Spring Data Redis

Spring Data

- Salah satu kelebihan Spring adalah ekosistemnya yang sudah sangat besar, contohnya adalah Spring memiliki ekosistem bernama Spring Data
- Sebelumnya kita pernah belajar tentang Spring Data JPA, yang menjadi salah satu ekosistem dari Spring Data
- Spring Data adalah ekosistem untuk berkomunikasi dengan berbagai jenis database, seperti Relational Database, MongoDB, Redis, Cassandra, Elasticsearch, dan lain-lain
- https://spring.io/projects/spring-data

Spring Data Redis

- Salah satu database yang bisa kita gunakan di ekosistem Spring Data, adalah Redis
- Spring Data Redis adalah fitur di Spring Data yang bisa kita gunakan untuk memudahkan kita berkomunikasi dengan basis data Redis
- Seperti yang kita tahu, Redis adalah salah satu database In Memory yang sangat populer
- Banyak sekali struktur data yang bisa kita gunakan di Redis
- Di materi ini, kita akan bahas bagaimana menggunakan berbagai macam struktur data di Redis menggunakan Spring Data Redis

Membuat Project

Menjalankan Redis

- Pastikan sudah menjalankan Redis Server terlebih dahulu

Membuat Project

- https://start.spring.io/
- Spring Web
- Spring Redis
- Spring Actuator
- Lombok

Configuration

Configuration

- Sebelum menggunakan Spring Data Redis, kita perlu melakukan konfigurasi terlebih dahulu agar aplikasi Spring Boot kita, bisa terkoneksi ke database Redis
- Untuk melakukan konfigurasi, kita cukup menggunakan application properties dengan prefix spring.data.redis
- https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.data.spring.data.redis.client-name

Kode : Application Properties

```
spring.data.redis.host=localhost
spring.data.redis.port=6379
spring.data.redis.database=0
spring.data.redis.timeout=5s
spring.data.redis.connect-timeout=5s
#spring.data.redis.username=redis
#spring.data.redis.password=redis
```

Redis Template

Redis Template

- Saat kita menggunakan Spring Data Redis, secara otomatis Spring Boot akan membuat sebuah bean dengan type RedisTemplate
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/RedisTemplate.html
- Redis Template merupakan tipe data generic, dan salah satu implementasinya yang biasa digunakan adalah StringRedisTemplate
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/StringRedisTemplate.html

Kode : Redis Template

```
@SpringBootTest
class StringTest {

  @Autowired
  private StringRedisTemplate redisTemplate;

  new *
  @Test
  void redisTemplate() {
    assertNotNull( actual: redisTemplate);
  }
}
```

Value Operation

Value Operation

- Struktur data yang biasa digunakan saat kita menggunakan Redis adalah String
- Dimana kita bisa menggunakan Key-Value berupa String di Redis
- Untuk berinteraksi dengan struktur data String, kita bisa menggunakan ValueOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/ValueOperations.html

Kode : Value Operation

```
@Test
void string() throws InterruptedException {
    ValueOperations<String, String> operations = redisTemplate.opsForValue();

    operations.set( key: "name",  value: "Eko",  timeout: Duration.ofSeconds( seconds: 2));
    assertEquals( expected: "Eko",  actual: operations.get("name"));

    Thread.sleep( duration: Duration.ofSeconds( seconds: 3));
    assertNull( actual: operations.get("name"));
}
}
```

List Operation

List Operation

- Untuk berinteraksi dengan struktur data List di Redis, kita bisa menggunakan ListOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/ListOperations.html

Kode : List Operation

```
@Test
void list() {
    ListOperations<String, String> operations = redisTemplate.opsForList();
    operations.rightPush("names", "Eko");
    operations.rightPush("names", "Kurniawan");
    operations.rightPush("names", "Khannedy");

    assertEquals( expected: "Eko",  actual: operations.leftPop( key: "names"));
    assertEquals( expected: "Kurniawan",  actual: operations.leftPop( key: "names"));
    assertEquals( expected: "Khannedy",  actual: operations.leftPop( key: "names"));
}
}
```

Set Operation

Set Operation

- Untuk berinteraksi dengan struktur data Set di Redis, kita bisa menggunakan SetOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/SetOperations.html

