

Java Backend Development

Live-85

lecture-17

Agenda

- Code Coverage
- Redis Intro
 - Redis-server And Redis-cli
 - Basic Command
- Cloud Redis setup.
- Connect java application to Redis-Server
 - Spring Data Redis
 - JedisConnectionFactory and RedisTemplate
 - REST APIs with Redis
- Spring Security Intro
 - Authentication vs. Authorization
 - Cookie Based Authentication
 - User Store

Code Coverage














In software development, Code Coverage is a measure used to describe the degree to which the source code of an application is executed when a test suite is executed. A report is generated to view and analyze the code coverage of a software application. This code coverage report could then be used for ensuring code quality.

For optimal code testing, many companies use the **JaCoCo-Maven** plugin that helps generate detailed code coverage reports. JaCoCo-Maven plugin is a free code coverage library for Java projects.

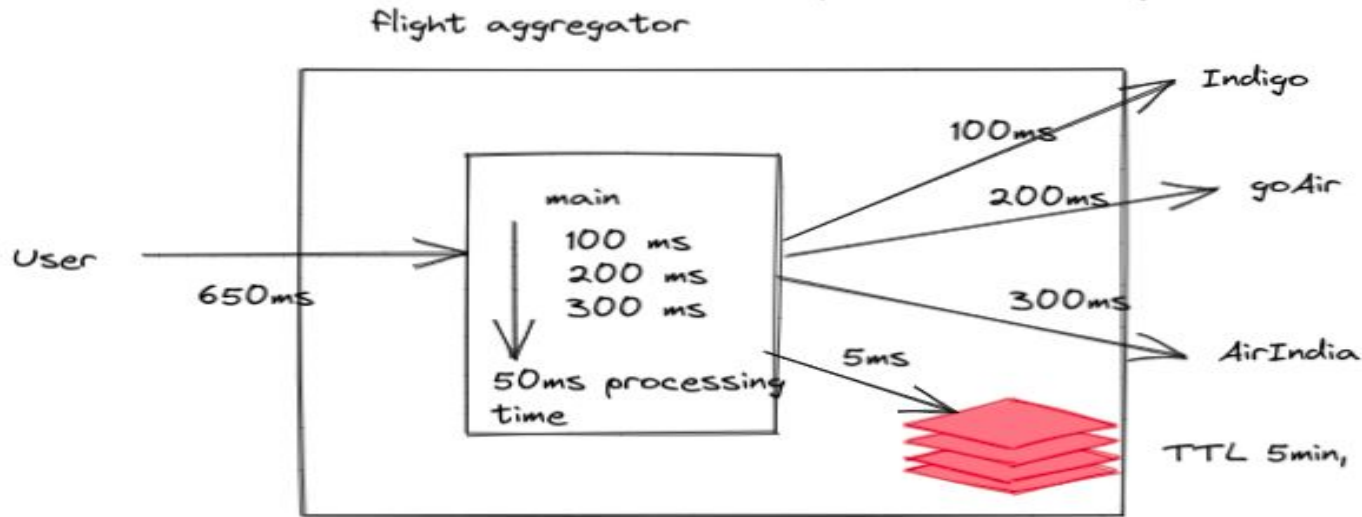
JaCoCo Code Coverage

 L16-UTandITdemo

L16-UTandITdemo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 com.example.L16_UTandITdemo.service		50%		50%	6	10	20	43	4	8	0	2
 com.example.L16_UTandITdemo.entity		21%		8%	12	18	14	20	6	12	0	1
 com.example.L16_UTandITdemo.controller		32%		n/a	4	7	7	11	4	7	0	2
 com.example.L16_UTandITdemo.exception		0%		n/a	2	2	4	4	2	2	2	2
 com.example.L16_UTandITdemo		86%		100%	1	6	2	9	1	5	0	2
Total	195 of 341	42%	13 of 18	27%	25	43	47	87	17	34	2	9

Optimizing Flight Aggregator with Cache



Using 3 threads:
In 300ms all data will available.
API Response time(With multithreading): 350ms

With Redis: 55ms (Cache hit)
360ms (Cache Miss)

Redis Architecture

Redis, which stands for "Remote Dictionary Server," is a popular data store extensively used by companies for caching. In addition to caching, it also supports replication, pub-sub messaging, sharding, geospatial indexing, etc.

Redis stores the data in the RAM. Accessing data stored on the disk requires an I/O call. I/O calls are time consuming and increases the latency. Redis is able to bypass the I/O call and serve the data directly from the memory.

