Day 14

Dependency

- Dependency is when an object requires other objects to function correctly
- Dependency example - Order
  - `Order` object requires `Item` object to be part of its member
  - A specific order eg `orderItem abc123` will require all Item that is part of that order
- Dependency can be fulfilled
  - Manually by a developer perform the necessary steps
  - Automatically by a framework like the Spring IoC Container

```
public class Order {
    private String orderId;
    private Date orderDate;
    private List<Item> orderItems;
    ...
}
```

```
public class Item {
    private String sku;
    private String description;
    private Integer quantity;
    private Float unitPrice;
    ...
}
```

Inversion of Control

- Transfer control from the object to an external party for some operation
- Traditional flow control
  - Eg. program creates order object, set `orderId` to `abc123`, lookup items for order `abc123`, assign items to order object
- Inversion of control
  - Eg. program request order `abc123`, IoC container creates order, set `abc123` to `orderId`, lookup items for order `abc123`, assign items to order object, passes the populated order object back to the program
- IoC requires the following
  - How an object is created
  - Where is the required

Spring Beans

- Objects managed by Spring IoC
  - Lifecycle of the object is managed by Spring IoC
  - Eg. `@Controller` where the controller class is created and destroyed by Spring
- Need to annotate a class to indicate that it is a bean
  - `@Component` - a ‘regular’ class
  - `@Controller` - controller in Spring Web
  - `@Configuration` - holds configuration (factory methods) for creating objects
    - Requires configuration
    - Factory methods are annotated with @`Bean`
  - `@Service` - holds business logic
  - `@Repository` - for storing and querying data

Example - Dependency Injection

```
@Component
@Scope("singleton")
public class UserCarts {
    private final ReadWriteLock rwLock = new ReentrantReadWriteLock();
    private final Map<String, Cart> carts = new HashMap<>();

    public Cart get(String userId) {
        final Lock l = rwLock.readLock();
        try {
            l.lock();
            if (carts.containsKey(userId))
                return carts.get(userId);
            Cart c = new Cart();
            carts.put(userId, c);
            return c;
        } finally { l.unlock(); }
    }
    ...
}
```

Annotate the class as a bean

Lifecycle of the object
When scope is singleton, only a single instance is created and shared. Need to consider concurrent access

⚠ Require default constructor or a constructor without any parameters

Example - Dependency Injection

```
@Controller
@RequestMapping(path="/cart")
public class CartController {

    @Autowired
    private UserCarts carts;

    @GetMapping("{userId}")
    public String get(Model model,
        @PathVariable String userId) {

        Cart cart = carts.get(userId);
        ...
    }

}
```

Bean annotation to request Spring IoC to create CartController when the HTTP resource is `/cart`

Annotation to indicate to Spring to use IoC to resolve this dependency

Spring try to resolve `UserCarts` object. If none is found, will instantiate a instance
Otherwise will return an existing copy because the scope is singleton

What is Persistence?

- The process of storing data of an application
  - Can be retrieved at a later stage
- Data stored to the persistent layer will survive application reboots and server crashes
- Data is stored in databases, flat files, tapes, etc
- Database is the most common way to store application related data
  - Data can be accessed and manipulated by many different application
  - Impose data integrity rules
  - Control access to the data
  - Queried with SQL or database specific language

Redis Data Store

- In memory key/value data store
  - The database is held in memory
- Holds the following data structure
  - String, hash maps, list, set, sorted set
- Used for caching data, chat messages, game score board, etc.
- Supports publish subscribe
  - Consumers can subscribe to specific channels
  - Producers publishes data to these channels
  - Redis will deliver to data to the subscribers