Kode : Set Operation

```
@Test
void set() {
  SetOperations<String, String> operations = redisTemplate.opsForSet();
  operations.add( key: "students",  ...values: "Eko");
  operations.add( key: "students",  ...values: "Eko");
  operations.add( key: "students",  ...values: "Kurniawan");
  operations.add( key: "students",  ...values: "Kurniawan");
  operations.add( key: "students",  ...values: "Khannedy");
  operations.add( key: "students",  ...values: "Khannedy");

  assertEquals( expected: 3,  actual: operations.members( key: "students").size());
  assertThat( actual: operations.members( key: "students"),  matcher: hasItems( ...items: "Eko", "Kurniawan", "Khannedy"));

  redisTemplate.delete( key: "students");
}
```

ZSet Operation

ZSet Operation

- Untuk berinteraksi dengan struktur data Sorted Set di Redis, kita bisa menggunakan ZSetOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/ZSetOperations.html

Kode : ZSet Operation

```
@Test
void zSet() {
  ZSetOperations<String, String> operations = redisTemplate.opsForZSet();
  operations.add( key: "score",  value: "Eko",  score: 100);
  operations.add( key: "score",  value: "Budi",  score: 85);
  operations.add( key: "score",  value: "Joko",  score: 95);

  assertEquals( expected: "Eko",  actual: operations.popMax( key: "score").getValue());
  assertEquals( expected: "Joko",  actual: operations.popMax( key: "score").getValue());
  assertEquals( expected: "Budi",  actual: operations.popMax( key: "score").getValue());
}
```

Hash Operation

Hash Operation

- Untuk berinteraksi dengan struktur data Hash di Redis, kita bisa menggunakan HashOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/HashOperations.html

Kode : Hash Operation

```java
@Test
void hash() {
    HashOperations<String, Object, Object> operations = redisTemplate.opsForHash();
    operations.put( key: "user:1",  hashKey: "id",  value: "1");
    operations.put( key: "user:1",  hashKey: "name",  value: "Eko");
    operations.put( key: "user:1",  hashKey: "email",  value: "eko@example.com");

    assertEquals( expected: "1",  actual: operations.get( key: "user:1",  hashKey: "id"));
    assertEquals( expected: "Eko",  actual: operations.get( key: "user:1",  hashKey: "name"));
    assertEquals( expected: "eko@example.com",  actual: operations.get( key: "user:1",  hashKey: "email"));

    redisTemplate.delete( key: "user:1");
}
```

Geo Operation

Geo Operation

- Untuk berinteraksi dengan struktur data Geo di Redis, kita bisa menggunakan GeoOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/GeoOperations.html

Kode : Geo Operation

```
@Test
void geo() {
  GeoOperations<String, String> operations = redisTemplate.opsForGeo();
  operations.add( key: "sellers",  point: new Point( x: 106.822702,  y: -6.177590),  member: "Toko A");
  operations.add( key: "sellers",  point: new Point( x: 106.820889,  y: -6.174964),  member: "Toko B");

  Distance distance = operations.distance( key: "sellers",  member1: "Toko A",  member2: "Toko B",  metric: Metrics.KILOMETERS);
  assertEquals( expected: 0.3543,  actual: distance.getValue());

  GeoResults<RedisGeoCommands.GeoLocation<String>> sellers = operations
      .search( key: "sellers",  within: new Circle(
          center: new Point( x: 106.821825,  y: -6.175105),
          radius: new Distance( value: 5,  metric: Metrics.KILOMETERS)));

  assertEquals( expected: 2,  actual: sellers.getContent().size());
  assertEquals( expected: "Toko A",  actual: sellers.getContent().get(0).getContent().getName());
  assertEquals( expected: "Toko B",  actual: sellers.getContent().get(1).getContent().getName());
}
```

Hyper Log Log Operation

Hyper Log Log Operation

- Untuk berinteraksi dengan struktur data Hyper Log Log di Redis, kita bisa menggunakan HyperLogLogOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/HyperLogLogOperations.html

Kode : Geo Operation

```
@Test
void hyperLogLog() {
    HyperLogLogOperations<String, String> operations = redisTemplate.opsForHyperLogLog();
    operations.add( key: "traffics",  ...values: "eko", "kurniawan", "khannedy");
    operations.add( key: "traffics",  ...values: "eko", "budi", "joko");
    operations.add( key: "traffics",  ...values: "budi", "joko", "rully");

    assertEquals( expected: 6L,  actual: operations.size( ...keys: "traffics"));
}
}
```

