

lettuce 学习记录

简单知识点集合

- Redis 的连接是 `长连接` 并 `线程安全` 的;
- 如果链接丢失, 会自动重连接;
- 执行中并且没有超时的命令, 会在重连接后再次发送指令;
- 每个连接都有一个默认超时时间(创建client的时候可以设置, 默认是60s);
- 所有异常父类—— `RedisException`
- `同步` 状态下, 如果Redis返回错误, 框架会抛出异常 `RedisCommandExecutionException`, 但是 `异步` 状态下, 不会抛出异常;

** Redis URI **支持的格式:

- `redis://[password@]host[:port][/databaseNumber]` 一般形式
- `rediss://[password@]host[:port][/databaseNumber]` SSL形式(3.1开始支持单例模式, 4.2开始支持集群模式)
 - 此框架 (`Lettuce` 没有实现SSL, 而是使用的 `stunnel` 工具)
 - [SSL连接-详细文档](#)
- `redis-sentinel://[password@]host[:port][,host2[:port2]]`
`[/databaseNumber]#sentinelMasterId`
 Redis-Sentinel形式
- `redis-socket:///tmp/redis` `Unix Domain Socket` connection to local Redis;
 - `Unix Domain Socket` 只对本地的单例模式的Redis有效, 对集群没用; 但是集群可以使用 `RedisClusterClient`, 实现对本地节点使用 `Unix Domain Socket`, 集群的其他节点使用TCP;
- `Redis Sentinel` 和 `Redis Master/Slave` 不支持SSL
- 使用SSL获取连接之前会有一个握手 (`handshake`), 如果握手发生错误, 会抛出 `RedisConnectionExceptions` ;
- 可以使用 `redisClient.setOptions` 对client进行设置, 但是当 `connection` 创建完成之后, `ClientOption` 就不可再变; 即使修改Option也不会影响client
 - 还可使用其他2种方式:

lettuce uses Java defaults for the `trust store` that is usually `cacerts` in your `jre/lib/security` directory and comes with customizable SSL options via `Client options` .

① If you need to add your own root certificate, so you can configure `SslOptions`, import it either to `cacerts` or ② you provide an own trust store and set the necessary system properties.

- 默认不支持 `TLS`，可以通过 `redisUri.setStartTls(true)` 打开；

创建 `RedisClient` 实例：

- 方式一：

```
RedisURI redisUri =  
RedisURI.create("redis://authentication@localhost/2");  
RedisClient client = RedisClient.create(redisUri);
```

- 方式二：

```
RedisURI redisUri = RedisURI.Builder.redis("localhost")  
    .withSsl(true) // SSL  
    .withPassword("authentication")  
    .withDatabase(2)  
    .build();  
RedisClient client = RedisClient.create(redisUri);
```

- 方式三：

```
RedisClient redisClient =  
RedisClient.create("redis://192.168.76.130:6379/0");  
StatefulRedisConnection<String, String> connection =  
redisClient.connect();
```

Client options

- `Since: 3.1, 4.0`：实现连接之前进行 `PING` 检测，如果ping失败，那么就表示一定连接失败；

Redis Cluster

- 连接 **集群** 的时候，不用填写所有节点的URI，只连接其中的一个节点，然后会自动感知 **Redis集群** 的 **拓扑 (topology)** 结构，连接到其他节点；
- **lettuce** 会为每一个节点建立一个 **connection**；每一个连接 (**connection**) 会绑定到特定的 **nodeId** 或 **host/port** 上，即使 **nodeId** 所对应的 **host/port** 变动了；
- **lettuce** 支持一些 **Cross-slot** 操作；
- 获取所有key和所有节点（必须再 **异步** 模式下）：
 - **lettuce** 支持选择部分节点（放入一个集合，eg: Set），然后在集合 (Set) 中执行命令；

```
RedisAdvancedClusterAsyncCommands<String, String> async =
clusterClient.connect().async();
AsyncNodeSelection<String, String> slaves =
connection.slaves();

AsyncExecutions<List<String>> executions =
slaves.commands().keys("*");
executions.forEach(result -> result.thenAccept(keys ->
System.out.println(keys)));
```