Redis Data Types

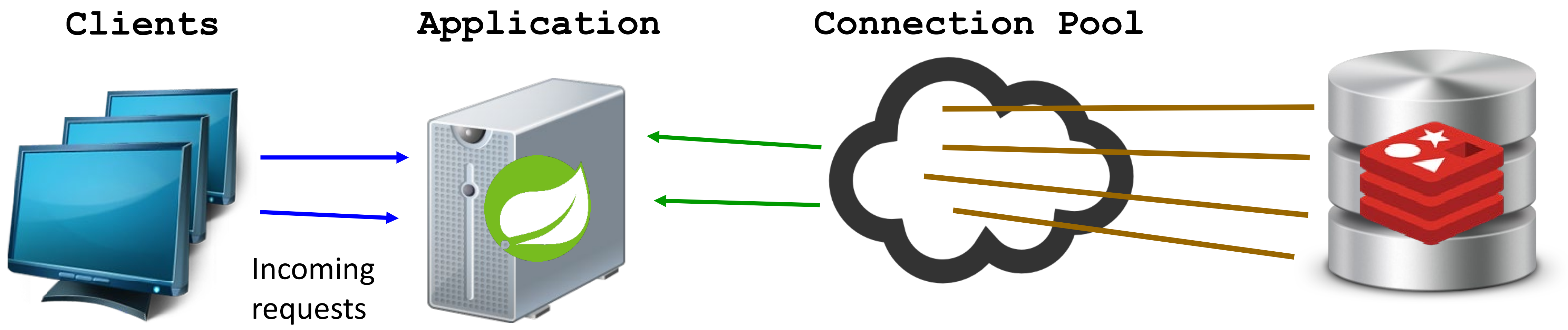
- String - “hello, world”
- Number – 3.14
- Bitmaps – 01101110
- Hashes – { fr: “Fred”, br: “Barney”, wm: “Wilma”, bt: “Betty”, wi: “Wilma” }
- List – [“Fred”, “Barney”, “Wilma”, “Betty”, “Fred”]
- Set – { “Fred”, “Barney”, “Wilma”, “Betty” }
- ZSet – { “Fred”: 0.1, “Berney”: 0.5, “Wilma”: 3, “Betty”: 3 }

Redis Use Cases

String	Session
	Cache
	Distributed Locks
Integer	Counter
	Rate Limiter
	Global ID
Hash	Shopping Cart
Bitmap	User Retention
List	Message Queue
ZSet	Ranking

Graphics recrreated from blogs.bytebytego.com https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fbucketeer-e05bbc84-baa3-437e-9518-adb32be77984.s3.amazonaws.com%2Fpublic%2Fimages%2F0205d074-5f42-402b-b150-99027cb4fc38_800x1114.jpeg?utm_source=substack&utm_medium=email

Using Redis in Web Application



Controller checks out a Redis connection from the pool
A Redis connection is represented by an instance of
`RedisTemplate<String, Object>`

Starting, Stopping and Testing Redis

- Start Redis as a single node

```
redis-server
```

  - Stop the service or CTRL-C
- Test connection to redis

```
redis-cli -h localhost ping
```

  - Will respond with PONG

Redis Commands - Primitives

- Set a key

```
set email fred@gmail.com
set count 0
```

- Read a key

```
get email
```

- Delete a key

```
del email
```

- Check if a key exists

```
exists email
```

- Increment and decrement a key
  - Numbers only

```
incr count
incrby count 5
decr count
decrby count 3
```

Redis Commands - List

- Push an item into a list
  - Left - front, right - back

```
lpush cart apple # apple
rpush cart orange # apple, orange
lpush apple pear # pear, apple, orange
```

- Remove an item from a list

```
lpop cart # apple, orange
rpop cart # apple
```

- Get an item from a list
  - Assume list is pear, apple, orange

```
lindex cart 1 # apple
```

- List length

```
llen cart
```

- Change the value of an element

```
lset cart 1 “fuji apple”
```

- Inserting elements into the list
  - Assume list is pear, apple, orange

```
linsert cart before apple pineapple grapes # pear, pineapple, grapes, apple, orange
```

```
linsert cart after apple banana # pear, pineapple, grapes, apple, banana, orange
```

Redis Commands - Map

- Set a key in a map
```
hset c01 email fred@gmail.com
hset c01 credit 100
```
- Read a key
```
hget c01 email
hgetall c01
```
- Delete a key
```
hdel c01 email
```
- Check if a key exists
```
hexists c01 comments
```
- Get all keys from a map
```
hkeys c01
```
- Get all values from a map
```
hvals c01
```
- Increment the value of a key in a map
  - Numerical values only
```
hincrby c01 credit 100
```
- Map size
```
hlen c01
```