Transaction

Transaction

- Seperti kita tahu, bahwa Redis mendukung fitur Transaction dengan menggunakan perintah MULTI DAN EXEC
- Hal ini juga bisa dilakukan di Spring Data Redis dengan menggunakan RedisTemplate
- Namun, agar menggunakan data koneksi ke redis yang sama, maka kita perlu menggunakan perintah RedisTemplate.execute()

Kode : Transaction

```
@Test
void transaction() {
    new *
    redisTemplate.execute(new SessionCallback<>() {
        new *
        @Override
        public Object execute(RedisOperations operations) throws DataAccessException {
            operations.multi();
            operations.opsForValue().set("test1", "Eko");
            operations.opsForValue().set("test2", "Budi");
            operations.exec();
            return null;
        }
    });

    assertEquals("Eko", redisTemplate.opsForValue().get("test1"));
    assertEquals("Budi", redisTemplate.opsForValue().get("test2"));
}
}
```

Pipeline

Pipeline

- Di kelas Redis, kita pernah belajar tentang pipeline, dimana kita bisa mengirim beberapa perintah secara langsung tanpa harus menunggu balasan satu per satu dari Redis
- Hal ini juga bisa dilakukan menggunakan Spring Data Redis menggunakan RedisTemplate.executePipelined()
- Return dari executePipelined() akan berisikan List status dari tiap perintah yang kita lakukan

Kode : Pipeline

```
void pipeline() {
    new *
    List<Object> list = redisTemplate.executePipelined(new SessionCallback<>() {
        new *
        @Override
        public Object execute(RedisOperations operations) throws DataAccessException {
            operations.opsForValue().set("test1", "Eko");
            operations.opsForValue().set("test2", "Eko");
            operations.opsForValue().set("test3", "Eko");
            operations.opsForValue().set("test4", "Eko");
            return null;
        }
    });

    assertThat(list, hasSize(4));
    assertThat(list, hasItem(true));
    assertThat(list, not(hasItem(false)));
}
```

Stream Operation

Stream Operation

- Untuk berinteraksi dengan struktur data Stream di Redis, kita bisa menggunakan StreamOperations class
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/StreamOperations.html

Kode : Publish Stream

```
@Test
void publish() {
  var operations = redisTemplate.opsForStream();
  var record = MapRecord.create("stream-1", Map.of(
      "name", "Eko Kurniawan",
      "address", "Indonesia"
  ));

  for (int i = 0; i < 10; i++) {
    operations.add(record);
  }
}
```

Kode : Subscribe Stream

```java
@Test
void subscribe() {
  var operations = redisTemplate.opsForStream();
  try {
    operations.createGroup("stream-1", "sample-group");
  } catch (RedisSystemException exception) {
    // group already exists
  }

  List<MapRecord<String, Object, Object>> records = operations.read(Consumer.from("sample-group", "sample-1"),
      StreamOffset.create("stream-1", ReadOffset.lastConsumed()));

  for (MapRecord<String, Object, Object> record : records) {
    System.out.println(record);
  }
}
```

Stream Listener

Listener

- Saat menggunakan StreamOperations secara manual, maka kita harus melakukan read secara manual tiap kali ada data masuk ke Stream
- Spring Data Redis memiliki fitur bernama Stream Listener, yang bisa kita buat untuk membaca ketika terdapat data di Stream secara otomatis menggunakan konsep Event Listener seperti yang sudah kita pelajari di Spring Dasar
- Untuk membuat Stream Listener agak sedikit kompleks, yang lebih mudah sebenarnya menggunakan Spring PubSub, namun di PubSub sayang nya tidak memiliki fitur Consumer Group

Stream Listener

- Untuk membuat Listener nya, kita harus membuat implementasi turunan dari StreamListener
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/stream/StreamListener.html

Kode : Order

```
no usages  new *
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Order {

  private String id;

  private Long amount;
}

```

Kode : Order Listener

```java
new *
@Slf4j
@Component
public class OrderListener implements StreamListener<String, ObjectRecord<String, Order>> {

    new *
    @Override
    public void onMessage( @NotNull ObjectRecord<String, Order> message) {
        log.info("Receive order: {}", message.getValue());
    }
}
```

