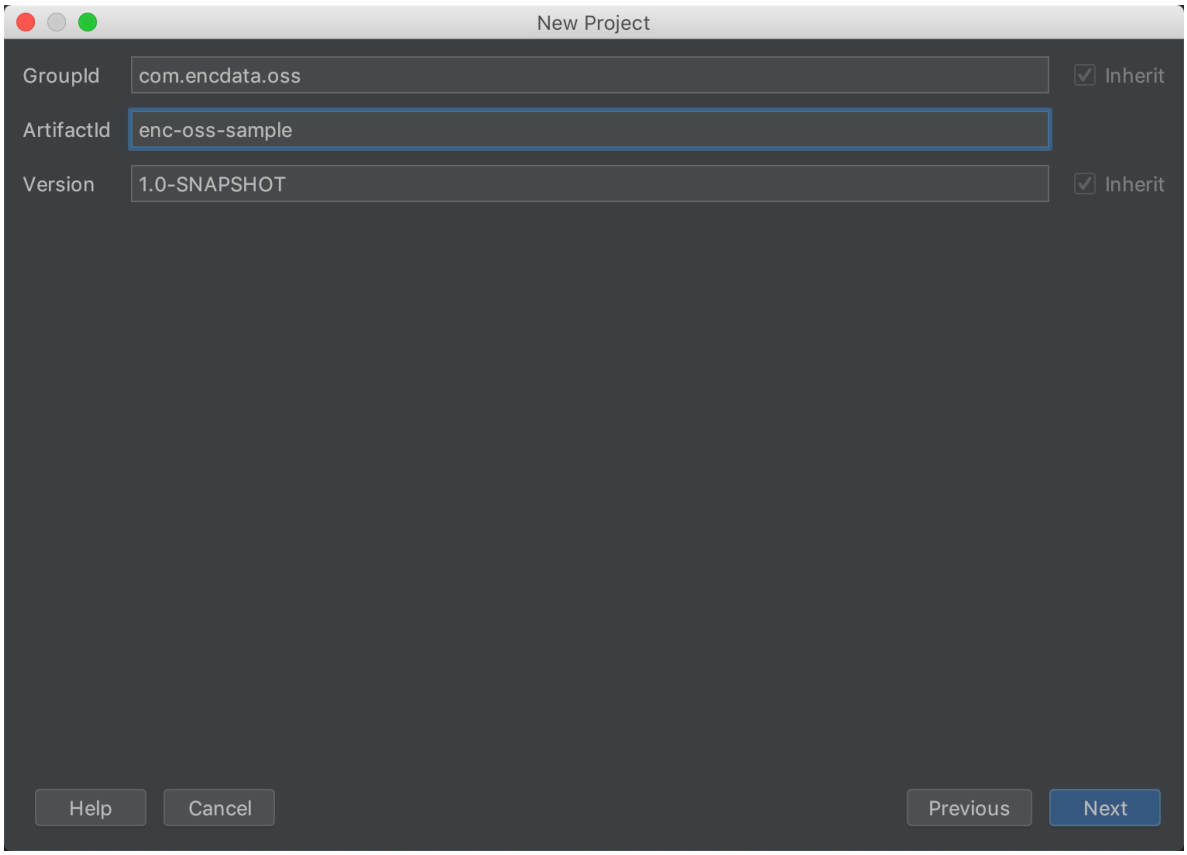
框架使用流程

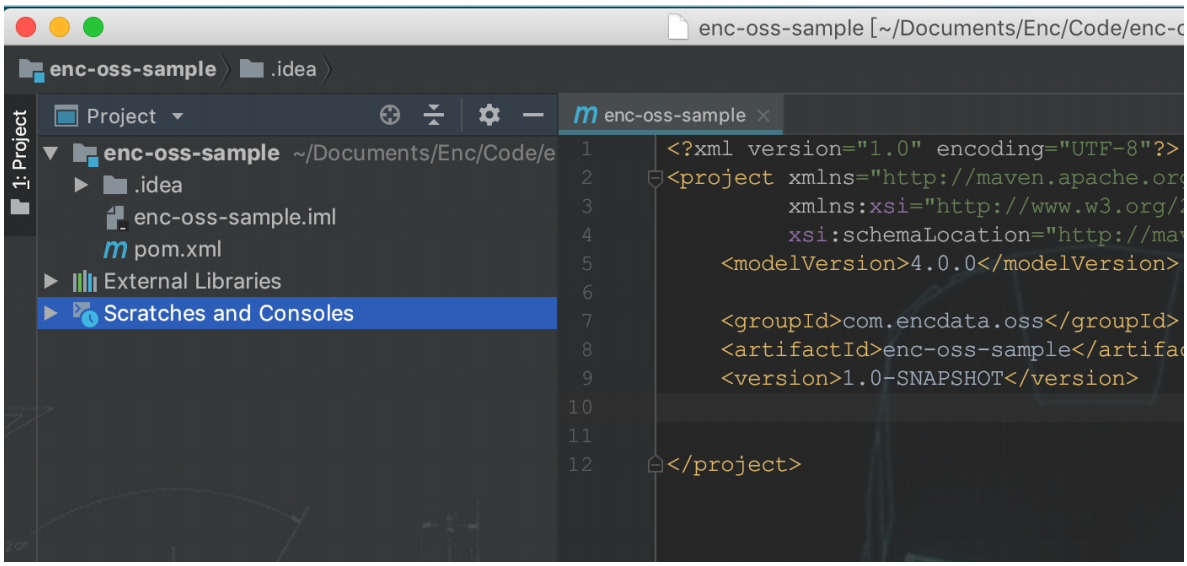
1. 环境搭建

1.1 创建主框架

1.1.1 在IDEA中选择一个Maven工程，并更改名称