Redis Commands - Key Management

- Return all keys

```
keys *name* # firstname, lastname
keys c?? # c01, cat
keys * # firstname, lastname, c01, cat
```

- Set expiration time, in seconds

```
expire cart 300
```

- Check expiration duration
  - -1 means no expiration time
  - -2 key has expired

```
ttl cart
```

Creating and Configuring a Redis Connection

Holds factory methods to create and configure beans

```
@Configuration
public class AppConfig {
    @Value("${spring.redis.host}")
    private String redisHost;
    @Value("${spring.redis.port}")
    private Optional<Integer> redisPort;
    ...
}
```

Indicates value is optional

Values are injected from `resources/application.properties`

- `spring.redis.host` - String
  - Host to connect to
- `spring.redis.port` - Integer
  - Redis port number, default to 6379
- `spring.redis.database` - Integer
  - Select a database
- `spring.redis.jedis.pool.max-active` - Integer
  - Maximum number of active connection. Set this value to restrict the pool size
- `spring.redis.jedis.pool.max-idle` - Integer
  - Maximum number of idle connection
- `spring.redis.jedis.pool.min-idle` - Integer
  - Minimum number of idle connection

Creating and Configuring a Connection Pool

```
@Bean
@Scope("singleton") // This is the default scope
public RedisTemplate<String, Object> createRedisTemplate()
 {
    final RedisStandaloneConfiguration config = new RedisStandaloneConfiguration();
    config.setDatabase(redisDatabase);
    config.setHostName(redisHost);

    final JedisClientConfiguration jedisClient = JedisClientConfiguration
           .builder().build();
    final JedisConnectionFactory jedisFac = new JedisConnectionFactory(
           config, jedisClient);
    jedisFac.afterPropertiesSet();

    final RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(jedisFac);
    template.setKeySerializer(new StringRedisSerializer());
    template.setValueSerializer(new StringRedisSerializer());
    return template;
}
```

Config values are injected from property file

Configure the database

Create the client and factory

Create the template with the client

Keys are in UTF-8

Optional value serializer if string values are to be saves as UTF-8

Note: Jedis Dependency for Spring Boot

- If you are using Spring Boot 3.x, then you can use Jedis client version 4.x
- If you are using Spring Boot 2.7.x then use the latest 3.x Jedis client
  - Spring Boot 2.x does not work with Jedis client version 4.x

```
<dependency>
   <groupId>redis.clients</groupId>
   <artifactId>jedis</artifactId>
   <version>3.9.0</version>
</dependency>
```

Example - Cart Service

```
@Repository
public class CartRepository {
    @Autowired
    private RedisTemplate<String, Object> template;

    public void add(String user, Cart cart) {
        template.opsForValue().set(user, cart, Duration.ofMinutes(10));
    }

    public Optional<Cart> getCart(String user) {
        return Optional.ofNullable(template.opsForValue().get(user));
    }
}
```

Inject an instance of `RedisTemplate` into this service object
`RedisTemplate` is created by `@Bean` method