Redis is single threaded, while one command is executing, no other commands will run. Redis uses epoll and event-loop to implement a concurrency strategy.

Redis Server and Redis-Cli

Run Redis Server: `redis-server [config file path]`

Default port: 6379

Use `redis-cli` to connect redis server.

`redis-cli -h 127.0.0.1 -p 6379`

Important Redis Commands

Set keyname value

Get keyname

Ttl keyname

Keys "prefix*"

Expire keyname 10

Incr keyname

Decr keyname

Redis-cli --help

Redis Data Structures

Strings: Text data.

Lists: A collection of Strings. Adding order will be preserved.

Sets: An unordered collection of strings with the ability to do set operations

Sorted Sets: Sets ordered by a value associate with a key. Can be used for leader board and scoreboard implementations.

Hashes: A data structure for storing a list of fields and values similar to Hashmaps.

Bitmaps: A data type that offers bit-level operations to be done.

Ref: <https://redis.io/docs/latest/commands/>

LIST

Lpush jbdI_batch “rahul” “ravi”

Lrange jbdI_batch <start index> <endIndex>

Rpush jbdI_batch “gaurav”

Lpop

Rpop

We can implement Queue and Stack using LIST.

SET

SADD userSet "rahul01" "ravi01" "vikas02"

SMEMBERS userSet

SISMEMBER userSet "ravi01"

SREM userSet "ravi01"

SDIFF set1 set2

SINTER set1 set2

SUNION set1 set2

Hashes

Hset product:1 name laptop price 40000 category gaming

Hgetall product:1

Hget product:1 name

Hmget product:1 name price

Bitmaps

SETBIT user:1001:resume 0 1

SETBIT user:1001:resume 1 0

SETBIT job:15:requirements 0 1

SETBIT job:15:requirements 1 1

GETBIT user:1001:resume 0

BITOP AND user:1001:eligibility user:1001:resume job:15:requirements

BITCOUNT user:1001:eligibility

Cloud Redis Setup

Setup free redis instance on app.redislabs.com

Storage Limit: 30MB

Access:

```
redis-cli -h <public_endpoint> -p <port> -a <password>
```

Spring Data Redis

Spring Data Redis, part of the larger Spring Data family, provides easy configuration and **access to Redis** from Spring applications.

It offers both low-level and high-level **abstractions** for interacting with the store, freeing the user from infrastructural concerns.

It provides connection package as low-level abstraction across multiple Redis drivers(**Lettuce** and **Jedis**).

RedisConnectionFactory

RedisConnectionFactory manages connections to Redis-Server.

@Bean

```
RedisConnectionFactory redisConnectionFactory() {  
    LettuceConnectionFactory redisConFactory  
        = new LettuceConnectionFactory();  
    redisConFactory.setHostName("localhost");  
    redisConFactory.setPort(6379);  
    return redisConFactory;  
}
```

```
#Redis Properties  
spring.data.redis.host=localhost  
spring.data.redis.port=6379  
spring.data.redis.password=
```


RedisTemplate

RedisTemplate provides a high-level abstraction for performing various **Redis operations**, exception translation and serialization support.

@Bean

```
public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
redisConnectionFactory) {

    RedisTemplate<String, Object> template = new RedisTemplate<>();

    template.setConnectionFactory(redisConnectionFactory());

    return template;
}
```

Redis Serializers

Redis Serializers are user to serialize and deserialize the objects.

We need to serialize the object to bytes before writing it to redis and deserialize the bytes read from redis to create the object.

- ValueSerializer
- KeySerializer
- HashKeySerializer
- HashValueSerializer

Spring Security

Spring Security is a powerful and highly customizable **authentication and access-control framework**. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both **authentication** and **authorization** to Java applications.

Authentication vs. Authorization

Authentication is the process of verifying who someone is, whereas **authorization** is the process of verifying what specific applications, files, APIs, and data a user has access to.

Authentication fail: 401

Authorization fail: 403

Cookie Based Authentication

Step 1: Client > Signing up

The client posts a HTTP request to the server containing his/her username and password.

Step 2: Server > Handling sign up

The server receives this request and hashes the password before storing the username and password in your database.

Step 3: Client > User login

User provides their username/password and again, this is posted as a HTTP request to the server.

Step 4: Server > Validating login

The server looks up the username in the database, hashes the supplied login password, and compares it to the previously hashed password in the database.

Step 5: Server > Generating access token

If everything checks out, we're going to create an access token, which uniquely identifies the user's session.