1.1.2 创建后的文件目录结构



1.2 创建子项目

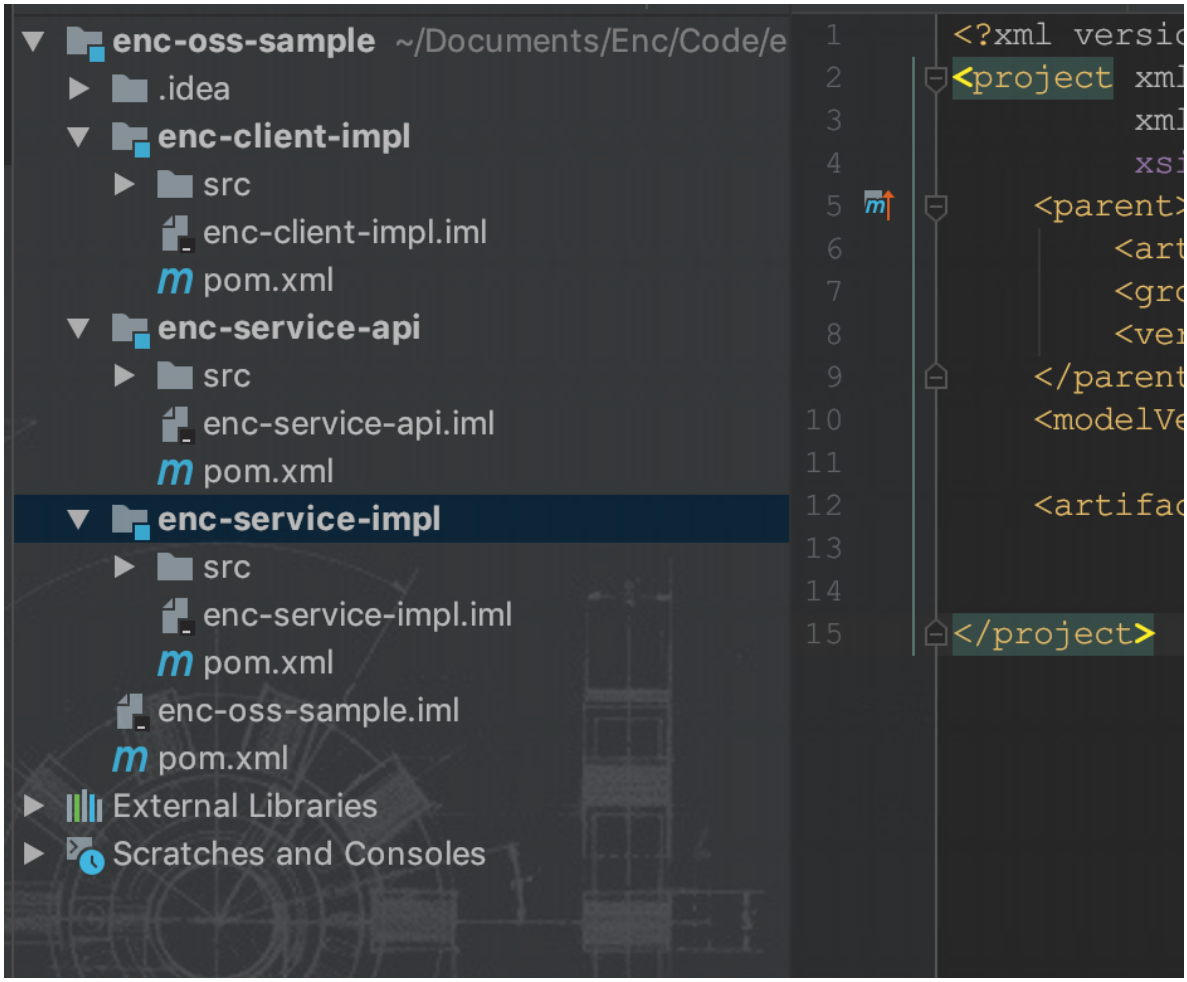
1.2.1 右键enc-oss-sample项目选择New->Module->Maven，在ArtifactId中输入enc-service-impl