- **lettuce** 支持在Runtime时向集群中添加节点；**RedisClusterClient** 管理着集群的拓扑结构；

同步/异步

- **异步** 和 **同步** 的区别在于发出 **命令请求(Request)** 到接收到 **Redis** 回复 (**Response**) 这段时间内，线程是否等待（阻塞）。
- **异步** Api创建一个 **RedisFuture**，用户异步处理结果的结合；``java // Future的使用示例
final CompletableFuture future = new CompletableFuture<>();

```
future.thenRun(new Runnable() {
    @Override
    public void run() {
        try {
            System.out.println("Got value: " + future.get());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```

```
System.out.println("Current state: " + future.isDone()); // false
future.complete("my value");
System.out.println("Current state: " + future.isDone()) // true
```

- 使用 `lettuce` 获取 `Future` :

```
RedisClient client = RedisClient.create("redis://localhost");
RedisAsyncCommands<String, String> commands =
client.connect().async();
RedisFuture<String> future = commands.get("key");
```

Publish/Subscribe

- 主要通过实现 `RedisPubSubListener` 接口来实现发布/订阅功能;
- `lettuce` 提供了一个 `RedisPubSubListener` 的简单实现: `RedisPubSubAdapter` ;
- 订阅示例:
 - **Synchronous subscription**

```
// 集群的话使用:StatefulRedisClusterConnection <String, String>
connection = redisClusterClient.connectPubSub()
StatefulRedisPubSubConnection<String, String> connection =
client.connectPubSub();
connection.addListener(new RedisPubSubListener<String, String>() {
... })

RedisPubSubCommands<String, String> sync = connection.sync();
sync.subscribe("channel");
```

- **Asynchronous subscription**

```
StatefulRedisPubSubConnection<String, String> connection =
client.connectPubSub()
connection.addListener(new RedisPubSubListener<String, String>()
{ ... })

RedisPubSubAsyncCommands<String, String> async =
connection.async();
```

```
RedisFuture<Void> future = async.subscribe("channel");
```

- ** Reactive API **机制搞不懂;

Connection Events

- ** 3.4/4.1 **版本之前: lettuce 会向用户发送 `connected`、`Disconnected`、以及各种异常 (原文: `Exceptions in the connection handler pipeline`);
 - `RedisClient#addListener()` 监听事件
 - `RedisClient.removeListener()` 取消事件监听
- ** 3.4/4.1 **版本之后: 由client发起事件, 并通过事件总线 (`EventBus`) 传递, 可以产生的事件包括: `Connection events`、`Metrics events`、`Cluster topology(拓扑) events`
 - 事件总线的配置包含在 `client resources` 中; `java RedisClient client = RedisClient.create() EventBus eventBus = client.getresources().eventBus();`

```
eventBus.get().subscribe(new Action1<Event>() {
    @Override
    public void call(Event event) {
        System.out.println(event);
    }
});

...
client.shutdown();
...
}
```

- 几种事件的具体解释可以查看[官方文档](#) * `First Response Latency`: 从 `命令开始发送` 开始 到 `接收到第一个返回字节` 结束 * `Completion Latency`: 从 `命令开始发送` 开始 到 `所有返回字节都已接收到并将字节处理完成` * `Cluster events` 不是立马响应, 因为拓补视图是客户端从 `集群` 拉取得到的;

Transactions

- 同步的时候, 如果命令在一个事务中执行, 那么将返回一个 `null` ;
- 如果是 `异步` , 在命令被处理的时刻, `Future` 会收到 `Response` ;
 - 异步的结果可以获取2次: 既可以在每个命令的 `Future` 中获取, 也可以在 `exe()` 的 `Future` 中获取; 因为异步时候, 它的命令是单独执行的;

- 如果是事件模式 `Reactive`，在命令被处理的时刻，监视器会触发 `onNext` / `onCompleted` 事件；

CDI

- 去了解什么是[CDI](#)吧！

Spring Support

- `Spring Data Redis` 使用了 `lettuce` 作为连接 `Redis` 的工具；

其他

- `lettuce-shaded` jar包中包含有各种 `Connection Interfaces` API；
- 默认编码集是 `UTF-8`；

```
// 自定义编码器
StatefulRedisConnection<K, V> connect(RedisCodec<K, V> codec);
StatefulRedisPubSubConnection<K, V> connectPubSub(RedisCodec<K, V> codec);
```

- `Redis` 中可能保存各种不同类型的数据，所以每个 `key` 和 `value` 之间的编码格式没有相互关联性，这样不同类型的数据就可以同时保存（比如：`stream`、`string`）；
- `lettuce` 可以通过配置 `codec` 实现对key和数据的压缩，比如：`GZIP`；