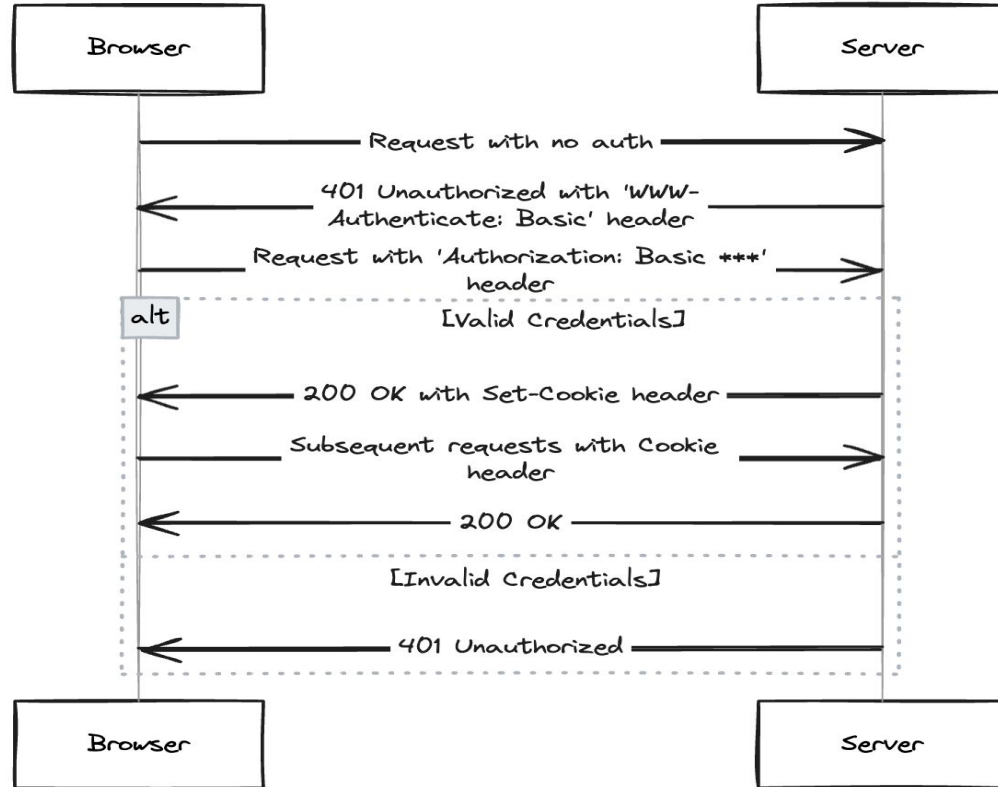
- Store it in the database associated with that user
- Attach it to a response cookie to be returned to the client.

Step 6: Client > Making page requests

Every time the client makes a request for a page it will send the cookie in all requests.

The server obtains the access token from the cookie and checks it against the one in the database associated with that user.

Basic Auth Request Flow



Spring-Security Configs

By default all APIs will be secure.

We can define specific beans (component) like PasswordEncoder, UserDetailsService, SecurityFilterChain etc which will get executed in the request flow.

We can provide **SecurityFilterChain** bean to configure HttpSecurity and define API access configs based on API **url pattern** and User's **role/authority**

Ref: <https://spring.io/blog/2022/02/21/spring-security-without-the-websecurityconfigureradapter>

In Memory Users

You can define single user in application properties.

```
spring.security.user.name=amit  
spring.security.user.password=amit123  
spring.security.user.roles=ADMIN
```

For more than one user you can define bean of UserDetailsService with **InMemoryUserDetailsManager**.

```
@Bean  
public UserDetailsService userDetailsService(){  
    UserDetails user = User.builder()  
        .username("rahu1").password("rahu1@123")  
        .roles("USER").build();  
    return new InMemoryUserDetailsManager(user);  
}
```

PasswordEncoder

We should never save password as plain text.

PasswordEncoder bean is required to encode the passwords.eg.
BCryptPasswordEncoder, NoOpPasswordEncoder

This encoding is one way.

```
@Bean
public PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
}
```

Read from SecurityContext

SecurityContextHolder is used to fetch details of logged in user.

SecurityContextHolder.getContext().getAuthentication().getPrincipal() returns instance of UserDetails class.

In controller we can use @AuthenticationPrincipal to get UserDetails of authenticated user.

UserDetailsService

Spring Security application can interact with any User store via UserDetailsService interface.

We can provide implementation of UserDetailsService based on our user-store.

User-store can be in-memory(JVM), Redis, MySQL, any other Application exposed via APIs.

UserDetailsService has only one method definition. UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;