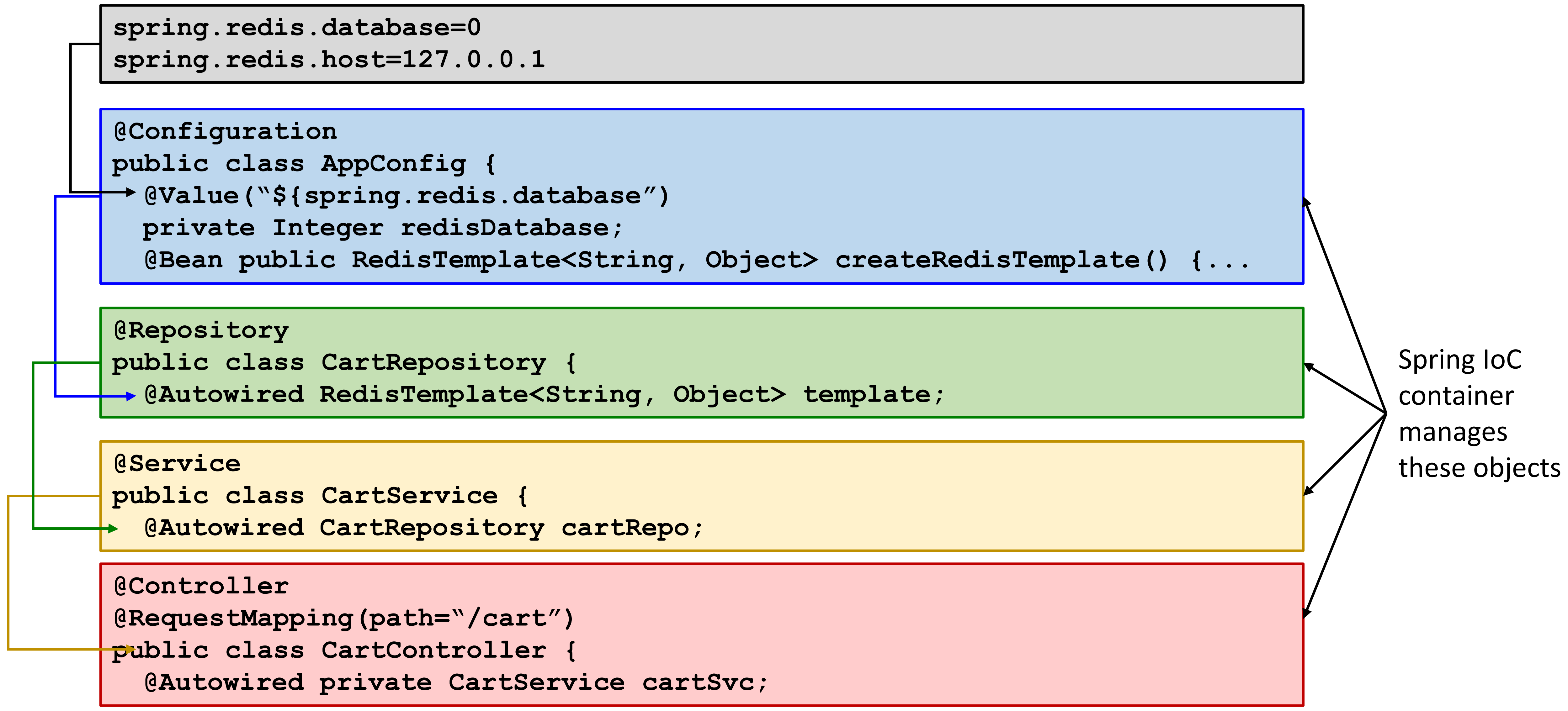
Store cart as a single valued object.
Objects must be serializable

Example - Cart Controller

```
@Controller
@RequestMapping(path="/cart")
public class CartController {
   @Autowired
   private CartRepository cartRepo;

   @GetMapping("{user}")
   public String get(Model model, @PathVariable() String user) {
      Optional<Cart> opt = cartRepo.get(user);
      Cart cart = opt.isEmpty()? new Cart(): opt.get();
      // Do something with cart
      ...
   }
}
```

Example - Dependency Injection Illustrated



```
spring.redis.database=0
spring.redis.host=127.0.0.1
```

```
@Configuration
public class AppConfig {
  @Value(“${spring.redis.database”)
  private Integer redisDatabase;
  @Bean public RedisTemplate<String, Object> createRedisTemplate() {...
```

```
@Repository
public class CartRepository {
  @Autowired RedisTemplate<String, Object> template;
```

```
@Service
public class CartService {
  @Autowired CartRepository cartRepo;
```

```
@Controller
@RequestMapping(path=“/cart”)
public class CartController {
  @Autowired private CartService cartSvc;
```

Value Operations

- `redisTemplate.opsForValue()` to access value operations
- Adding and removing keys
  - `set(keyName, value),get(keyName)`
  - `set(keyName, value, duration)`
  - `setIfAbsent(keyName, value),setIfPresent(keyName, value)`
  - Value must be serializable
- Working with numbers
  - `decrement(keyName),increment(keyName)`
  - If the value is a number

Set a Value

```
set email “Fred Flintstone”
set age 50
```

```
template.opsForValue().set(“name”, “Fred Flintstone”);
template.opsForValue().set(“age”, 50);
```

Get a Value

```
get name
```

```
Optional<String> opt = Optional.ofNullable(
        template.opsForValue().get(“name”));
if (opt.isPresent())
    String name = opt.get();
```

