Spring Redis Connection Auto Config

Spring Boot Redis Properties

이 문서는 Spring Boot에서 application.properties or yml을 이용해서 Redis Server에 연결하는 방법을 설명합니다.
레디스 접속 기본 클라이언트 라이브러리로 Spring Boot 1.x에서는 Jedis를 사용하였고, 2.x부터는 Lettuce를 사용합니다.
이 문서에서는 Lettuce를 기준으로 설명합니다.

Spring Redis Properties 관련 공식 문서는 아래 링크를 참고하세요.

- **Spring Boot 2.7.14** RedisProperties (docs.spring.io)
  spring.redis @ConfigurationProperties(prefix="spring.redis")
- **Spring Boot 3.1.3** RedisProperties (docs.spring.io)
  spring.data.redis @ConfigurationProperties(prefix="spring.data.redis")
- **Spring Boot 최신(current)** RedisProperties (docs.spring.io)
- **Spring Boot** RedisProperties.java (github.com java soruce)
- **Spring Boot** RedisAutoConfiguration.java (github.com java soruce)
- Spring Data Redis (3.1.3 docs.spring.io 설명)

Spring Boot 3.x

application.properties 설정

- Spring Boot 2.x에서는 spring.redis로 시작
- Spring Boot 3.x에서는 spring.data.redis로 시작

이 문서는 Spring Boot 3.x를 기준으로 작성합니다.

Auto Configuration 설정 방법

Auto Configuration은 application.properties에 설정만 해 놓으면 설정한 값으로 Redis Server에 연결은 Spring이 알아서 해줍니다.
Redis Auto Configuration에는 세 가지가 있다.

- Standalone
- Sentinel
- Cluster

Standalone 설정

redis.host: 레디스 서버의 IP를 입력합니다. (필수)
redis.port: 레디스 서버의 port를 입력합니다. (필수)

password: redis server에 password(requirepass) 설정했을 경우 사용합니다. (선택)

application.properties

```
spring.data.redis.host: 192.168.56.102
spring.data.redis.port: 18500
spring.data.redis.password: redisgate
```

application.yml

```
spring:
  data:
    redis:
      host: 192.168.56.102
      port: 18500
      password: redisgate
```

Sentinel 설정

sentinel.nodes: 센티널 서버의 IP:Port를 입력합니다. (필수) 센티널 서버 3대 정보를 모두 입력합니다.
sentinel.master: sentinel.conf에 "sentinel monitor mymaster 192.168.56.102 18520 2" 이와 같이 입력했을 경우 'mymaster'를 입력합니다. (필수)
Lettuce는 위 정보로 센티널 서버에 접속해서 'sentinel master mymaster' 명령을 실행해서 레디스 마스터 서버 정보를 얻어오고, 'sentinel replicas mymaster' 명령을 실행해서 복제 서버 정보를 얻어옵니다. 그러므로 레디스 서버의 IP:Port는 입력할 필요없습니다.
sentinel.password: 센티널 서버에 password(requirepass) 설정했을 경우 사용합니다. (선택)
redis.password: 레디스 서버에 password(requirepass) 설정했을 경우 사용합니다. (선택)

application.properties

```
spring.data.redis.sentinel.nodes: 192.168.56.102:18525, 192.168.56.102:18526, 192.168.56.102:18527
spring.data.redis.sentinel.master: mymaster
spring.data.redis.sentinel.password: redisgate
spring.redis.password: redisgate
```

application.yml

```
spring:
  data:
    redis:
      sentinel:
        nodes: 192.168.56.102:18525, 192.168.56.102:18526, 192.168.56.102:18527
        master: mymaster
        password: redisgate
      password: redisgate
```

Cluster 설정

cluster.nodes: 클러스터로 설정한 레디스 서버의 IP:Port를 입력합니다. (필수) 6대 정보를 모두 입력합니다.
redis.password: 레디스 서버에 password(requirepass) 설정했을 경우 사용합니다. (선택)

application.properties

```
spring.data.redis.cluster.nodes: 192.168.56.102:18530, 192.168.56.102:18531,
 192.168.56.102:18532, 192.168.56.102:18533, 192.168.56.102:18534,
 192.168.56.102:18535
spring.data.redis.password: redisgate
```

application.yml

```
spring:
  data:
    redis:
      cluster:
        nodes: 192.168.56.102:18530, 192.168.56.102:18531, 192.168.56.102:18532,
               192.168.56.102:18533, 192.168.56.102:18534, 192.168.56.102:18535
      password: redisgate
```

