampleMatcher.matching()
            .withMatcher( s: "name", exact().ignoreCase());
    Optional<Coffee> coffee = coffeeRepository.findOne(
            Example.of(Coffee.builder().name(name).build(), matcher));
    log.info("Coffee Found: {}", coffee);
    if (coffee.isPresent()) {
        log.info("Put coffee {} to Redis.", name);
        hashOperations.put(CACHE, name, coffee.get());
        redisTemplate.expire(CACHE,  timeout: 1, TimeUnit.MINUTES);
    }
    return coffee;
}
```

一定注意设置过期时间！！！

“Talk is cheap, show me the code.”

Chapter 4 / redis-demo

Redis Repository

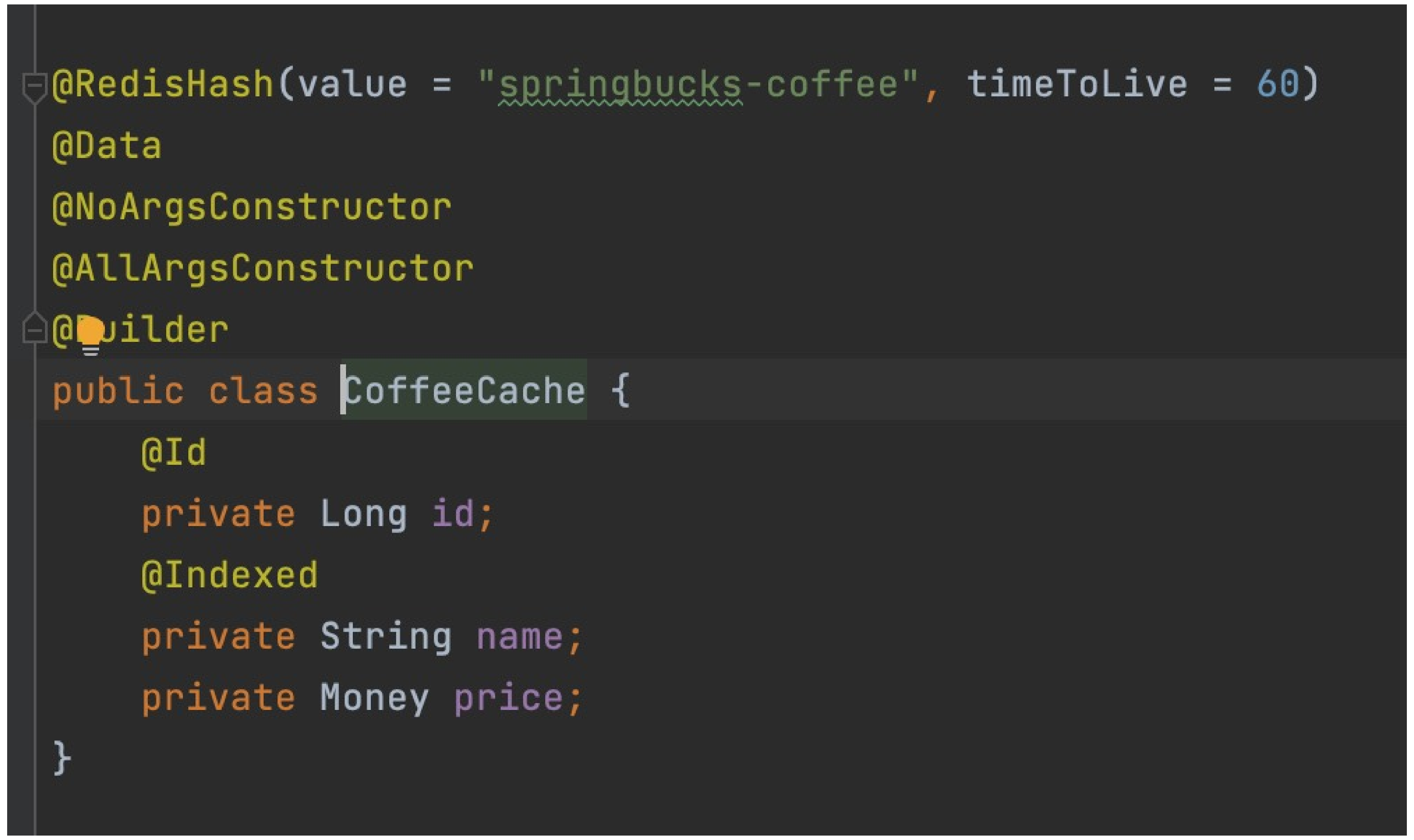
实体注解

- @RedisHash　　类似于@Entity的Class
- @Id
- @Indexed　　二级索引

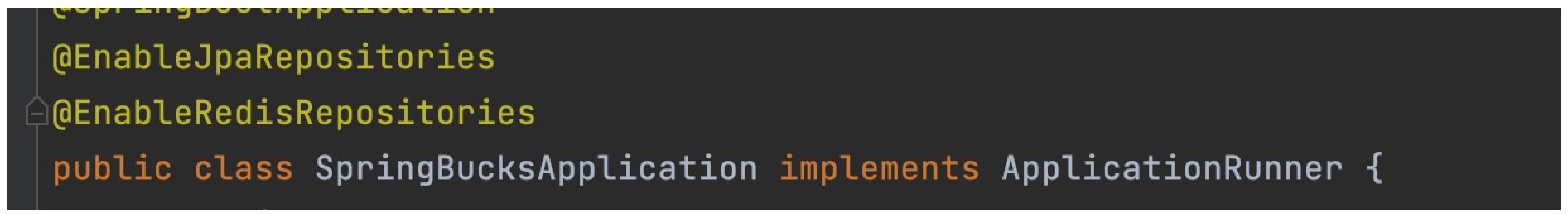
处理不同类型数据源的 Repository

如何区分这些 Repository

- 根据实体的注解
- 根据继承的接口类型
- 扫描不同的包



```java
@RedisHash(value = "springbucks-coffee", timeToLive = 60)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CoffeeCache {
    @Id
    private Long id;
    @Indexed
    private String name;
    private Money price;
}
```



```java
@SpringBootApplication
@EnableJpaRepositories
@EnableRedisRepositories
public class SpringBucksApplication implements ApplicationRunner {
```

“Talk is cheap, show me the code.”

Chapter 4 / redis-repository-demo

SpringBucks 进度小结

本章小结

- 了解了 Docker 在本地的基本用法
- 了解了 Spring Data MongoDB 的基本用法
- 了解了 Spring Data Redis 的基本用法
- 了解了 Redis 的几种运行模式
- 了解了 Spring 的缓存抽象

SpringBucks 进度小结

- 使用不同类型的数据库存储咖啡信息
- 结合 JPA 与 Redis 来优化咖啡信息的存储