1.2.2 右键enc-oss-sample项目选择New->Module->Maven，在ArtifactId中输入enc-client-impl

1.2.3 右键enc-oss-sample项目选择New->Module->Maven，在ArtifactId中输入enc-service-api

> 注意：
>
> 1. 正常项目只需service-impl和service-api两个项目，本次示例加入client方便调试和访问。
> 2. service-impl负责编写业务，以及各种功能的实现。
> 3. service-api为了将service的服务暴露给外部访问，其内部不要写入任何业务逻辑。

1.2.4 整体结构如下图



1.3 配置Maven

前置说明

> 注意： ***不要自行引入任何第三方依赖，如有需要可以联系管理员***

1.3.1 首先在enc-oss-sample的根pom文件中引入本次示例需要的依赖

```
    <parent>
        <groupId>com.encdata.oss</groupId>
        <artifactId>enc-oss-parent</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>enc-oss-sample</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>enc-service-impl</module>
        <module>enc-client-impl</module>
```

```
        <module>enc-service-api</module>
    </modules>
    <dependencies>
        <dependency>
            <groupId>com.encdata.oss</groupId>
            <artifactId>enc-oss-starter</artifactId>
        </dependency>
    </dependencies>
```

1.3.2 在enc-serivce-impl的pom中引入以下依赖

```
<dependencies>
        <dependency>
            <groupId>com.encdata.oss</groupId>
            <artifactId>enc-service-api</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

1.3.3 在enc-client-impl中引入以下依赖

```
<dependencies>
    <dependency>
        <groupId>com.encdata.oss</groupId>
        <artifactId>enc-service-api</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

2. 功能示例

前置说明

> 1. 下列项目需要先去申请APPINFO，将服务名称注册后才可以使用
> 2. 所有REST请求的返回统一使用R
> 3. 所有分页内容的返回统一使用P

2.1 实现RESTful API

2.1.1 在enc-client-impl右键->New->Package，输入com.encdata.oss

2.1.2 在底层oss文件夹右键->New -> Java Class，输入ClientApplication，并加入以下内容

```
@SpringBootApplication(scanBasePackages = {"com.encdata"})
@EnableDiscoveryClient
public class ClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class,args);
    }
}
```

2.1.3 在oss文件中右键->New->Package，输入client.controller.user

2.1.4 在user文件夹中右键->New -> Java Class，输入ClientController，并加入以下内容

```
@RestController
@RequestMapping("/client")
public class UserController {

    @GetMapping("/hello")
    public R userDemo() {
        return R.success("success");
    }
}
```

2.1.5 在main文件夹中的resources里，创建application.yml以及mybatis-config.xml

application.yml

```
server:
  port: 8100
spring:
  servlet:
    multipart:
      max-file-size: 100MB
      max-request-size: 100MB
  profiles:
    active: dev
  application:
    name: enc-provider
  redis:
    database: 0
    host: 10.19.146.53
    port: 6379
    timeout: 5000
  datasource:
    url: jdbc:mysql://10.19.151.143:3306/cityoss?
useUnicode=yes&characterEncoding=UTF-8
    username: bocom
    password: bocommysql
    driver-class-name: com.mysql.cj.jdbc.Driver
  cloud:
    consul:
      host: 10.19.146.53
      port: 8500
      discovery:
        hostname: 10.20.38.64
        health-check-critical-timeout: 30s
  kafka:
    bootstrap-servers: 10.19.146.53:9092
    producer:
```

```
      retries: 0
      batch-size: 16384
      buffer-memory: 33554432
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer

management:
  endpoints:
    web:
      exposure:
        include: "*"

mybatis:
  mapper-locations:
    - classpath*:/mapper/*-mapper.xml
  config-location: classpath:mybatis-config.xml
mapper:
    mappers: com.encdata.oss.starter.web.base.EncMapper
    not-empty: false
    identity: MYSQL

#pagehelper
pagehelper:
    helperDialect: mysql
    reasonable: true
    supportMethodsArguments: true
    params: count=countSql
    pageSizeZero: true
    rowBoundsWithCount: true
    offsetAsPageNum: true
#日志
logging:
  config: classpath:logback-${spring.profiles.active}.xml
info:
  security:
    enabled: false
    jwtSecret: encdata2019
    defaultExpiredDate: 1800
    whiteUrlList:
      - /user/login
fdfs:
  enabled: true
  # 读取时间
  soTimeout: 5000
  # 连接超时时间
  connectTimeout: 5000
  # 缩略图
  thumbImage:
    width: 150
    height: 150
  # tracker列表
  trackerList:
     - 139.224.114.222:22122
```

Mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <!-- 这个配置使全局的映射器启用或禁用缓存。|true,false|true -->
        <setting name="cacheEnabled" value="true" />
        <!-- 全局启用或禁用延迟加载。当禁用时,所有关联对象都会即时加载。|true,false|true
-->
        <!-- 通hessian集成时，反序列化会有问题 -->
        <setting name="lazyLoadingEnabled" value="false" />
        <!-- 当启用时,有延迟加载属性的对象在被调用时将会完全加载任意属性。否则，每种属性将会
按需要加载。|true,false|true -->
        <setting name="aggressiveLazyLoading" value="true" />
        <!-- 允许或不允许多种结果集从一个单独 的语句中返回(需要适合的驱
动)。|true,false|true -->
        <setting name="multipleResultSetsEnabled" value="true" />
        <!-- 使用列标签代替列名。不同的驱动在这 方便表现不同。参考驱动文档或充分测试两种方法
来决定所使用的驱动。|true,false|true -->
        <setting name="useColumnLabel" value="true" />
        <!-- 允许 JDBC 支持生成的键。需要适合的驱动。如果设置为 true 则这个设置强制生成的键
被使用,尽管一些驱动拒绝兼容但仍然有效(比如
            Derby)。|true,false|false -->
        <setting name="useGeneratedKeys" value="false" />
        <!-- 指定 MyBatis 如何自动映射列到字段/属性。PARTIAL只会自动映射简单，没有嵌套的
结果。FULL会自动映射任意复杂的结果(嵌套的或其他情况)。|NONE,
            PARTIAL, FULL| PARTIAL -->
        <setting name="autoMappingBehavior" value="PARTIAL" />
        <!-- 配置默认的执行器。SIMPLE 执行器没 有什么特别之处。REUSE 执行器重用 预处理语
句。BATCH 执行器重用语句和批量更新
            |SIMPLE, REUSE, BATCH|SIMPLE -->
        <setting name="defaultExecutorType" value="SIMPLE" />
        <!-- 设置超时时间,它决定驱动等待一个数据库响应的时间。| Any positive integer
|Not Set (null) -->
        <setting name="defaultStatementTimeout" value="25000" />
        <!-- 记录器的名称前缀 -->
        <setting name="logPrefix" value="dao."/>
    </settings>
    <typeAliases>
        <package name="com.encdata.oss.provider.dal.user.model" />
    </typeAliases>
</configuration>
```

2.1.6 启动ClientApplication，在浏览器中访问 http://127.0.0.1:8100/client/hello ，可以看到json格式的返回值

```
{"resultCode":0,"resultMessage":"success"}
```

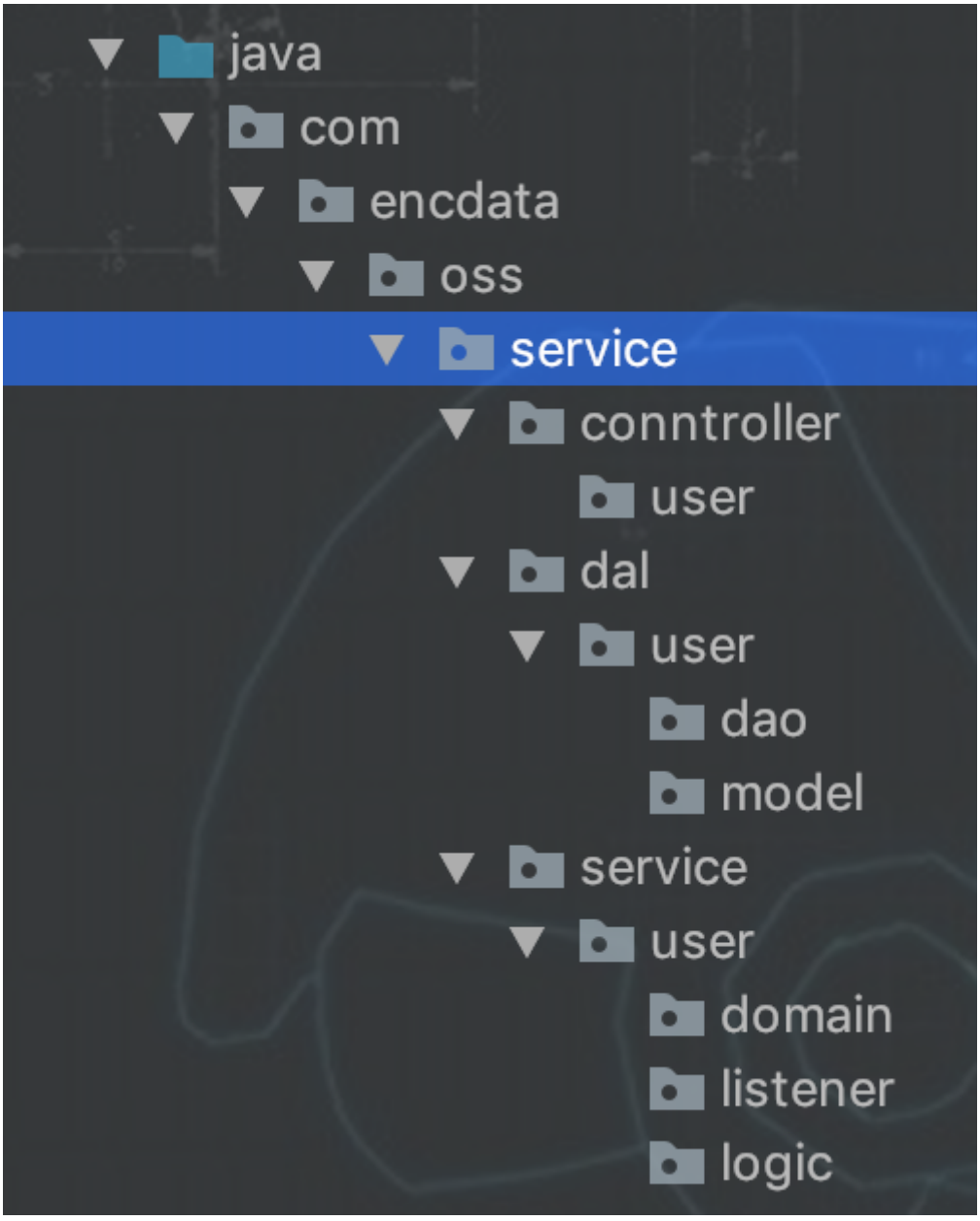
2.2 使用Mybatis

前置说明

> 因为使用了tkmybatis框架，所以示例中未创建mapper.xml文件，对于单表操作直接使用内置的方法即可， ***不需要生成任何映射mapper.xml文件*** ，其他的多表关联或复杂查询仍可以使用之前的编写Mapper.xml的方式查询。

2.2.1 在enc-service-impl中

- 在oss下创建如下目录



2.2.2 在oss中创建ServiceApplication类，并添加以下内容

> 注意：确保MapperScan的value能够扫描到指定的dao

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableTransactionManagement
@MapperScan(value = "com.encdata.oss.service.dal.user.dao",annotationClass =
Repository.class)
public class ServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceApplication.class,args);
    }
}
```