프로젝트 생성

start.spring.io

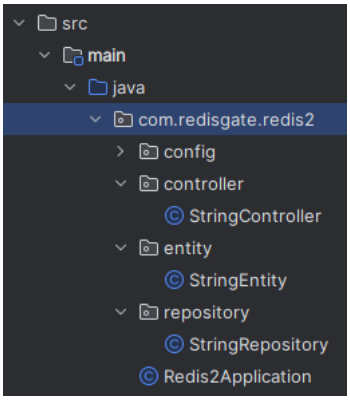
- Project: Gradle
- Language: Java
- Spring Boot 3.1.3
- Group: com.redisgate
- Artifact: redis2
- Packaging: Jar
- Java: 17
- Dependencies
  - Spring Web
  - Lombok
  - Spring Data Redis(Access+Driver)

Spring Boot 3.x -> Java 17 이상
Spring Boot 2.x -> Java 11 이하

개발 IDE: IntelliJ

Java 소스

Package 구성



Java 파일(클래스) 설명

- Redis2Application.java: main program
- StringEntity.java: Redis String data type용 entity
- StringRepository.java: 저장(save), 조회(get)한다.
- StringController.java: url 요청을 받아 실행한다.

Redis2Application.java

```
package com.redisgate.redis2;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@Slf4j
public class Redis2Application {

	public static void main(String[] args) {
		SpringApplication.run(Redis2Application.class, args);
		log.info("Spring Boot 3.x application.properties Test");
	}
}
```

StringEntity.java

```
package com.redisgate.redis2.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class StringEntity {
   private String key;
   private String value;
}
```

StringRepository.java

```
package com.redisgate.redis2.repository;

import com.redisgate.redis2.entity.StringEntity;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.stereotype.Repository;

@Repository
@Slf4j
public class StringRepository {
    private StringRedisTemplate redisCmds;
    private ValueOperations redisString;

    public StringRepository(StringRedisTemplate stringRedisTemplate) {
        this.redisCmds = stringRedisTemplate;
        this.redisString = stringRedisTemplate.opsForValue();
    }

    public void save(StringEntity stringEntity) {
        redisString.set(stringEntity.getKey(), stringEntity.getValue());
    }

    public StringEntity get(String key) {
        String value = (String) redisString.get(key);
        if (value == null) {
            log.info("value is null");
            return null;
        }
        return new StringEntity(key, value);
    }
}
```

StringController.java

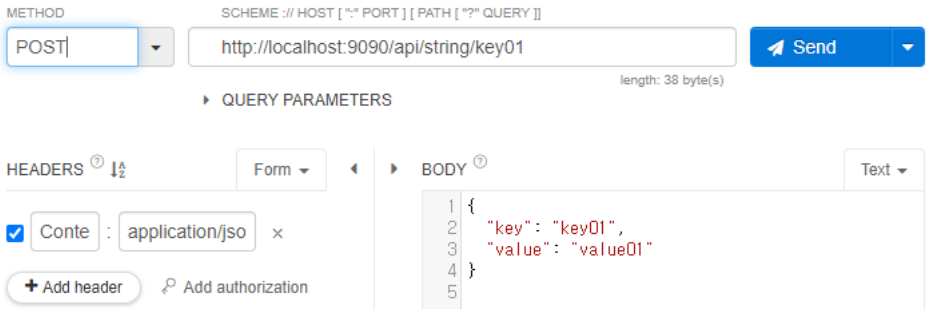
```
package com.redisgate.redis2.controller;

import com.redisgate.redis2.entity.StringEntity;
import com.redisgate.redis2.repository.StringRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/")
public class StringController {

    @Autowired
    StringRepository stringRepository;

    @PostMapping("string")
    public StringEntity saveString(@RequestBody StringEntity stringEntity){
        stringRepository.save(stringEntity);
        return stringEntity;
    }

    @GetMapping("string/{key}")
    public StringEntity get(@PathVariable("key") String key) {
        return stringRepository.get(key);
    }
}
```

프로그램 테스트

프로그램 테스트는 Talend api Tester 확장 프로그램을 사용했습니다.

