一、普通项目使用说明

两种模式介绍

Leaf-snowflake模式

- Leaf-snowflake不同于原始snowflake算法地方，主要是在workId的生成上，Leaf-snowflake依靠Zookeeper生成workId。Leaf中workId是基于ZooKeeper的顺序Id来生成的，每个应用在使用Leaf-snowflake时，启动时都会都在Zookeeper中生成一个顺序Id，相当于一台机器对应一个顺序节点，也就是一个workId。
- 生成的id如下

```
order_id: 1234644838356353109
order_id: 1234644838759006288
order_id: 1234644839161659450
```

- 特点 1、全局唯一性：不能重复 2、信息安全：如何防止用户恶意根据订单号获取数据 3、数据递增: 保证下一个订单号要大于上一个订单号

Leaf-segment号段模式

- Leaf-segment号段模式是对直接用数据库自增ID充当分布式ID的一种优化，减少对数据库的频率操作。
- 生成的段号如1、2、3、4 连续
- 主要应用场景微服务下的1、业务编号生成 2、微服务场景下特殊ID规则生成如：年份+业务编号+序号。其中序号部分可用leaf-segment生成。
- 注意：由于提高发号并发效率，有将号码按照step存储在内存中，重启发号器会出现断号（其中一些id 没有被使用），为了减少断号最好是单独部署个发号器微服务。(待优化)

项目整合使用leaf（Jar方式）

- 使用发号器可以引用jar和微服务调用的方式，下面先介绍使用**jar方式**

1、引入pom

```
    <properties>
        <loit-build-commom-parent.version>1.0-SNAPSHOT</loit-build-commom-
parent.version>
    </properties>

<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.timeloit.project</groupId>
                <artifactId>loit-build-commom-parent</artifactId>
                <version>${loit-build-commom-parent.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

```
        <dependency>
            <groupId>com.timeloit.project</groupId>
            <artifactId>loit-keygen-leaf-config</artifactId>
        </dependency>
```

2、配置文件

指定zookeeper 配置文件地址

```
leaf:
  namespace: leaf_${spring.application.name}
  zkserver:
    # zookeeper 地址
    list: 127.0.0.1:2181
  segment:
    # 初始化Id
    id.initial.value: 1
    # 步长
    step: 50
```

2、注解使用方式

如果是实体id 可以使用注解的方式如： @AutoId

```
public class Order implements Serializable {

    private static final long serialVersionUID = 661434701950670670L;

    @AutoId
    private String orderId;

    private Integer userId;

    private Long addressId;

    private String status;
}
```

- @AutoId 可以指定LEAF_SNOWFLAKE、LEAF_SEGMENT。默认是使用LEAF_SNOWFLAKE。如果使用LEAF_SEGMENT 如：**@AutoId(AutoId.IdType.LEAF_SEGMENT)**
- orderId 的数据类型可以是**String**(数据库varchar)和 **Long**(数据库bigint) 如果是leaf_snowflake 长度可以指定 **20**
- mybatis sql配置如下，需包含order_id 字段

```
    <insert id="insert" useGeneratedKeys="true" keyProperty="orderId">
        INSERT INTO t_order (order_id, user_id, address_id, status) VALUES (#
{orderId,jdbcType=VARCHAR}, #{userId,jdbcType=INTEGER}, #
{addressId,jdbcType=BIGINT}, #{status,jdbcType=VARCHAR});
    </insert>
```

3、代码调用方式

LEAF_SEGMENT:

```
    @GetMapping("/leafSegment")
    public void leafSegment() {
        Comparable<?> t_order_id =
leafSegmentKeyGeneratorFactory.getKeyGenerator(Order.class.getName()).generateKe
y();
        System.out.println(t_order_id);
    }
```

LEAF_SNOWFLAKE:

```

    @Autowired
    private IDGen idGen;

    @GetMapping("/leafSnowflake")
    public void leafSnowflake() {
        Result result = idGen.get("");
        System.out.println(result);
    }
```

项目整合使用leaf（Feign 接口方式）

leaf 发号器微服务代码 LoitKeyGenerate 地址如下

http://39.100.254.140:12011/loit-Infrastructure/LoitKeyGenerate

1、启动LoitKeyGenerate 微服务

2、编写feign 调用

```
@FeignClient(name = "loit-keygen-server")
public interface KeyGenApiService {