Increment/Decrement a Key

```
incr count
incrby count 10
decr count
decrby count 3
```

```
template.opsForValue().increment(“count”);
template.opsForValue().increment(“count”, 10);
template.opsForValue().decrement(“count”);
template.opsForValue().decrement(“count”, 3);
```

Delete a Key

```
del email
```

```
template.delete(“email”);
```

Check if a Key Exists

```
exists email
```

```
boolean hasEmail = template.hasKey("email");
```

List Operations

- `redisTemplate.opsForList()` to access value operations
- Adding and removing items from list
  - `leftPush`(keyName, value), rightPush(keyName, value)
  - `leftPop(keyName)`, rightPop(keyName), leftPop(keyName, duration)
  - `set(keyName, index, value),indexOf(keyName, value)`
  - `remove(keyName, count, value)`
- List size
  - `size(keyName)`
- Sublist
  - `range(keyName, startIndex, endIndex)`

Push a Value into a List

```
lpush cart apple
rpush cart orange
```

```
template.opsForList().leftPush(“cart”, “apple”);
template.opsForList().rightPush(“cart”, “orange”);
```

Pop a Value into a List

```
lpop cart
rpop cart
```

```
template.opsForList().leftPop(“cart”);
template.opsForList().rightPop(“cart”);
```

Get Value at Index Position

```
lindex cart 1
```

```
String item = template.opsForList().index(“cart”, 1L);
```

List Size

```
llen cart
```

```
long cartLen = template.opsForList().size(“cart”);
```

Get Value at Index Position

```
linsert cart before apple pineapple
linsert cart after apple banana
```

```
template.opsForList().leftPush(“cart”, “apple”, “pineapple”);
template.opsForList().rightPush(“cart”, “apple”, “banana”);
```

Map Operations

- `redisTemplate.opsForHash()` to access value operations
- Add and remove entries
  - `put(keyName, mapKey, value)`,`delete(keyName, mapKey)`
- Check if a give mapKey is present
  - `hasKey(keyName, mapKey)`
- Get all keys or values
  - `keys(keyName)`,`values(keyName)` - returns `Set` and `List` respectively
- Map size
  - `size(keyName)`

Set Value for a Map

```
hset c0 email fred@gmail.com
hset c0 credit 100
```

```
template.opsForHash().put(“c0”, “email”, “fred@gmail.com”);
template.opsForHash().put(“c0”, “credit”, 100);
```

Read Values from Map

```
hget c0 email
hget c0 credit
```

```
template.opsForHash().get(“c0”, “email”);
template.opsForHash().get(“c0”, “credit”);
```

Delete a Key from a Map

```
hdel c0 email
```

```
template.opsForHash().delete(“c0”, “email”);
```

Check if a Key Exists

```
hexists c0 email
```

```
template.opsForHash().hasKey(“c0”, “email”);
```

Get All Keys and Values from Map

```
hkeys c01
hvals c01
```

```
Set<String> keys = template.opsForHash().keys(“c01”);
List<Object> values = template.opsForHash().values(“c01”)l
```

Map Size

```
hlen c01
```

```
long mapSize = template.opsForHash().size("c01");
```

Increment the Value of a Key

```
hincrby c01 count 1
```

```
template.opsForHash().increment(“c0”, “count”, 1);
```

Appendix

Creating and Configuring a Connection Pool

```
@Bean
@Scope("singleton") // This is the default scope
public RedisTemplate<String, Object> createRedisTemplate() {
    final RedisStandaloneConfiguration config = new RedisStandaloneConfiguration();
    config.setDatabase(redisDatabase);
    config.setHostName(redisHost);
    final GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
    poolConfig.setMaxTotal(redisMaxActive);
    poolConfig.setMinIdle(redisMinIdle);
    poolConfig.setMaxIdle(redisMaxIdle);
    final JedisClientConfiguration jedisClient = JedisClientConfiguration
            .builder()
            .usePooling().poolConfig(poolConfig).build();

    final RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(new JedisConnectionFactory(config, jedisClient));
    template.setKeySerializer(new StringRedisSerializer());
    return template;
}
```

Config values are injected from property file

Configure the database

Configure the pool

Create the client

Create the template with the client