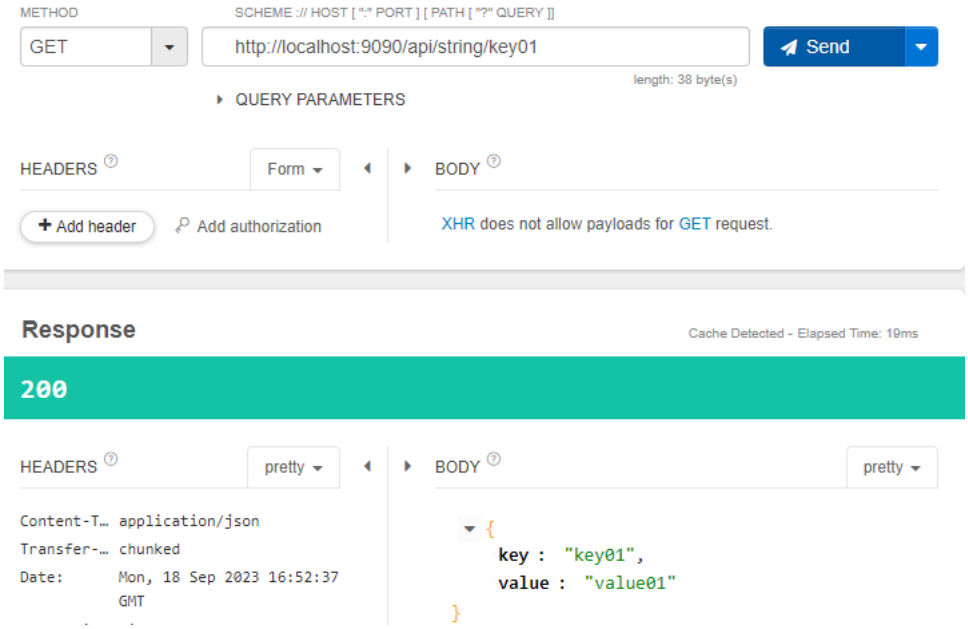
입력

METHOD: POST
URL: http://localhost:9090/api/string
BODY(입력 내용): { "key": "key01", "value": "value01" }



조회

METHOD: GET
URL: http://localhost:9090/api/string/key01
BODY(조회 결과): { "key": "key01", "value": "value01" }



Manual Configuration 설정 방법

여기서는 수동(manual)으로 application.properties를 적용하는 방법을 설명합니다.

LettuceConnectionFactory Class

LettuceConnectionFactory는 기본적으로 redis configuration과 client configuration으로 구성된다.
Redis configuration은 host(ip), port, password, database 등 서버 구성 관련 속성들이 있다.
Client configuration은 clientName, readFrom 등 클라이언트 관련 속성들이 있다.

LettuceConnectionFactory 클래스에는 다음과 같은 생성자가 있다.

- public LettuceConnectionFactory(RedisConfiguration **redisConfiguration**, LettuceClientConfiguration **clientConfig**)
- public LettuceConnectionFactory(RedisStandaloneConfiguration **standaloneConfig**, LettuceClientConfiguration **clientConfig**)
- public LettuceConnectionFactory(RedisSentinelConfiguration **sentinelConfiguration**, LettuceClientConfiguration **clientConfig**)

- public LettuceConnectionFactory(RedisClusterConfiguration **clusterConfiguration**, LettuceClientConfiguration **clientConfig**)

레디스 서버(마스터) 단독 구성일 때는 standaloneConfig를 사용하고,
센티널 구성일 때는 sentinelConfiguration을 사용하고,
클러스터 구성일 때는 clusterConfiguration을 사용한다.
그러면 마스터-복제(replica) 구성일 때는 위 4가지 생성자에는 없지만 RedisStaticMasterReplicaConfiguration을 사용할 수 있다. 이유는 RedisStaticMasterReplicaConfiguration이 redisConfiguration을 implements하므로 가능하다.

테스트 프로그램은 위에서 설명한 StringController.java, StringEntity.java, StringRepository.java를 이용합니다.

수동(manual) 설정에서는 spring.data를 빼고 redis부터 시작하는 것으로 했습니다.

Standalone 설정

항목 설명

redis.database: 데이터베이스를 선택할 수 있습니다. 레디스는 기본으로 0~15까지 16개의 DB를 제공합니다.
redis.clientName: client name을 정할 수 있습니다. client name은 redis-cli에서 client list 명령으로 클라이언트들을 확인할 때 name 항목에 표시되므로 보다 쉽게 클라이언트를 확인할 수 있습니다.

Properties, YML

application.properties

```
redis.host: 192.168.56.102
redis.port: 18500
redis.password: redisgate
redis.database: 0
redis.clientName: redisgate
```

application.yml

```
redis:
  host: 192.168.56.102
  port: 18500
  password: redisgate
  database: 0
  clientName: redisgate
```

Java 소스

RedisInfo1.java: properties, yml에서 항목을 읽어오는 클래스

```
package com.redisgate.redis2.config;

import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ConfigurationProperties(prefix="redis")
@Configuration
public class RedisInfo1 {
    private String host;
    private int port;
    private String password;
    private int database;
    private String clientName;
}
```

RedisConfig1.java: 연결(connection)을 구성하는 클래스

```
package com.redisgate.redis2.config;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

@Configuration
@RequiredArgsConstructor
@Slf4j
public class RedisConfig1 {
    private final RedisInfo1 redisInfo1;

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {

        // Redis Config 설정
        log.info("Step 1 - Standalone Config");
        log.info("host: {}:{}", redisInfo1.getHost(), redisInfo1.getPort());
        log.info("Database: {}", redisInfo1.getDatabase());
        RedisStandaloneConfiguration redisConfig = new RedisStandaloneConfiguration(
                redisInfo1.getHost(),
                redisInfo1.getPort());
        redisConfig.setPassword(redisInfo1.getPassword());
        redisConfig.setDatabase(redisInfo1.getDatabase());

        // Client Config 설정
        log.info("Step 2 - Client Config");
        LettuceClientConfiguration clientConfig = LettuceClientConfiguration.builder()
                .clientName(redisInfo1.getClientName())
                .build();

        return new LettuceConnectionFactory(redisConfig, clientConfig);
    }
}
```

Master-Replica 설정 방법 1

항목 설명

redis.replicas:를 replicas[0], replicas[1] 이렇게 배열로 구성했다.