    @RequestMapping(value = "/api/segment/get/{key}")
    public String getSegmentId(@PathVariable("key") String key);

    @RequestMapping(value = "/api/snowflake/get/{key}")
    public String getSnowflakeId(@PathVariable("key") String key);
}
```

```
    @Autowired
    private KeyGenApiService keyGenApiService;


    @GetMapping("/keyGenFeign")
    public void keyGenFeign() {
        String segId = keyGenApiService.getSegmentId("t_order");
        log.info(segId);
        String snowId = keyGenApiService.getSnowflakeId("t_order");
        log.info(snowId);
    }
```

二、shardingsphere使用分布式ID

- 分表分库中使用分布式id 可参照例子demo: sharding-leaf-mybatis-example

项目整合分布式id

1、引入pom

```
    <properties>
        <loit-build-commom-parent.version>1.0-SNAPSHOT</loit-build-commom-
parent.version>
    </properties>

<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.timeloit.project</groupId>
                <artifactId>loit-build-commom-parent</artifactId>
                <version>${loit-build-commom-parent.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

```
        <dependency>
            <groupId>com.timeloit.project</groupId>
            <artifactId>sharding-keygen-leaf</artifactId>
        </dependency>
```

1、配置

LEAF_SNOWFLAKE 配置如下

```
spring.shardingsphere.sharding.tables.t_order.key-generator.type=LEAF_SNOWFLAKE
spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.zk.list=localhost:2181
```

LEAF_SEGMENT 配置如下

```
spring.shardingsphere.sharding.tables.t_order.key-generator.type=LEAF_SEGMENT
spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.key=t_order_id
spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.jdbc.url=jdbc:mysql://39.98.202.173:3306/leaf?
serverTimezone=UTC&useSSL=false
spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.jdbc.username=root
spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.jdbc.password=abcd1234A!
```

备注 全部配置文件如下:

```
spring.shardingsphere.datasource.names=ds_0,ds_1

spring.shardingsphere.datasource.ds_0.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds_0.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds_0.jdbc-
url=jdbc:mysql://39.98.202.173:3306/demo1_ds_0?
serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds_0.username=root
spring.shardingsphere.datasource.ds_0.password=abcd1234A!

spring.shardingsphere.datasource.ds_1.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds_1.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds_1.jdbc-
url=jdbc:mysql://39.98.202.173:3306/demo1_ds_1?
serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds_1.username=root
spring.shardingsphere.datasource.ds_1.password=abcd1234A!

spring.shardingsphere.sharding.default-database-strategy.inline.sharding-
column=user_id
spring.shardingsphere.sharding.default-database-strategy.inline.algorithm-
expression=ds_$->{user_id % 2}
spring.shardingsphere.sharding.binding-tables=t_order,t_order_item
spring.shardingsphere.sharding.broadcast-tables=t_address

spring.shardingsphere.sharding.tables.t_order.actual-data-nodes=ds_$->
{0..1}.t_order
spring.shardingsphere.sharding.tables.t_order.key-generator.column=order_id

# 1、雪花算法
#spring.shardingsphere.sharding.tables.t_order.key-generator.type=SNOWFLAKE
#spring.shardingsphere.sharding.tables.t_order.key-generator.props.worker.id=123
# 2、LEAF_SEGMENT
#spring.shardingsphere.sharding.tables.t_order.key-generator.type=LEAF_SEGMENT
#spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.key=t_order_id
#spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.jdbc.url=jdbc:mysql://39.98.202.173:3306/leaf?
serverTimezone=UTC&useSSL=false
```

```
#spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.jdbc.username=root
#spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.jdbc.password=abcd1234A!

# 2、LEAF_SNOWFLAKE
spring.shardingsphere.sharding.tables.t_order.key-generator.type=LEAF_SNOWFLAKE
spring.shardingsphere.sharding.tables.t_order.key-
generator.props.leaf.zk.list=localhost:2181

spring.shardingsphere.sharding.tables.t_order_item.actual-data-nodes=ds_$->
{0..1}.t_order_item
spring.shardingsphere.sharding.tables.t_order_item.key-
generator.column=order_item_id
spring.shardingsphere.sharding.tables.t_order_item.key-generator.type=SNOWFLAKE
spring.shardingsphere.sharding.tables.t_order_item.key-
generator.props.worker.id=123
```