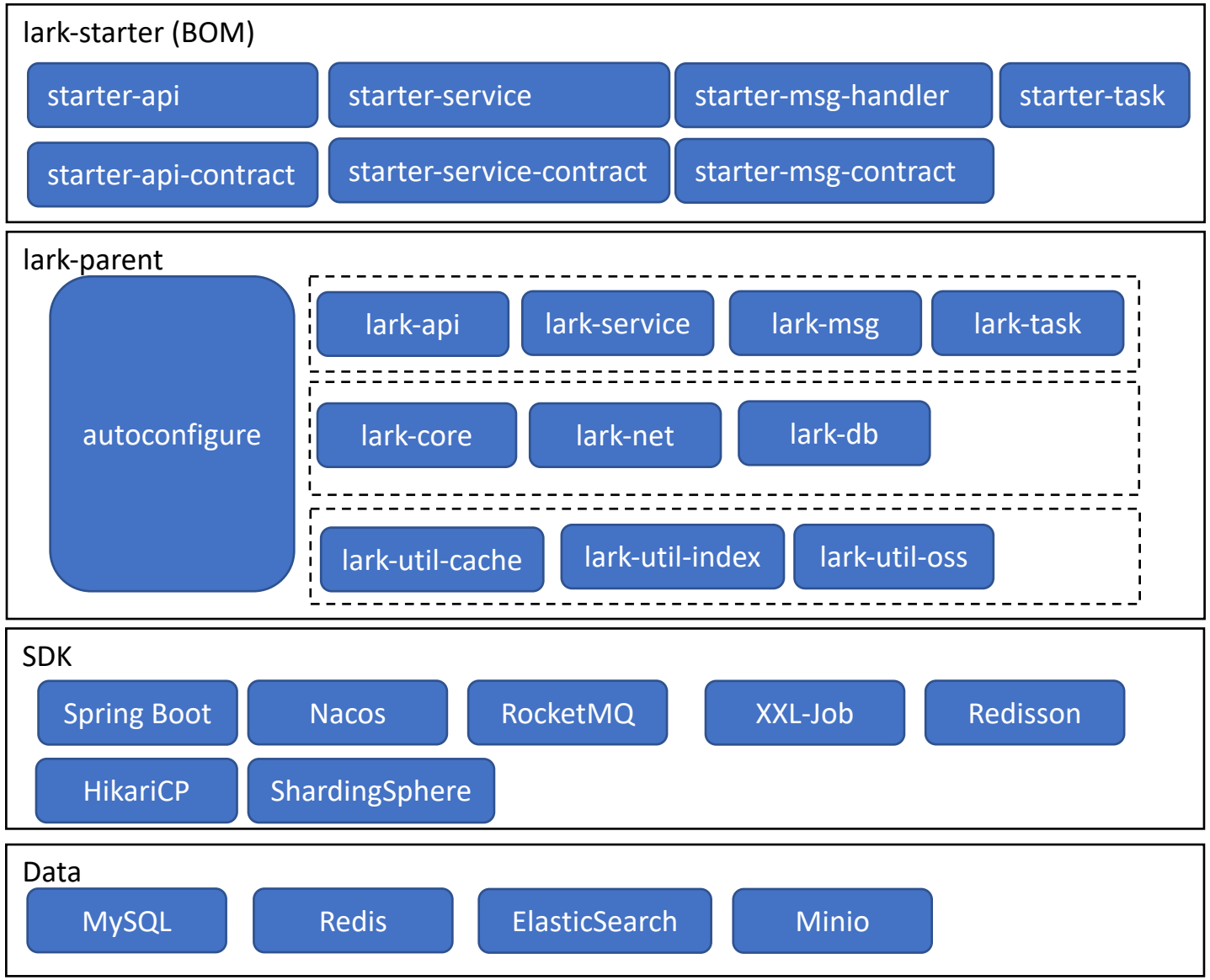
# lark

Ver 1.5.0说明

# 目录

- 架构
- 模块
- 组件

# 架构（工程化）




```
|-- file #文件模块
|   |-- api
|   |-- api-contract
|   |-- msg-contract
|   |-- msg-handler
|   |-- service
|   |-- service-contract
|   |-- task
|   |-- pom.xml
|-- user #会员模块
|   |-- admin-api
|   |-- admin-api-contract
|   |-- admin-service
|   |-- admin-service-contract
|   |-- api
|   |-- api-contract
|   |-- msg-contract
|   |-- msg-handler
|   |-- service
|   |-- service-contract
|   |-- task
|   |-- pom.xml
|-- merchant #商家模块
|-- goods #商品模块
|-- pay #支付模块
...
```

