4-10 [数据查询]高并发下的用户查询如何提速？

修复两个问题

1.之前我们写了一个配置类用于初始化数据库连接池的时候去提前建立连接：

这块我们统一将其挪动到 **qiyu-live-framework-datasource-starter** 模块中进行管理：

```
Java
package org.idea.qiyu.live.framework.datasource.starter;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.ApplicationRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;
import java.sql.Connection;

/**
 * 做成配置，实现通过参数来触发 Hikari 数据源的初始化
 *
 * @Author idea
 * @Date: Created in 18:06 2023/5/7
 * @Description
 */
@Configuration
public class ShardingJdbcDatasourceAutoInitConnectionConfig {

    private static final Logger LOGGER =
LoggerFactory.getLogger(ShardingJdbcDatasourceAutoInitConnectionCo
nfig.class);

    @Bean
    public ApplicationRunner runner(DataSource dataSource) {
        return args -> {
            LOGGER.info(" ==================
[ShardingJdbcDatasourceAutoInitConnectionConfig] dataSource: {}",
```

```
dataSource);
            //手动触发下连接池的连接创建
            Connection connection = dataSource.getConnection();
        };
    }
}
```

相关的 maven 依赖：

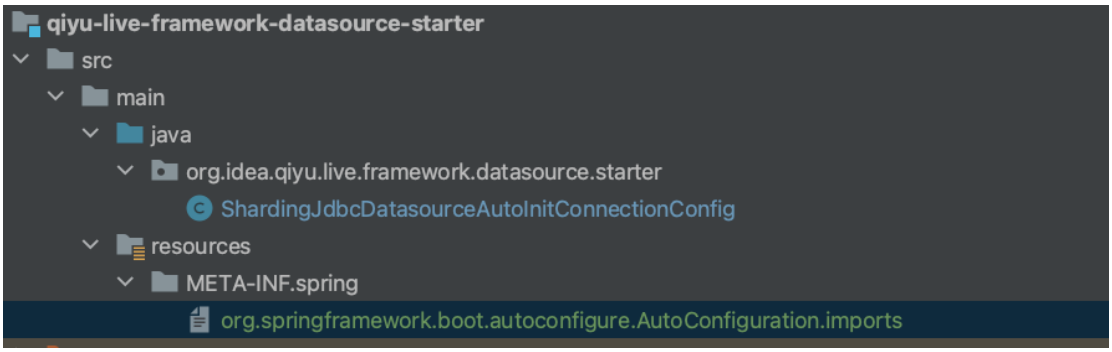
```XML
<description>提前初始化 MySQL 连接池</description>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

另外因为 springboot3 之后的 starter 机制改了下配置文件的位置，所以配置文件有所改动：

文件名：org.springframework.boot.autoconfigure.AutoConfiguration.imports

文件配置内容如下：

org.idea.qiyu.live.framework.datasource.starter.ShardingJdbcDatasourceAutoInitConnectionConfig



2.修改原先的数据库连接池配置，移除 connection-init-sql: select 1 的配置。

封装 Redis 基础组件

新建项目：

qiyu-live-framework

1. qiyu-live-framework-redis-starter

qiyu-live-framework-redis-starter 的 maven 依赖配置：

```XML
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <artifactId>log4j-to-slf4j</artifactId>
                <groupId>org.apache.logging.log4j</groupId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>jakarta.annotation</groupId>
        <artifactId>jakarta.annotation-api</artifactId>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