Properties

application.properties

```
redis.master.host: 192.168.56.102
redis.master.port: 18510
redis.password: redisgate
redis.replicas[0].host: 192.168.56.102
redis.replicas[0].port: 18511
redis.replicas[1].host: 192.168.56.102
redis.replicas[1].port: 18512
redis.database: 0
redis.readFrom: replicaPreferred
redis.clientName: redisgate
```

ReadFrom 조회 설정

이것은 client 구성 설정으로 Master-replica, Sentinel, Cluster에 적용된다.
AutoConfiguration에는 이 항목이 없으므로 Manual(수동) 구성으로 해야한다.

권고 설정

- **복제(replica) 노드가 1개 일 때: replicaPreferred** 설정 권고: 입력은 마스터에서, 조회는 복제 노드에서 실행된다.
- **복제(replica) 노드가 2개 이상일 때: any** 설정 권고: 입력은 마스터에서, 조회는 마스터, 복제 노드들에서 실행된다. 복제 노드가 2개 이상일 때 replicaPreferred를 사용하면 1개 복제 노드에만 집중되는 경향이 있어서 일반적으로 권고하지 않는다. 하지만 마스터의 사용률이 높아서 조회는 복제 노드에 실행하고 싶다면 이것을 선택한다.
- 다른 설정은 복제 노드가 다운되면 에러가 발생하거나, 마스터에 집중되므로 권고하지 않는다.
- 각 값에 대한 자세한 설명은 여기를 클릭하세요.

Java 소스

RedisInfo.java: properties에서 항목을 읽어오는 클래스
Master 정보는 redisInfo.getMaster().getHost() or getPost()로 읽어온다.

Replicas 정보는 redisInfo.getReplicas().forEach(...)로 읽어온다.

```
package com.redisgate.redis2.config;

import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

import java.util.List;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ConfigurationProperties(prefix="redis")
@Configuration
public class RedisInfo {
    private String host;
    private int port;
    private String password;
    private int database;
    private String readFrom;
    private String clientName;

    private RedisInfo master;
    private List<RedisInfo> replicas;
}
```

RedisConfig.java: 연결(connection)을 구성하는 클래스

```
package com.redisgate.redis2.config;

import io.lettuce.core.ReadFrom;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisStaticMasterReplicaConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfiguration;

@Configuration
@RequiredArgsConstructor
@Slf4j
public class RedisConfig {
    private final RedisInfo redisInfo;

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {

        // RedisConfig 설정
        RedisStaticMasterReplicaConfiguration redisConfig = new RedisStaticMasterReplicaConfiguration(
                redisInfo.getMaster().getHost(),
                redisInfo.getMaster().getPort());
        redisConfig.setPassword(redisInfo.getPassword());
        redisConfig.setDatabase(redisInfo.getDatabase());
        log.info("Step 1 - Master Config");
        log.info("Master: {}:{}", redisInfo.getMaster().getHost(), redisInfo.getMaster().getPort());
        log.info("Database: {}", redisInfo.getDatabase());

        // Replicas 추가
        log.info("Step 2 - Replicas Config");
        redisInfo.getReplicas().forEach(replica -> log.info("{}:{}", replica.getHost(), replica.getPort()));
        redisInfo.getReplicas().forEach(replica -> redisConfig.addNode(replica.getHost(), replica.getPort()));

        ReadFrom readFrom = ReadFrom.valueOf(redisInfo.getReadFrom());
        if (readFrom == null) readFrom = ReadFrom.ANY;

        LettuceClientConfiguration clientConfig = LettuceClientConfiguration.builder()
                .readFrom(readFrom)
                .clientName(redisInfo.getClientName())
                .build();
        log.info("Step 3 - Client Config");

        return new LettuceConnectionFactory(redisConfig, clientConfig);
    }
}
```

Master-Replica 설정 방법 2

항목 설명

redis.master: host:port 방식으로 입력했다.
redis.replicas: host:port, host:port 방식으로 입력했다.