# 目录

- 架构
- 模块
- 组件
- Playground环境

# 模块

- api
  - 面向前端/客户端的接口模块
- service
  - 提供服务的模块
- msg-handler
  - 处理消息的模块
- task
  - 计划任务执行模块

# 模块-扩展（拆分协议）

- api
  - 面向前端/客户端的接口模块
  - api-contract
- service
  - 提供服务的模块
  - service-contract
- msg-handler
  - 处理消息的模块
  - msg-contract
- task
  - 计划任务执行模块

# 模块-扩展（拆分前后台）

- api
  - 面向前端/客户端的接口模块
  - api-contract
  - admin-api/admin-api-contract
- service
  - 提供服务的模块
  - service-contract
  - admin-service/admin-service-contract
- msg-handler
  - 处理消息的模块
  - msg-contract
- task

# 目录

- 架构
- 模块
- 组件

# 组件

- Spring & SpringMvc & SpringBoot
- Nacos
  - 服务发现
- mysql（JDBC & JSD）
  - 数据存储
- rocketmq
  - 消息队列
- xxl-job
  - 计划任务调度
- elastic search
  - 搜索
- Redis（redisson）
  - 缓存
- Oss（minio）
  - 对象存储

# 相关组件-服务&调用服务

接口定义

```java
/**
 * 测试服务
**/
@RpcService(description = "测试服务")
public interface TestService {
    /**
     * 测试
    **/
    @RpcMethod(description = "测试")
    HelloResponse hello(HelloRequest request);
}
```

接口实现

```java
@Service("测试服务")
public class TestServiceImpl implements TestService {
    @Autowired
    private TestBiz testBiz;

    @Override
    public TestDto.HelloResponse hello(TestDto.HelloRequest request) {
        TestObject object = testBiz.getObject(request.getId());
        TestDto.HelloResponse response = new TestDto.HelloResponse();
        response.setTime( Times.toEpochMilli( LocalDateTime.now().minusDays(-1) ) );
        response.setType(TestType.GOOD);
        response.setResult(object.getName());
        return response;
    }
```

接口调用

```java
@Autowired
private TestService testService;

public TestVo.HelloResponse hello( TestVo.HelloRequest request ) {
    TestDto.HelloRequest helloRequest = new TestDto.HelloRequest();
    helloRequest.setId( request.getId() );
    helloRequest.setType( TestType.GOOD );
    TestDto.HelloResponse helloResponse = testService.hello( helloRequest );
    //
    TestVo.HelloResponse response = new TestVo.HelloResponse();
    response.setResult( helloResponse.getResult() );
    response.setTime( helloResponse.getTime() );
    return response;
}
```

# 相关组件-数据库访问

数据库连接配置

```yaml
lark:
  db:
    source:
      - user_master
      - order_master_0
      - order_master_1
    user_master:
      name: demo
      address: db-dev.lark-cloud.com:3306
      username: lark
      password: 12345678
      type: mysql
    order_master_0:
      name: demo_order_0
      address: db-dev.lark-cloud.com:3306
      username: lark
      password: 12345678
    order_master_1:
      name: demo_order_1
      address: db-dev.lark-cloud.com:3306
      username: lark
      password: 12345678
```

数据库分片配置（ShardingSphere）

```yaml
shard:
  - order
order:
  database: order_master_0, order_master_1
  route: order_master_$->{0..1}.t_order_$->{0..1}
  database-sharding:
    column: user_id
    algorithm: order_master_$->{user_id % 2}
  table-sharding:
    column: order_id
    algorithm: t_order_$->{order_id % 2}
```