Stream Message Listener Container

- Untuk menjalankan Stream Listener, kita harus membuat Container (tempat) untuk menjalankan semua Stream Listener tersebut
- Kita bisa membuat Bean dengan type StreamMessageListenerContainer
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/stream/StreamMessageListenerContainer.html

Kode : Listener Container

```
new *
@Bean(destroyMethod = "stop", initMethod = "start")
public StreamMessageListenerContainer<String, ObjectRecord<String, Order>> orderContainer(RedisConnectionFactory connectionFactory) {
  var options = StreamMessageListenerContainer.StreamMessageListenerContainerOptions
      .builder()
      .pollTimeout(Duration.ofSeconds(5))
      .targetType(Order.class)
      .build();

  return StreamMessageListenerContainer.create(connectionFactory, options);
}
```

Subscription

- Setelah kita membuat Listener Container, kita bisa membuat Subscriber dengan menggunakan bean Stream Listener yang sudah kita buat
- Caranya kita perlu meregistrasikan Stream Listener ke Listener Container
- Hasilnya adalah sebuah Subscription
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/connection/Subscription.html

Kode : Subscription

```java
@Bean
public Subscription helloSubscription(StreamMessageListenerContainer<String, ObjectRecord<String, Order>> orderContainer,
                                      OrderListener helloStream) {
  try {
    redisTemplate.opsForStream().createGroup("orders", "my-group");
  } catch (Throwable throwable) {
    // consumer group already exists
  }

  var offset = StreamOffset.create("orders", ReadOffset.lastConsumed());
  var consumer = Consumer.from("my-group", "consumer-1");
  var readRequest = StreamMessageListenerContainer.StreamReadRequest.builder(offset) StreamReadRequestBuilder<String>
      .consumer(consumer) ConsumerStreamReadRequestBuild...<String>
      .autoAcknowledge(true)
      .cancelOnError(throwable -> false)
      .errorHandler(throwable -> log.warn(throwable.getMessage()))
      .build();

  return orderContainer.register(readRequest, helloStream);
}
```

Kode : Publisher

```java
@Component
public class OrderPublisher {

  @Autowired
  private StringRedisTemplate redisTemplate;

  new *
  @Scheduled(fixedRate = 10, timeUnit = TimeUnit.SECONDS)
  public void run() {
    Order order = new Order(UUID.randomUUID().toString(), 1000L);
    ObjectRecord<String, Order> record = ObjectRecord.create("orders", order);
    redisTemplate.opsForStream().add(record);
  }
}
```

PubSub

Pub Sub

- Khusus untuk fitur Redis PubSub, tidak terdapat class Operation
- Kita bisa langsung menggunakan RedisTemplate.convertAndSend() untuk mengirim ke PubSub
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/core/RedisTemplate.html#convertAndSend(java.lang.String,java.lang.Object)
- Dan RedisConnection.subscribe() untuk membuat Subscriber di PubSub
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/connection/RedisPubSubCommands.html#subscribe(org.springframework.data.redis.connection.MessageListener,byte%5B%5D...)

Kode : PubSub

```java
@Test
void pubSub() {
    new *
    redisTemplate.getConnectionFactory().getConnection().subscribe(new MessageListener() {
        new *
        @Override
        public void onMessage(Message message, byte[] pattern) {
            System.out.println("Message: " + new String(message.getBody()));
        }
    }, "my-channel".getBytes());

    for (int i = 0; i < 10; i++) {
        redisTemplate.convertAndSend("my-channel", "Hello World");
    }
}
```

PubSub Listener

PubSub Listener

- Sama seperti Stream Listener, Spring Data Redis juga bisa membuat Listener untuk PubSub
- Caranya hampir sama, kita perlu membuat Message Listener Container
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/listener/RedisMessageListenerContainer.html
- Lalu Membuat Message Container
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/connection/MessageListener.html

Kode : Listener

```
@Slf4j
@Component
public class CustomerListener implements MessageListener {

    new *
    @Override
    public void onMessage( @NotNull Message message, byte[] pattern) {
        log.info("Receive message: {}", new String(message.getBody()));
    }
}
```