Properties

application.properties

```
redis.master: 192.168.56.102:18510
redis.replicas: 192.168.56.102:18511, 192.168.56.102:18512
redis.password: redisgate
redis.database: 0
redis.readFrom: replicaPreferred
redis.clientName: redisgate
```

Java 소스

RedisInfo2.java: properties에서 항목을 읽어오는 클래스

```
package com.redisgate.redis2.config;

import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ConfigurationProperties(prefix="redis")
@Configuration
public class RedisInfo2 {
    private String master;
    private String replicas;
    private String password;
    private int database;
    private String readFrom;
    private String clientName;
}
```

RedisConfig2.java: 연결(connection)을 구성하는 클래스

```
package com.redisgate.redis2.config;

import io.lettuce.core.ReadFrom;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisStaticMasterReplicaConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.util.Assert;
import org.springframework.util.StringUtils;

@Configuration
@RequiredArgsConstructor
@Slf4j
public class RedisConfig2 {
   private final RedisInfo2 redisInfo2;

   @Bean
   public LettuceConnectionFactory redisConnectionFactory() {

      String[] master = StringUtils.split(redisInfo2.getMaster(), ":");
      RedisStaticMasterReplicaConfiguration redisConfig = new RedisStaticMasterReplicaConfiguration(
            master[0], Integer.valueOf(master[1]));
      redisConfig.setPassword(redisInfo2.getPassword());
      redisConfig.setDatabase(redisInfo2.getDatabase());
      log.info("Master: {}:{}",master[0],master[1]);

      for (String node : StringUtils
            .commaDelimitedListToStringArray(redisInfo2.getReplicas())) {
         try {
            String[] replica = StringUtils.split(node, ":");
            Assert.state(replica.length == 2, "Must be defined as 'host:port'");
            redisConfig.addNode(replica[0].trim(), Integer.valueOf(replica[1]));
            log.info("Replica: {}:{}",replica[0].trim(),replica[1]);
         } catch (RuntimeException ex) {
            throw new IllegalStateException(
                  "Invalid redis sentinel " + "property '" + node + "'", ex);
         }
      }
```

```
        ReadFrom readFrom = null;
        try {
            readFrom = ReadFrom.valueOf(redisInfo2.getReadFrom());
        } catch (IllegalArgumentException e) {
            // "Name must not be empty"
            // "ReadFrom 'name' not supported"
            log.error("readFrom Exception ERROR: {} ", e.getMessage());
        }

        if (readFrom == null) {
            readFrom = ReadFrom.ANY;
            log.info("readFrom is ANY");
        } else {
            log.info("readFrom is {}",redisInfo2.getReadFrom());
        }

        LettuceClientConfiguration clientConfig = LettuceClientConfiguration.builder()
                .readFrom(readFrom)
                .clientName(redisInfo2.getClientName())
                .build();
        log.info("Step 3 - Client Config");

        return new LettuceConnectionFactory(redisConfig, clientConfig);
    }
}
```

Sentinel 설정 방법

항목 설명

redis.sentinel.nodes: host:port, host:port 방식으로 입력했다.

Properties

application.properties

```
redis.sentinel.nodes: 192.168.56.102:18525, 192.168.56.102:18526, 192.168.56.102:18527
redis.sentinel.master: mymaster
redis.sentinel.password: redisgate
redis.password: redisgate
redis.database: 0
redis.readFrom: replicaPreferred
redis.clientName: redisgate
```

application.yml

```
redis:
  sentinel:
    nodes: 192.168.56.102:18525, 192.168.56.102:18526, 192.168.56.102:18527
    master: mymaster
    password: redisgate
  password: redisgate
  database: 0
# replicaPreferred, any, anyReplica, replica
  readFrom: replicaPreferred
  clientName: redisgate
```

Java 소스

RedisInfo3.java: properties에서 항목을 읽어오는 클래스

```
package com.redisgate.redis2.config;

import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ConfigurationProperties(prefix="redis")
@Configuration
public class RedisInfo3 {
    private String nodes;
    private String master;
    private String password;
    private int database;
    private String readFrom;
    private String clientName;
    private RedisInfo3 sentinel;
}
```

RedisConfig3.java: 연결(connection)을 구성하는 클래스

```
package com.redisgate.redis2.config;

import io.lettuce.core.ReadFrom;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisSentinelConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.util.StringUtils;

import java.util.HashSet;
import java.util.Set;

@Configuration
@RequiredArgsConstructor
@Slf4j
public class RedisConfig3 {
    private final RedisInfo3 redisInfo3;

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {

        log.info("Step 1 - Redis Sentinel Config");
        log.info("Sentinel Nodes: {}",redisInfo3.getSentinel().getNodes());
        log.info("Sentinel Master: {}",redisInfo3.getSentinel().getMaster());

        Set<String> nodes = new HashSet<String>();
        for (String node : StringUtils
                .commaDelimitedListToStringArray(redisInfo3.getSentinel().getNodes())) {
            try {
                nodes.add(node.trim());
            } catch (RuntimeException ex) {
                throw new IllegalStateException(
                        "Invalid redis sentinel " + "property '" + node + "'", ex);
            }
        }
```

```
        RedisSentinelConfiguration redisConfig =
                new RedisSentinelConfiguration(redisInfo3.getSentinel().getMaster(), nodes);
              // Sentinel Password
        redisConfig.setSentinelPassword(redisInfo3.getSentinel().getPassword());
        redisConfig.setPassword(redisInfo3.getPassword()); // Redis Password
        ReadFrom readFrom = null;
        try {
            readFrom = ReadFrom.valueOf(redisInfo3.getReadFrom());
        } catch (IllegalArgumentException e) {
            // "Name must not be empty"
            // "ReadFrom name not supported"
            log.error("readFrom Exception ERROR: {} ", e.getMessage());
        }

        if (readFrom == null) {
            readFrom = ReadFrom.ANY;
            log.info("readFrom is ANY");
        } else {
            log.info("readFrom is {}",redisInfo3.getReadFrom());
        }

        LettuceClientConfiguration clientConfig = LettuceClientConfiguration.builder()
                .readFrom(readFrom)
                .clientName(redisInfo3.getClientName())
                .build();
        log.info("Step 2 - Client Config");

        return new LettuceConnectionFactory(redisConfig, clientConfig);
    }
}
```