2.2.3 复制enc-client-impl中导入的application.yml和mybatis-config.xml到resources目录中

将端口改为8200

2.2.4 在controller.user中添加UserController

```
@Slf4j
@RestController
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserServiceBiz userService;

    @GetMapping("/selectAll")
    public R<List> selectOne(){
        List<Users> user = userService.selectAll(1,10);
        return R.success(user);
    }
}
```

2.2.5 在dal.user.dao中添加UserMapper

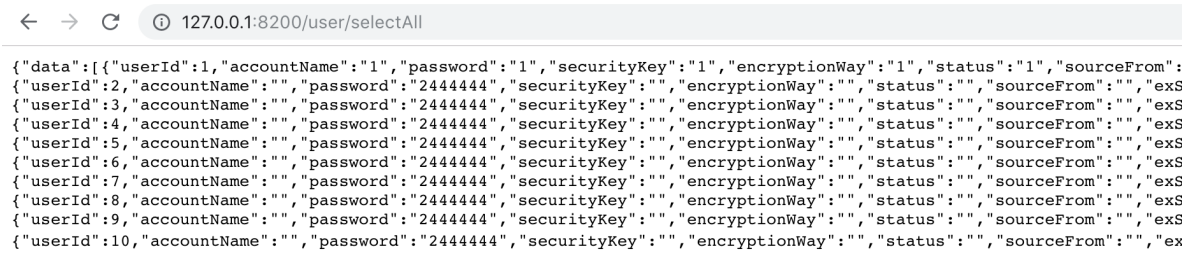
```
@Repository
public interface UserMapper extends EncMapper<Users> {
}
```

2.2.6 在dal.user.model添加Users（依据具体数据表内容）

2.2.7 在service.user.logic中添加UserServiceBiz

```
@Service
@Slf4j
public class UserServiceBiz {
    @Autowired
    private UserMapper userDAL;
    /**
     * 查询全部数据
     */
    public List<Users> selectAll(Integer page, Integer rows){
        PageHelper.startPage(page,rows);
        List<Users> list = userDAL.selectAll();
        return list;
    }
}
```

2.2.8 启动ServiceApplication，在浏览器中访问 **http://127.0.0.1:8200/user/selectAll**



2.3 使用Redis

> 示例中使用spring-boot-starter-data-redis依赖

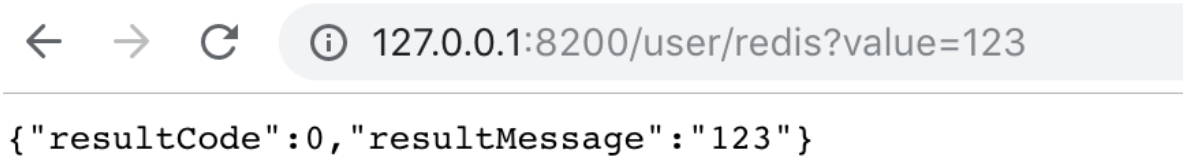
2.3.1 查看application.yml是否已经配置了redis的IP和端口

2.3.1 在enc-service-impl中的UserController里添加代码

```
@Autowired
private EncRedisUtils encRedisUtils;

@RequestMapping("/redis")
    public R redis(String value){
        encRedisUtils.append("oss-sample",value);
        String result = encRedisUtils.get("oss-sample");
        return R.success(result);
    }
```