# 相关组件-数据库访问

访问数据库

```
@Autowired
DatabaseService databaseService;

public TestDO getObject(int id) {
    SqlQuery userSqlQuery = databaseService.get( "user_master" );
    UserDO user = userSqlQuery.select( ...columns: "id", "name" ) SelectClause
            .from( table: "users" ) FromClause
            .where( f( column: "id", id ) ) WhereClause
            .one( UserDO.class );
    //
    TestDO object = new TestDO();
    if ( user != null ) {
        object.setId( user.getId() );
        object.setName( user.getName() );
```

访问分片的数据库

```
@Autowired
DatabaseService databaseService;

public OrderDO getOrder(long orderId) {
    SqlQuery orderSqlQuery = databaseService.getShard( logicTableName: "order" );
    List<OrderDO> orders = orderSqlQuery.select( ...columns: "order_id", "user_id", "
            from( table: "order").
            where( f( column: "order_id", orderId )).
            list( OrderDO.class );
    if ( orders != null && orders.size() > 0 ) {
        return orders.get(0);
    }
    return null;
}
```

# 相关组件-缓存

缓存连接配置

```yaml
lark:
  util:
    cache:
      address: cache-dev.lark-cloud.com:6379
      password: 12345678
```

访问缓存

```java
@Autowired
CacheService cacheService;
```

```java
cacheService.set( key: "test", value: "123", Duration.ofMinutes(3));
String v = cacheService.get( "test" );
LOGGER.info( "Test Cache: >>> test: {}", v );
UserItem item = new UserItem();
item.setId(123);
item.setName( "123");
cacheService.set( key: "testuser", item, Duration.ofMinutes( 3 ) );
item = cacheService.get( key: "testuser", UserItem.class );
LOGGER.info( "Test Cache: >>> userid: {}", item.getId() );
```

# 相关组件-搜素索引

## 索引连接配置

```yaml
lark:
  util:
    index:
      address: index-dev.lark-cloud.com:9200
```

## 访问索引

```java
@Service
public class UserIndexService {
    /**
     * 索引名称，一般为：项目名.模块名.索引名
     */
    private static final String INDEX_NAME = "lark.example.index.user";
    private static final int EXCEPTION_CODE = 10009;
    /**
     * 注入索引服务
     */
    @Autowired
    IndexService indexService;
    /**
     * 保存或修改用户文档
     *
     * @param user 用户信息
     */
    public void save( UserDocument user ) {
        IndexDocument<UserDocument> document = new IndexDocument<>();
        document.setId( String.valueOf( user.getId() ) );
        document.setData( user );
        try {
            indexService.save(INDEX_NAME, document);
        } catch (IOException e) {
            throw new BusinessException( EXCEPTION_CODE, "Failed to add user index", e );
        }
    }
```

# 相关组件-对象存储

OSS连接配置

```yaml
lark:
  util:
    oss:
      address: oss-dev.lark-cloud.com
      username: minio
      password: 12345678
      type: minio
```

访问OSS

```java
@Autowired
OssService ossService;
private final String BUCKET_NAME = "public";

@Override
public TestVo.HelloResponse hello(TestVo.HelloRequest hello) throws IOException {
    // 测试Oss
    File file = new File( pathname: "/Users/andy/Downloads/1.jpg" );
    byte[] fileData = Files.readAllBytes(file.toPath());
    String objectName = ossService.upload( BUCKET_NAME, file.getName(), fileData );
    String objectUrl = ossService.getObjectUrl( BUCKET_NAME, objectName );
    fileData = ossService.download( BUCKET_NAME, objectName );
```

# 目录

- 架构
- 模块
- 组件
- Playground环境

# Playground环境（用于熟悉框架和流程）

- WIFI: techwis-ac68u
- Gitlab: code-repo-dev.lark-cloud.com
  - 框架：code-repo-dev.lark-cloud.com/lark-projects/lark-doc/-/blob/develop/README.md
  - 环境：**code-repo-dev.lark-cloud.com/lark-projects/lark-doc/-/blob/develop/Playground.md**
- Nexus: package-repo-dev.lark-cloud.com
  - 配置文件: code-repo-dev.lark-cloud.com/lark-projects/lark-doc/-/blob/develop/config/playground/nexus/settings.xml
- Harbor: image-repo-dev.lark-cloud.com
- ……