Kode : Listener Container

```
new *
@Bean
public RedisMessageListenerContainer messageListenerContainer(RedisConnectionFactory connectionFactory,
                                                               CustomerListener customerListener) {
  var container = new RedisMessageListenerContainer();
  container.setConnectionFactory(connectionFactory);
  container.addMessageListener(customerListener, new ChannelTopic("customers"));
  return container;
}
}
```

Kode : Publisher

```java
@Component
public class CustomerPublisher {

  @Autowired
  private StringRedisTemplate redisTemplate;

  new *
  @Scheduled(fixedRate = 10L, timeUnit = TimeUnit.SECONDS)
  public void publish() {
    redisTemplate.convertAndSend("customers", "Eko " + UUID.randomUUID().toString());
  }
}
```

Collection

Java Collection

- Saat kita menggunakan Java, kita tahu bahwa hampir kebanyakan struktur data di Redis, itu sudah ada di Java, seperti java.util.List, java.util.Set, dan java.util.Map
- Spring Data Redis, memiliki fitur bisa melakukan konversi otomatis dari tipe data yang ada di Redis menjadi tipe data di Java
- Selain itu operasi yang kita lakukan di data tersebut, secara otomatis berdampak ke data yang ada di Redis nya secara otomatis

Spring Data Redis Collection

- Berikut adalah beberapa tipe data yang disediakan oleh Spring Data Redis agar bisa kompatibel dengan tipe data di Java Collection

Spring Data Redis	Java Collection	Redis Data Structure
RedisList	List	List
RedisSet	Set	Set
RedisZSet	Set	Sorted Set
RedisMap	Map	Hash

Kode : Redis List

```
@Test
void list() {
  List<String> list = RedisList.create("names", redisTemplate);
  list.add("Eko");
  list.add("Kurniawan");
  list.add("Khannedy");

  List<String> names = redisTemplate.opsForList().range("names", 0, -1);

  assertThat(list, hasItems("Eko", "Kurniawan", "Khannedy"));
  assertThat(names, hasItems("Eko", "Kurniawan", "Khannedy"));
}
}
```

Kode : Redis Set

```
@Test
void set() {
  Set<String> set = RedisSet.create("traffic", redisTemplate);
  set.addAll(Set.of("eko", "kurniawan", "khannedy"));
  set.addAll(Set.of("eko", "budi", "rully"));
  assertThat(set, hasItems("eko", "kurniawan", "khannedy", "budi", "rully"));

  Set<String> members = redisTemplate.opsForSet().members("traffic");
  assertThat(members, hasItems("eko", "kurniawan", "khannedy", "budi", "rully"));
}
```

Kode : Redis ZSet

```
@Test
void zset() {
    RedisZSet<String> set = RedisZSet.create("winner", redisTemplate);
    set.add("Eko", 100);
    set.add("Budi", 85);
    set.add("Joko", 95);
    assertThat(set, hasItems("Eko", "Budi", "Joko"));

    Set<String> members = redisTemplate.opsForZSet().range("winner", 0, -1);
    assertThat(members, hasItems("Eko", "Budi", "Joko"));

    assertEquals("Eko", set.popLast());
    assertEquals("Joko", set.popLast());
    assertEquals("Budi", set.popLast());
}
```

Kode : Redis Map

```
@Test
void map() {
    Map<String, String> map = new DefaultRedisMap<>("user:1", redisTemplate);
    map.put("name", "Eko");
    map.put("address", "Indonesia");
    assertThat(map, hasEntry("name", "Eko"));
    assertThat(map, hasEntry("address", "Indonesia"));

    Map<Object, Object> user = redisTemplate.opsForHash().entries("user:1");
    assertThat(user, hasEntry("name", "Eko"));
    assertThat(user, hasEntry("address", "Indonesia"));
}
}
```

Repository

Repository

- Saat kita belajar Spring Data JPA, salah satu fitur yang sangat menarik adalah fitur Repository
- Dimana kita tidak perlu melakukan manual lagi manipulasi data menggunakan JPA, cukup menggunakan Repository
- Spring Data Redis juga mendukung fitur Repository
- Data yang disimpan di redis, akan disimpan dalam bentuk tipe data Hash

Enable Redis Repositories

- Untuk mengaktifkan fitur Repository, kita harus menggunakan annotation @EnableRedisRepositories pada Spring Boot Application
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/repository/configuration/EnableRedisRepositories.html