2.3.2 启动ServiceApplication，在浏览器中访问 http://127.0.0.1:8200/user/redis?value=123



{"resultCode":0,"resultMessage":"123"}

2.4 使用Kafka

2.4.1 查看application.yml是否已经配置了kafka的IP和端口

2.4.2 在enc-service-impl的UserController里添加

```
@Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @RequestMapping("kafka")
    public R kafka(String value){
        kafkaTemplate.send("enc.service", ""+value);
        return R.success();
    }
```

2.4.3 启动ServiceApplication，在浏览器中访问 http://127.0.0.1:8200/user/kafka?value=123 可以发送数据

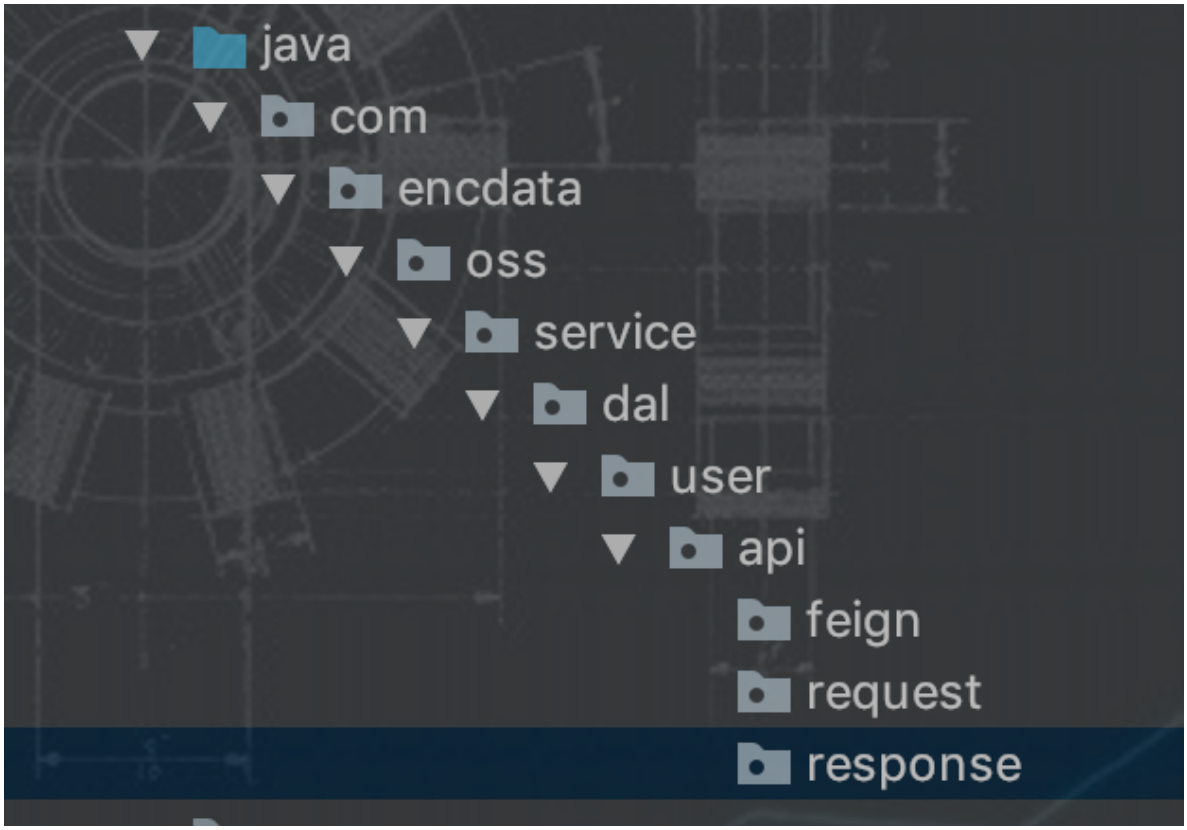
2.4.4 在enc-service-impl的UserServiceBiz中添加

```
// 配置监听的主体，groupId 和配置文件中的保持一致
@KafkaListener(topics = { "enc.service" })
public void listen(ConsumerRecord<String, String> record) {
    Optional<?> kafkaMessage = Optional.ofNullable(record.value());
        System.out.println("这是接收到的数据------"+record.value());
}
```

2.4.5 重新启动ServiceApplication，在浏览器中访问<http://127.0.0.1:8200/user/kafka?value=123，看到控制台里打印出消费到的数据。

2.5 使用feign

2.5.1 在enc-service-api中建立如下包结构



2.5.2 在feign中新建IUserService接口

> 注意：
>
> 1. contextid为了保证唯一性，使用接口名称的小写
> 2. AppInfo.APPLICATION_SAMPLE_SERVICE需要先申请方可使用

```
@FeignClient(contextId = "userService",value =
AppInfo.APPLICATION_SAMPLE_SERVICE, path = "/user")
public interface IUserService {
    @GetMapping("/searchDetailUserInfo")
    R<SearchDetailUserInfoResponse>
searchDetailUserInfo(SearchDetailUserInfoRequest searchDetailUserInfoRequest);
}
```