Cluster 설정 방법

항목 설명

redis.master: host:port 방식으로 입력했다.
redis.replicas: host:port, host:port 방식으로 입력했다.

Properties

application.properties

```
redis.cluster.nodes: 192.168.56.102:18530, 192.168.56.102:18531, 192.168.56.102:18532,
  192.168.56.102:18533, 192.168.56.102:18534, 192.168.56.102:18535
redis.password: redisgate
redis.readFrom: replicaPreferred
redis.clientName: redisgate
```

application.yml

```
redis:
  cluster:
    nodes: 192.168.56.102:18530, 192.168.56.102:18531, 192.168.56.102:18532,
        192.168.56.102:18533, 192.168.56.102:18534, 192.168.56.102:18535
  password: redisgate
# replicaPreferred, any, anyReplica, replica
  readFrom: replicaPreferred
  clientName: redisgate
```

Java 소스

RedisInfo4.java: properties에서 항목을 읽어오는 클래스

```
package com.redisgate.redis2.config;

import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ConfigurationProperties(prefix = "redis")
@Configuration
public class RedisInfo4 {
    private String nodes;
    private String password;
    private String readFrom;
    private String clientName;
    private RedisInfo4 cluster;
}
```

RedisConfig4.java: 연결(connection)을 구성하는 클래스

```
package com.redisgate.redis2.config;

import io.lettuce.core.ReadFrom;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.util.StringUtils;

import java.util.HashSet;
import java.util.Set;

@Configuration
@RequiredArgsConstructor
@Slf4j
public class RedisConfig4 {
    private final RedisInfo4 redisInfo4;

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {

        log.info("Step 1 - Redis Cluster Config");
        log.info("Cluster Nodes: {}",redisInfo4.getCluster().getNodes());

        Set<String> nodes = new HashSet<String>();
        for (String node : StringUtils
                .commaDelimitedListToStringArray(redisInfo4.getCluster().getNodes())) {
            try {
                nodes.add(node.trim());
            } catch (RuntimeException ex) {
                throw new IllegalStateException(
                        "Invalid redis sentinel " + "property '" + node + "'", ex);
            }
        }
```

```
        RedisClusterConfiguration redisConfig = new RedisClusterConfiguration(nodes);
        redisConfig.setPassword(redisInfo4.getPassword());  // Redis Password

        ReadFrom readFrom = null;
        try {
            readFrom = ReadFrom.valueOf(redisInfo4.getReadFrom());
        } catch (IllegalArgumentException e) {
            // "Name must not be empty"
            // "ReadFrom 'name' not supported"
            log.error("readFrom Exception ERROR: {} ", e.getMessage());
        }

        if (readFrom == null) {
            readFrom = ReadFrom.ANY;
            log.info("readFrom is ANY");
        } else {
            log.info("readFrom is {}",redisInfo4.getReadFrom());
        }

        LettuceClientConfiguration clientConfig = LettuceClientConfiguration.builder()
                .readFrom(readFrom)
                .clientName(redisInfo4.getClientName())
                .build();
        log.info("Step 2 - Client Config");

        return new LettuceConnectionFactory(redisConfig, clientConfig);
    }
}
```

Properties 분리 방법

항목 설명

application.properties
spring.profiles.include=**redis-standalone**
or
spring.profiles.active=**redis-standalone**

지정한 이름으로 파일을 만든다.
application-**redis-standalone**.properties
설정은 이 파일에 한다.

class RedisProperties 설명

여기에 있는 항목(필드)를 application.properties에 설정하면 자동 적용된다.
application.properties: spring.data.redis로 시작한다.
예) spring.data.redis.host = localhost or 127.0.0.1
springboot 3.1.3 기준

Redis general Properties

No	Field	설명 Desc
1	String host = "localhost"	Redis server host(IP). redis.conf: bind
2	int port = 6379	Redis server port. redis.conf: port
3	int database = 0	Database index used by the connection factory. 기본으로 제공되는 16개(0~15번) DB를 접속시 선택한다. Spring Auto Config에서는 접속 후 DB를 변경할 수 없다. (select 명령이 제공되지 않는다) * 참고: Redis Cluster는 0번 DB만 사용된다.