Kode : Enable Redis Repositories

```
@SpringBootApplication
@EnableScheduling
@EnableRedisRepositories
public class BelajarSpringRedisApplication {

    Eko Kurniawan Khannedy
    public static void main(String[] args) {
        SpringApplication.run(BelajarSpringRedisApplication.class, args);
    }

}
```

Entity

- Untuk membuat Entity di Spring Data Redis, kita bisa menggunakan annotation @KeySpace
- https://docs.spring.io/spring-data/keyvalue/docs/current/api/org/springframework/data/keyvalue/annotation/KeySpace.html
- Selain itu, kita juga perlu menentukan attribute yang akan dijadikan sebagai @Id pada Entity tersebut
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/annotation/Id.html

Kode : Product

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@KeySpace("products")
public class Product {

  @Id
  private String id;

  private String name;

  private Long price;
}
```

Key Value Repository

- Untuk membuat Repository, kita bisa membuat interface turunan dari KeyValueRepository
- https://docs.spring.io/spring-data/keyvalue/docs/current/api/org/springframework/data/keyvalue/repository/KeyValueRepository.html

Kode : Product Repository

```
1 usage  new *
@Repository
public interface ProductRepository extends KeyValueRepository<Product, String> {
}
```

Kode : Repository Test

```
@Test
void repository() {
    Product product = Product.builder()
        .id("1")
        .name("Mie Ayam Goreng")
        .price(20_000L)
        .build();
    productRepository.save(product);

    Map<Object, Object> map = redisTemplate.opsForHash().entries("products:1");
    assertEquals(product.getId(), map.get("id"));
    assertEquals(product.getName(), map.get("name"));
    assertEquals(product.getPrice().toString(), map.get("price"));

    Product product2 = productRepository.findById("1").get();
    assertEquals(product, product2);
}
```

Entity Time to Live

Entity Time to Live

- Saat membuat Entity, kadang-kadang kita ingin juga mengatur waktu expired nya
- Kita bisa menggunakan annotation @TimeToLive pada salah satu atribut yang bernilai number
- Secara otomatis Spring Data Redis akan menggunakan nilai di atribut @TimeToLive untuk menentukan berapa lama data harus dihapus di Redis

Kode : Product Entity

```java
@Builder
@KeySpace("products")
public class Product {

  @Id
  private String id;

  private String name;

  private Long price;

  @TimeToLive(unit = TimeUnit.SECONDS)
  private Long ttl = -1L;
}
```

Kode : Time to Live Test

```
@Test
void ttl() throws InterruptedException {
    Product product = Product.builder()
        .id("1")
        .name("Mie Ayam Goreng")
        .price(20_000L)
        .ttl(3L)
        .build();
    productRepository.save(product);

    assertTrue(productRepository.findById("1").isPresent());
    Thread.sleep(Duration.ofSeconds(5));

    assertFalse(productRepository.findById("1").isPresent());
}
```

Monitoring

Monitoring

- Saat kita menggunakan Spring Data Redis di Spring Boot, secara otomatis akan diredistrasikan RedisHealthIndicator
- https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/actuate/data/redis/RedisHealthIndicator.html
- Artinya secara otomatis, saat endpoint health diakses, Spring Boot akan melakukan ping koneksi ke Redis

Kode : Application Properties

application.properties

```
management.endpoints.web.exposure.include=health

management.endpoint.health.enabled=true
management.endpoint.health.show-details=always

management.health.redis.enabled=true

```

Spring Caching

Spring Caching

- Spring memiliki fitur bernama Caching, fitur ini digunakan untuk menyimpan data di memory secara sementara (Cache)
- Fitur Spring Caching, bisa di integrasikan dengan Spring Data Redis, sehingga data Cache bisa disimpan di Redis secara otomatis
- Untuk mengaktifkan fitur Caching, kita harus menggunakan annotation @EnableCaching
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/cache/annotation/EnableCaching.html

Kode : Enable Caching

```
Eko Kurniawan Khannedy
@SpringBootApplication
@EnableScheduling
@EnableRedisRepositories
@EnableCaching
public class BelajarSpringRedisApplication {

    Eko Kurniawan Khannedy
    public static void main(String[] args) {
        SpringApplication.run(BelajarSpringRedisApplication.class, args);
    }