自定义 Redis 的配置

```Java
package org.idea.qiyu.live.framework.redis.starter.config;

import
org.springframework.boot.autoconfigure.condition.ConditionalOnClas
```

```
s;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import
org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
@ConditionalOnClass(RedisTemplate.class)
public class RedisConfig {

    @Bean
    public RedisTemplate<String, Object>
redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        RedisTemplate<String, Object> redisTemplate = new
RedisTemplate<>();

redisTemplate.setConnectionFactory(redisConnectionFactory);
        IGenericJackson2JsonRedisSerializer valueSerializer = new
IGenericJackson2JsonRedisSerializer();
        StringRedisSerializer stringRedisSerializer = new
StringRedisSerializer();
        redisTemplate.setKeySerializer(stringRedisSerializer);
        redisTemplate.setValueSerializer(valueSerializer);
        redisTemplate.setHashKeySerializer(stringRedisSerializer);
        redisTemplate.setHashValueSerializer(valueSerializer);
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }
}
```

Mapper 配置工厂

```
Java
package org.idea.qiyu.live.framework.redis.starter.config;

import com.fasterxml.jackson.annotation.JsonTypeInfo.As;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectMapper.DefaultTyping;
import com.fasterxml.jackson.databind.SerializerProvider;
import com.fasterxml.jackson.databind.module.SimpleModule;
```

```
import com.fasterxml.jackson.databind.ser.std.StdSerializer;
import org.springframework.cache.support.NullValue;
import org.springframework.util.StringUtils;

import java.io.IOException;

public class MapperFactory {

    public static ObjectMapper newInstance() {
        return initMapper(new ObjectMapper(), (String) null);
    }

    private static ObjectMapper initMapper(ObjectMapper mapper,
String classPropertyTypeName) {
        mapper.registerModule(new SimpleModule().addSerializer(new
MapperNullValueSerializer(classPropertyTypeName)));

        if (StringUtils.hasText(classPropertyTypeName)) {

mapper.enableDefaultTypingAsProperty(DefaultTyping.NON_FINAL,
classPropertyTypeName);
        } else {
            mapper.enableDefaultTyping(DefaultTyping.NON_FINAL,
As.PROPERTY);
        }


mapper.disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);

        return mapper;
    }


    /**
     * {@link StdSerializer} adding class information required by
default typing. This allows de-/serialization of
     * {@link NullValue}.
     *
     * @author Christoph Strobl
     * @since 1.8
     */
    private static class MapperNullValueSerializer extends
StdSerializer<NullValue> {
        private static final long serialVersionUID =
```

```
1999052150548658808L;
        private final String classIdentifier;
        /**
         * @param classIdentifier can be {@literal null} and will
be defaulted to {@code @class}.
         */
        MapperNullValueSerializer(String classIdentifier) {

            super(NullValue.class);
            this.classIdentifier =
StringUtils.hasText(classIdentifier) ? classIdentifier : "@class";
        }

        /*
         * (non-Javadoc)
         * @see
com.fasterxml.jackson.databind.ser.std.StdSerializer#serialize(jav
a.lang.Object, com.fasterxml.jackson.core.JsonGenerator,
com.fasterxml.jackson.databind.SerializerProvider)
         */
        @Override
        public void serialize(NullValue value, JsonGenerator jgen,
SerializerProvider provider)
                throws IOException {

            jgen.writeStartObject();
            jgen.writeStringField(classIdentifier,
NullValue.class.getName());
            jgen.writeEndObject();
        }
    }
}
```

序列化实现类

```
Java
package org.idea.qiyu.live.framework.redis.starter.config;

import
org.springframework.data.redis.serializer.GenericJackson2JsonRedis
Serializer;
import
org.springframework.data.redis.serializer.SerializationException;

public class IGenericJackson2JsonRedisSerializer extends
```

```
GenericJackson2JsonRedisSerializer {

    public IGenericJackson2JsonRedisSerializer() {
        super(MapperFactory.newInstance());
    }

    @Override
    public byte[] serialize(Object source) throws
SerializationException {

        if (source != null && ((source instanceof String) ||
(source instanceof Character))) {
            return source.toString().getBytes();
        }
        return super.serialize(source);
    }
}
```