4	String username	Login username of the redis server.
5	String password	Login password of the redis server. redis.conf: requirepass
6	Duration timeout	Read(command) timeout. Default 60초 io.lettuce.core.RedisURI DEFAULT_TIMEOUT = 60 (sec) Redis에 적절한 timeout: 3~5초 권장
7	String url	redis://[[username:]password@]host [: port][/database] [? [timeout=timeout[d|h|m|s|ms|us|ns]] [&database=database] [&clientName=clientName]] 예) redis://password@127.0.0.1:6379/0?timeout=10s 위와 같이 설정하면 password, host, port, database, timeout을 따로 설정하지 않아도 된다.
8	Duration connectTimeout	Connection timeout. Default 10초 io.lettuce.core.SocketOptions DEFAULT_CONNECT_TIMEOUT = 10 (sec)
9	String clientName	Client name to be set on connections with CLIENT SETNAME.
10	ClientType clientType	Type of client to use. By default, auto-detected according to the classpath. LETTUCE -> Use the Lettuce redis client. JEDIS -> Use the Jedis redis client.
11	Sentinel sentinel	센티널 노드, 마스터이름 등 설정, 아래 Sentinel properties 참조
12	Cluster cluster	클러스터 노드 등 설정, 아래 Cluster properties 참조
13	final Ssl ssl = new Ssl()	SSL: 보안 소켓 계층(Secure Sockets Layer) 사용 여부 설정 아래 SSL properties 참조
14	final Jedis jedis = new Jedis()	Jedis pool 설정, 아래 Jedis properties 참조
15	final Lettuce lettuce = new Lettuce()	Pool, Cluster topology refresh period 등 설정 아래 Lettuce properties 참조

Pool properties (static class)

Pool은 바로 지정할 수 없고 아래와 같이 lettuce, jedis 다음에 지정한다.

spring.data.redis.lettuce.pool.enabled = true

spring.data.redis.jedis.pool.enabled = true

No	Field	설명 Desc
1	Boolean enabled	Whether to enable the pool. Enabled automatically if "commons-pool2" is available. With Jedis, pooling is implicitly enabled in sentinel mode and this setting only applies to single node setup. Pool을 활성화할지 여부입니다. "commons-pool2"를 사용할 수 있는 경우 자동으로 활성화됩니다. Jedis를 사용하면 센티넬 모드에서 풀링이 암시적으로 활성화되며 이 설정은 단일 노드 설정에만 적용됩니다. spring.data.redis.lettuce.pool.enabled = true Lettuce: GenericObjectPoolConfig(), ClientResources
2	int maxIdle = 8	Maximum number of "idle" connections in the pool. Use a negative value to indicate an unlimited number of idle connections. Pool의 최대 "유휴(idle)" 연결 수입니다. 유휴 연결 수에 제한이 없음을 나타내려면 음수 값을 사용합니다.
3	int minIdle = 0	Target for the minimum number of idle connections to maintain in the pool. This setting only has an effect if both it and time between eviction runs are positive. Pool에서 유지 관리할 최소 유휴 연결 수의 목표입니다. 이 설정은 해당 설정과 제거 실행 사이의 시간이 모두 양수인 경우에만 효과가 있습니다.
4	int maxActive = 8	Maximum number of connections that can be allocated by the pool at a given time. Use a negative value for no limit. 주어진 시간에 pool에서 할당할 수 있는 최대 연결 수입니다. 제한이 없으면 음수 값을 사용합니다.
5	Duration maxWait = Duration.ofMillis(-1)	Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. Use a negative value to block indefinitely. Pool이 소진되었을 때 예외가 발생하기 전에 연결 할당이 차단되어야 하는 최대 시간입니다. 무기한 차단하려면 음수 값을 사용하세요.

6	Duration timeBetweenEvictionRuns	Time between runs of the idle object evictor thread. When positive, the idle object evictor thread starts, otherwise no idle object eviction is performed. 유휴(idle) 개체 축출기 스레드 실행 사이의 시간입니다. 양수이면 유휴 개체 제거 스레드가 시작되고, 그렇지 않으면 유휴 개체 제거가 수행되지 않습니다.