}
```

Cache Manager

- Spring Caching menggunakan class Cache Manager sebagai Cache Management nya
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/cache/CacheManager.html
- Untuk memanipulasi data di Cache, kita bisa menggunakan class Cache
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/cache/Cache.html
- Saat kita menambah Spring Data Redis, secara otomatis akan meregistrasikan RedisCacheManager sebagai implementasi dari Cache Manager
- https://docs.spring.io/spring-data/redis/docs/current/api/org/springframework/data/redis/cache/RedisCacheManager.html

Kode : Application Properties

```
spring.cache.type=redis
spring.cache.redis.use-key-prefix=true
spring.cache.redis.key-prefix=cache:
spring.cache.redis.cache-null-values=true
spring.cache.redis.enable-statistics=true
spring.cache.redis.time-to-live=60s
```

Kode : Caching

```
@Test
void cache() {
    Cache sample = cacheManager.getCache("scores");
    sample.put("Eko", 100);
    sample.put("Budi", 95);

    assertEquals(100, sample.get("Eko", Integer.class));
    assertEquals(95, sample.get("Budi", Integer.class));

    sample.evict("Eko");
    sample.evict("Budi");
    assertNull(sample.get("Eko", Integer.class));
    assertNull(sample.get("Budi", Integer.class));
}
}
```

Declarative Caching

Declarative Caching

- Selain menggunakan Cache Manager, untuk melakukan manajemen data Cache, kita bisa menggunakan cara Declarative, yaitu menggunakan Annotation
- Jika sebuah method ditambahkan annotation untuk Caching, secara otomatis hasil dari method akan disimpan di Cahe
- Ada banyak annotation yang bisa kita gunakan untuk melakukan management Cache

Cacheable

- Annotation @Cacheable Menandakan bahwa return value dari method harus disimpan di cache
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/cache/annotation/Cacheable.html
- Secara default, RedisCacheManager akan menyimpan data dalam bentuk binary (byte[]), oleh karena itu kita perlu memastikan data Object nya implement Serializable agar bisa disimpan sebagai binary data
- https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html

Kode : Product

```java
@Builder
@KeySpace("products")
public class Product implements Serializable {

  @Id
  private String id;

  private String name;

  private Long price;

  @TimeToLive(unit = TimeUnit.SECONDS)
  private Long ttl = -1L;
}
```

Kode : Cacheable

```java
@Slf4j
@Component
public class ProductService {

    no usages   new *
    @Cacheable(value = "products", key = "#id")
    public Product getProduct(String id) {
        log.info("Get product {}", id);
        return Product.builder().id(id).name("Sample").build();
    }

}
```

Kode : Test Cacheable

```
@Test
void findProduct() {
  Product product = productService.getProduct("P-001");
  assertNotNull(product);
  assertEquals("P-001", product.getId());
  assertEquals("Sample", product.getName());

  Product product2 = productService.getProduct("P-001");
  assertEquals(product, product2);
}
```

CachePut

- Annotation CachePut digunakan untuk mengubah data di Cache, tanpa harus mengakses method dengan annotation @Cacheable
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/cache/annotation/CachePut.html

Kode : Cache Put

```
no usages  new *
@CachePut(value = "products", key = "#product.id")
public Product save(Product product) {
  log.info("Save product {}", product);
  return product;
}

}
```

Kode : Test Cache Put

```
  @Test
  void saveProduct() {
    Product product = Product.builder().id("P002").name("Sample").build();
    productService.save(product);

    Product product2 = productService.getProduct("P002");
    assertEquals(product, product2);
  }
}
```

CacheEvict

- Untuk menghapus data di Cache, selain secara otomatis menggunakan TTL, kita bisa menggunakan annotation @CacheEvict
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/cache/annotation/CacheEvict.html

Kode : Cache Evict

```
no usages  new *
@CacheEvict(value = "products", key = "#id")
public void remove(String id) {
  log.info("Remove product {}", id);
}

}
```

Kode : Test Cache Evict

```
new *
@Test
void remove() {
    Product product = productService.getProduct("P003");
    assertNotNull(product);

    productService.remove("P003");

    Product product2 = productService.getProduct("P003");
    assertEquals(product, product2);
}
}
```