2.5.3 在request中新建SearchDetailUserInfoRequest

```
public class SearchDetailUserInfoRequest extends BaseRequest {

}
```

2.5.4 在response中新建SearchDetailUserInfoResponse

```
@Data
@ApiModel(description = "用户信息")
public class SearchDetailUserInfoResponse extends BaseResponse {
    /**
     * 邮箱
     */
    @ApiModelProperty(value = "邮箱")
    @Email
    private String email;
    /**
     * 手机
     */
    @ApiModelProperty(value = "手机")
    @Mobile
    private String phone;
}
```

2.5.5 在enc-service-impl的UserController中继承IUserService接口，并加入

```
@RequestMapping("/searchDetailUserInfo")
@Override
public R<SearchDetailUserInfoResponse>
searchDetailUserInfo(SearchDetailUserInfoRequest searchDetailUserInfoRequest) {
    SearchDetailUserInfoResponse search = new SearchDetailUserInfoResponse();
    search.setEmail("xxx@qq.com");
    search.setPhone("10000");
    return R.success(search);
}
```

2.5.6 查看enc-service-impl的resources里的applicationn.yml文件，修改application.name

```
application:
  name: enc-service-impl
cloud:
    consul:
      host: 10.19.146.53
      port: 8500
      discovery:
      #这里修改为本地IP地址
        hostname: 10.20.55.29
        health-check-critical-timeout: 30s
```

2.5.7 在enc-client-impl中的com.encdata.oss.client.service.user.logic中新建UserServiceBiz

```
@Service
@Slf4j
public class UserServiceBiz {
    @Autowired
    private IUserService userService
    public R<SearchDetailUserInfoResponse> userDetail() {
        log.info("fegin远程调用--------------------------");
        R<SearchDetailUserInfoResponse> responseR=
userService.searchDetailUserInfo(new SearchDetailUserInfoRequest());
        return responseR;
    }
}
```

2.5.8 在enc-client-impl的UserController添加

```
@RestController
@RequestMapping("/client")
public class UserController {
    @Autowired
    private UserServiceBiz userDetailBiz;
    @RequestMapping("/hello")
    public R<SearchDetailUserInfoResponse> userDemo() {
        System.out.println("session的ID是-->" + SecureUtil.getUserLoginName());
        return userDetailBiz.userDetail();
    }
}
```

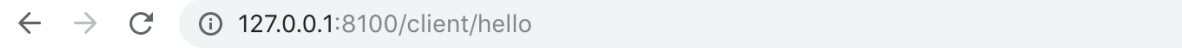
2.5.9 在enc-client-impl中的ClientApplication里添加注解

```
// 这里的注解位置必须能够扫描到定义Fegin注解的IUserService的位置
@EnableFeignClients(basePackages = {"com.encdata"})
```

2.5.10 查看enc-client-impl的resources里的applicationn.yml文件，修改consul本机IP

```
cloud:
  consul:
    host: 10.19.146.53
    port: 8500
    discovery:
    #这里修改为本地IP地址
      hostname: 10.20.55.29
      health-check-critical-timeout: 30s
```

2.5.11 启动enc-service-impl和enc-client-impl，在浏览器中访问 http://127.0.0.1:8100/client/hello



{"data":{"email":"xxx@qq.com","phone":"10000"},"resultCode":0,"resultMessage":"成功"}

可以看到预先设定的值

2.6 使用BeanMapper

2.6.1 在enc-service-impl中新建两个model，Users和User，User中只包含一部分Users的属性

```
Users users = new Users();
users.setPassword("324");
User user = BeanMapper.map(users, User.class)
```

> 通过map方法可以将bean之间互相转换
>
> 1. mapToBean（将map装换为javabean对象）
> 2. beanToMap（对象封装成map）
> 3. mapsToObjects（将List<Map<String,Object>>转换为List）
> 4. objectsToMaps（将List转换为List<Map<String, Object>>）

2.7 使用P

2.7.1 P中包含以下属性

```
/**
   * 当前页码
   */
  private int pageIndex;
  /**
   * 单页面记录总数
   */
  private int pageSize;
  /**
   * 总页数
   */
  private int pageTotal;
  /**
   * 总记录数
   */
  private long recordTotal;
  /**
   * 数据
   */
  private T data;
```

2.7.2 P中包含3个构造参数

> 1. P(Page page)
> 2. P(int pageIndex, int pageSize, int pageTotal, int recordTotal, T t)记录数数据
> 3. P(int recordTotal, T t)只返回记录数和数据

2.7.3 P中包含4个valueOf方法

> 1. P valueOf(int pageIndex, int pageSize, int pageTotal, int recordTotal, T t) 包裹分页总数和分页数据
> 2. P valueOf(int recordTotal, T t)包裹分页总数和分页数据
> 3. P valueOf(Page page)
> 4. <T, F> P valueOf(PageInfo pageInfo)