Cluster properties (static class)

No	Field	설명 Desc
1	List<String> nodes	Comma-separated list of "host:port" pairs to bootstrap from. This represents an "initial" list of cluster nodes and is required to have at least one entry. 부트스트랩할 "host:port" 쌍의 쉼표로 구분된 목록입니다. 이는 클러스터 노드의 초기(initial) 목록을 나타내며 하나 이상의 항목이 있어야 합니다. 마스터 노드 뿐만 아니라 복제 노드까지 모두 나열하세요. (마스터가 모두 다운되었을 경우를 대비해서)
2	Integer maxRedirects	Maximum number of redirects to follow when executing commands across the cluster. 클러스터 전체에서 명령을 실행할 때 따라야 할 최대 리디렉션 수입니다. 슬롯 이동(reshard)과 관련있습니다. 내부적으로 ClusterClientOptions class에서 int DEFAULT_MAX_REDIRECTS = 5 설정되어 있습니다. 특별한 사유가 없으면 기본값대로 사용하세요. (별도로 설정하지 마세요)

Sentinel properties (static class)

No	Field	설명 Desc
1	String master	Name of the Redis server. sentinel.conf: sentinel monitor **mymaster** 127.0.0.1 6379 2 application.properties: spring.data.redis.sentinel.master = **mymaster**
2	**List**<String> nodes	Comma-separated list of "host:port" pairs. application.properties: spring.data.redis.sentinel.nodes = **127.0.0.1:16379, 127.0.0.1:16380** **List**로 선언되어 있는데 구현에서 RedisSentinelConfiguration(String master, **Set<String>** sentinelHostAndPorts)는 Set으로 되어있으므로 Set으로 변경해야 한다. 예제) RedisSentinelConfiguration(redisInfo.getSentinel().getMaster(), **new HashSet<>(redisInfo.getSentinel().getNodes())**);
3	String username	Login username for authenticating with sentinel(s). 센티널에 username을 설정한 경우 sentinel.conf: sentinel sentinel-user <username>
4	String password	Password for authenticating with sentinel(s). 센티널에 password를 설정한 경우 sentinel.conf: requirepass <password> or sentinel sentinel-pass <password>

SSL properties (static class)

No	Field	설명 Desc
1	Boolean enabled	Whether to enable SSL support. Enabled automatically if "bundle" is provided unless specified otherwise. SSL 지원 활성화 여부. 별도로 지정하지 않는 한 "번들(bundle)"이 제공되면 자동으로 활성화 됩니다.
2	String bundle	SSL bundle name.

Jedis properties (static class)

No	Field	설명 Desc
1	final Pool pool = new Pool()	Jedis pool configuration.

Lettuce properties (static class)

No	Field	설명 Desc
1	Duration shutdownTimeout = Duration.ofMillis(100)	Shutdown timeout.

2	final Pool pool = new Pool()	Lettuce pool configuration.
3	final Cluster cluster = new Cluster()	아래 Cluster properties 참조.

Lettuce.Cluster properties (static class)

No	Field	설명 Desc
1	final Refresh refresh = new Refresh()	ClusterTopologyRefreshOptions: 기간(여부 포함) 설정한다. 아래 Refresh properties 참조.

Lettuce.Cluster.Refresh properties (static class)

No	Field	설명 Desc
1	boolean dynamicRefreshSources = true	Whether to discover and query all cluster nodes for obtaining the cluster topology. When set to false, only the initial seed nodes are used as sources for topology discovery. 클러스터 토폴로지를 얻기 위해 모든 클러스터 노드를 검색하고 쿼리할지 여부입니다. false로 설정하면 초기 시드(seed) 노드만 토폴로지 검색의 소스로 사용됩니다. true: CLUSTER NODES 명령을 실행해서 얻은 모든 노드들에 CLUSTER NODES 명령을 실행한다. false: spring.data.redis.cluster.nodes 에 지정된 노드들에만 CLUSTER NODES 명령을 실행한다.
2	**Duration period** (중요)	Cluster topology refresh period. 클러스터 토폴로지 새로고침 기간(주기). spring.data.redis.lettuce.cluster.refresh.period: 60s ClusterTopologyRefreshOptions.enablePeriodicRefresh(Duration.ofSeconds(60)) 설정된 주기마다 레디스 서버에 HELLO, CLIENT SETNAME, CLUSTER NODES, INFO 4개 명령을 실행합니다. 3 마스터, 3 복제 구성에서 마스터1번이 다운되면 새로고침(refresh) 전까지는 마스터1번에 입력되는 명령은 에러가 발생합니다. 다른 마스터에 입력되는 명령과 복제에 조회 명령은 정상적으로 실행됩니다. 새로고침 후에는 복제1번이 새 마스터가 된것을 인지해서 입력 명령이 정상적으로 실행됩니다. 설정해야 합니다. 설정하지 않으면 새로고침을 하지 않습니다. 적정 값: 30초 ~ 60초 (일반적인 기준)
3	boolean adaptive	Whether adaptive topology refreshing using all available refresh triggers should be used. 사용 가능한 모든 새로 고침 트리거를 사용하여 적응형 토폴로지 새로 고침을 사용해야 하는지 여부입니다. true: ClusterTopologyRefreshOptions.enableAllAdaptiveRefreshTriggers()

Spring Boot Auto Config Manual Config >>

redisgate@gmail.com 02.503.2235
서울시 강남구 강남대로 342 역삼빌딩 5층 (역삼동) 우 06242