KeyBuilder 的设计

设计一个统一的 Key 管理类,希望每个 Redis 的 key 都能有统一的前缀进行管理和匹配。

```
Java
package org.idea.qiyu.live.framework.redis.starter.key;

import org.springframework.beans.factory.annotation.Value;

/**
 * @Author idea
 * @Date: Created in 20:29 2023/5/14
 * @Description
 */
public class RedisKeyBuilder {

    @Value("${spring.application.name}")
    private String applicationName;
    private static final String SPLIT_ITEM = ":";

    public String getSplitItem() {
        return SPLIT_ITEM;
    }
```

```
    public String getPrefix() {
        return applicationName + SPLIT_ITEM;
    }
}
```

条件注入类

```
Java
package org.idea.qiyu.live.framework.redis.starter.keys;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Condition;
import org.springframework.context.annotation.ConditionContext;
import org.springframework.core.type.AnnotatedTypeMetadata;

import java.lang.reflect.Field;
import java.util.Arrays;
import java.util.List;

/**
 * @Author idea
 * @Date: Created in 20:38 2023/5/14
 * @Description
 */
public class RedisKeyLoadMatch implements Condition {

    private final static Logger LOGGER =
LoggerFactory.getLogger(RedisKeyLoadMatch.class);

    private static final String PREFIX = "qiyu";

    @Override
    public boolean matches(ConditionContext context,
AnnotatedTypeMetadata metadata) {
        String appName =
context.getEnvironment().getProperty("spring.application.name");
        if (appName == null) {
            LOGGER.error("没有匹配到应用名称，所以无法加载任何
RedisKeyBuilder 对象");
            return false;
        }
```

```
        try {
            Field classNameField =
metadata.getClass().getDeclaredField("className");
            classNameField.setAccessible(true);
            String keyBuilderName = (String)
classNameField.get(metadata);
            List<String> splitList =
Arrays.asList(keyBuilderName.split("\\."));
            //忽略大小写，统一用 qiyu 开头命名
            String classSimplyName = PREFIX +
splitList.get(splitList.size() - 1).toLowerCase();
            boolean matchStatus =
classSimplyName.contains(appName.replaceAll("-", ""));
            LOGGER.info("keyBuilderClass is {},matchStatus is {}",
keyBuilderName, matchStatus);
        } catch (NoSuchFieldException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        }
        return true;
    }
}
```

用户中台专属的 keyBuiler

```
Java
package org.idea.qiyu.live.framework.redis.starter.key;

import org.springframework.beans.factory.annotation.Configurable;
import org.springframework.context.annotation.Conditional;

/**
 * @Author idea
 * @Date: Created in 20:31 2023/5/14
 * @Description
 */
@Configurable
@Conditional(RedisKeyLoadMatch.class)
public class UserProviderCacheKeyBuilder extends RedisKeyBuilder{

    private static String USER_INFO_KEY = "userInfo";

    public String buildUserInfoKey(Long userId) {
```

```
        return super.getPrefix() + USER_INFO_KEY +
super.getSplitItem() + userId;
    }

}
```

对照的另一个 keyBuilder 类

```
Java
package org.idea.qiyu.live.framework.redis.starter.key;

import org.springframework.beans.factory.annotation.Configurable;
import org.springframework.context.annotation.Conditional;

/**
 * @Author idea
 * @Date: Created in 20:31 2023/5/14
 * @Description
 */
@Configurable
@Conditional(RedisKeyLoadMatch.class)
public class OtherCacheKeyBuilder extends RedisKeyBuilder {

    private static String USER_INFO_KEY = "other";

    public String buildUserInfoKey(Long userId) {
        return super.getPrefix() + USER_INFO_KEY +
super.getSplitItem() + userId;
    }

}
```

spring-starter 配置

/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports 文件中写入：

1. org.idea.qiyu.live.framework.redis.starter.config.RedisConfig
1. org.idea.qiyu.live.framework.redis.starter.keys.UserProviderCacheKeyBuilder