Spring Boot Redis

#1. REDIS AS DATABASE

Dependencies : Redis, Lombok

https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100

1) Redis Config File

```
package in.nareshit.raghu.config;

@Configuration
public class RedisConfig {

	@Bean
	public RedisConnectionFactory cf() {
		return new LettuceConnectionFactory();
	}
	@Bean
	public RedisTemplate<String, Person> rt(){
		RedisTemplate<String, Person> template = new RedisTemplate<>();
		template.setConnectionFactory(cf());
		return template;
	}
}
```

2) application.properties

```
spring.redis.host=localhost
spring.redis.port=6379
```

3. Model class

```
package in.nareshit.raghu.model;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Person implements Serializable{

        private static final long serialVersionUID = 1L;

        private int id;
        private String name;
        private int age;
}
```

4. IDao

```
package in.nareshit.raghu.dao;
public interface IPersonDao {

        public void addPerson(Person p);
        public void modifyPerson(Person p);
        public Person getPerson(Integer id);
        public Map<Integer,Person> getAllPersons();
        public void removePerson(Integer id);

}
```

5. DaoImpl

```
package in.nareshit.raghu.dao.impl;
@Repository
public class PersonDaoImpl implements IPersonDao {
	private final String KEY ="PERSON";

	@Resource(name = "rt")
	private HashOperations<String, Integer, Person> opr;

	@Override
	public void addPerson(Person p) {
		opr.putIfAbsent(KEY, p.getId(), p);
	}

	@Override
	public void modifyPerson(Person p) {
		opr.put(KEY, p.getId(), p);
	}

	@Override
	public Person getPerson(Integer id) {
		return opr.get(KEY, id);
	}

	@Override
	public Map<Integer, Person> getAllPersons() {
		return opr.entries(KEY);
	}

	@Override
	public void removePerson(Integer id) {
		opr.delete(KEY, id);
	}

}
```

6. Runner

```
package in.nareshit.raghu.runner;
@Component
public class RedisOprTestRunner implements CommandLineRunner{
	@Autowired
	private IPersonDao dao;

	@Override
	public void run(String... args) throws Exception {
		dao.addPerson(new Person(10,"A",52));
		dao.addPerson(new Person(11,"B",48));
		dao.modifyPerson(new Person(12,"C",91));

		dao.removePerson(11);
		dao.getAllPersons().forEach((k,v)->System.out.println(k+"-"+v));
		System.out.println(dao.getPerson(10));
	}
}
```

#2. Redis as Cache using Spring Boot Rest

Dependencies : Spring Web, Redis, Data Jpa, MySQL, Lombok, DevTools, actuator

1. application.yml

```yaml
spring:
  cache:
    type: redis
    redis:
      time-to-live: 60000
      cache-null-values: true

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/nit
    username: root
    password: root
  jpa:
    database-platform: org.hibernate.dialect.MySQL5Dialect
    generate-ddl: true
    show-sql: true
    hibernate:
      ddl-auto: create

management:
  endpoints:
    web:
      exposure:
        include: "*"
```

2. Model class

```
package in.nareshit.raghu.model;
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Employee implements Serializable {

        private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue
   private Integer id;

   private String empName;
   private Double empSal;

}
```

3. Repository

```
package in.nareshit.raghu.repository;
@Repository
public interface EmployeeRepository
extends JpaRepository<Employee,Integer> {
}
```

4. Service Interface

```
package in.nareshit.raghu.service;
public interface IEmployeeService {
	public Employee createEmployee(Employee employee);
	public Employee updateEmployee(Integer empId,Employee employee);
	public void deleteEmployee(Integer empId);

	public Employee getOneEmployee(Integer empId);
	public List<Employee> getAllEmployees();
}
```

5. ServiceImpl

```
package in.nareshit.raghu.service.impl;

@Service
public class EmployeeServiceImpl implements IEmployeeService {

	@Autowired
	private EmployeeRepository repo;

	@Override
	public Employee createEmployee(Employee employee) {
		return repo.save(employee);
	}

	@Transactional
	@CachePut(value = "employees",key = "#empId")
	public Employee updateEmployee(Integer empId, Employee employee) {
		Employee emp = repo.findById(empId)
					.orElseThrow(() ->
		new ResouceNotFoundException("Employee Not Found " + empId));
		emp.setEmpName(employee.getEmpName());
		emp.setEmpSal(employee.getEmpSal());
		return emp;
	}
	@CacheEvict(value = "employees", allEntries = true)
	@Transactional
	public void deleteEmployee(Integer empId) {
		Employee employee = repo.findById(empId).orElseThrow(
				() -> new ResouceNotFoundException("Employee not 
found" + empId));
		repo.delete(employee);
	}

	@Transactional(readOnly = true)
	@Cacheable(value = "employees",key = "#empId")
	public Employee getOneEmployee(Integer empId) {
		return repo.findById(empId)
				.orElseThrow(() -> new 
ResouceNotFoundException("Employee not found" + empId));
	}

	@Override
	@Transactional(readOnly = true)
	public List<Employee> getAllEmployees() {
		return repo.findAll();
	}

}
```

6. Custom Exception

```
package in.nareshit.raghu;
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResouceNotFoundException extends RuntimeException {
	private static final long serialVersionUID = 8722013793153647430L;

	public ResouceNotFoundException(String message) {
        super(message);
    }
}
```

7. Controller

```
package in.nareshit.raghu.controller;

@RestController
@RequestMapping("/employees")
public class EmployeeController {

        @Autowired
        private IEmployeeService service;

        @PostMapping("/save")
        public Employee createEmployee(
                        @RequestBody Employee employee)
        {
                return service.createEmployee(employee);
        }

        @GetMapping("/all")
        public ResponseEntity<List<Employee>> getAllEmployees() {
                return ResponseEntity.ok(service.getAllEmployees());
        }

        @GetMapping("/one/{empId}")
        public Employee getOneEmployee(
                        @PathVariable(value = "empId") Integer empId)
        {

                return service.getOneEmployee(empId);
```

```
	}


	@PutMapping("/modify/{empId}")
	public Employee updateEmployee(
			@PathVariable(value = "empId") Integer empId,
			@RequestBody Employee employee) {

		Employee updatedEmployee = 
service.updateEmployee(empId,employee);
		return updatedEmployee;

	}

	@DeleteMapping("/remove/{empId}")
	public void deleteEmployee(
			@PathVariable(value = "empId") Integer empId)
	{
		service.deleteEmployee(empId);
	}
}
```

8. Starter

```
package in.nareshit.raghu;
@SpringBootApplication
@EnableCaching
public class SpringbootRedisCacheApplication {
 public static void main(String[] args) {
  SpringApplication.run(SpringbootRedisCacheApplication.class, args);
 }

}
```