3. 常用工具

> enc-oss-util中包含了大部分工具类
>
> pom中已经添加Hutool的依赖，可以使用Hutool提供的其他方法，参考文档 https://hutool.cn/docs/#/

3.1 集合

名称	功能	类型
Collections3	提供对集合的操作	集合

3.2 时间

名称	功能	类型
ConcurrentDateFormat	不使用ThreadLocal，创建足够的SimpleDateFormat对象来满足并发性要求	时间
DateUtil	提供多种日期，时间的格式化操作	时间

3.3 加密

名称	功能	类型
AesUtil	AES加密解密	
Base64Util	Base64加密解密	
DESUtil	DES加密解密	
DigestUtil	加密工具类	

3.4 异常

名称	功能	类型
ExceptionUtil	处理异常	

3.5 文件

名称	功能	类型
FileConstant	文件Model	
FileProxyFactory	文件代理类	
FileProxyManager	文件管理类	
FileUtil	文件工具类	
FileWrap	上传文件封装	
IFileProxy	文件代理接口	

3.6 JSON

名称	功能	类型
AbstractReadWriteJackson2HttpMessageConverter	分读写的 json 消息 处理器	
BeanSerializerModifier	jackson 默认值为 null 时的处理	
JsonUtil	Jackson工具类	
MappingApiJackson2HttpMessageConverter	针对 api 服务对 android 和 ios 和 web 处理的 分读写的 jackson 处理	

3.7 邮件

名称	功能	类型
MailAddress	邮箱地址名称	
MailMessageObject	邮件信息内容	
MailUtil	邮件发送工具实现类	

3.8 Mapper

名称	功能	类型
BeanMapper	简单封装Dozer, 实现深度转换Bean<->Bean的Mapper.实现	
JaxbMapper	使用Jaxb2.0实现XML<->Java Object的Mapper	
JsonMapper	简单封装Jackson，实现JSON String<->Java Object的Mapper	

3.9 Net

名称	功能	类型
AppNameUtil		
HostNameUtil	获取IP和HOST	
OkHttpUtil	Http请求工具类	
PathUtil	用来获取各种目录	
RestTemplateUtil	构建多种请求方式	
UrlUtil	url处理工具类	
WebUtil	处理Web请求	

3.10 数字

名称	功能	类型
NumberUtil	数字类型工具类	
RandomType	生成的随机数类型	

3.11 工具

名称	功能	类型
BeanUtil	实体工具类	
ClassUtil	类工具类	
CollectionUtil	集合工具类	
ObjectUtil	对象工具类	
SpringUtil	spring 工具类	
ReflectionUtil	反射工具类	

3.12 REST

名称	功能	类型
RequestUtil	request工具类	
RestClient	处理REST请求	
ResultUtil	对返回给前端的数据进行格式封装处理	

3.13 stream

名称	功能	类型
ImageUtil	图片工具类	
IoUtil	IO转换工具	
ResourceUtil	资源工具类	
SuffixFileFilter	文件后缀过滤器	

3.14 string

名称	功能	类型
CharsetUtil	字符集工具类	
StringFormatter	字符串格式化	
StringPool	静态 String 池	
StringSpliter	字符串切分器	
StringUtil	继承自Spring util的工具类，减少jar依赖	

3.15 Xss

名称	功能	类型
HtmlFilter	HTML过滤	
SqlFilter	SQL过滤	
XssFilter	XSS过滤	
XssHttpServletRequestWrapper	XSS过滤处理	

3.16 support

名称	功能	类型
BaseBeanCopier	spring cglib 魔改	
BeanProperty	Bean属性	
FastStringWriter	快速构建字符串	
ImagePosition	图片操作类	
IMultiOutputStream	用于创建MultiOutputStream对象	

3.17 Tree

名称	功能	类型
Tree	构造树节点的接口规范	
TreeNode	tree节点参数的封装	
TreeUitl	默认递归工具类，用于遍历有父子关系的节点，例如菜单树，字典树等等	

3.18 其他

名称	功能	类型
XmlUtil	xpath解析xml	
ENCUtils	工具包集合，只做简单的调用，不删除原有工具类	
R	统一API响应结果封装	
P	统一分页的内容管理	

4. 环境启动

4.1 本地启动

> 分别启动对应项目的Application即可

4.2 服务器启动

4.2.1 先将各项目打包，在IDEA右侧Maven Projects，点击对应项目的Lifecycle的install，打包成功后可以看到项目中生成target文件夹

4.2.2 将jar包上传至服务器

4.2.3 启动jar包 Java -jar xxx.jar

4.2.4 启动jar包并打印日志到指定位置

```
nohup java -jar xxxx.jar >> logs/xxxx.log 2>&1 &
```

4.2.5 使用外置application.yml启动项目

```
nohup java -Dspring.config.location=properties/application-adaptor.yml -jar enc-oss-adaptor.jar >> logs/gateway.log 2>&1 &
```

##