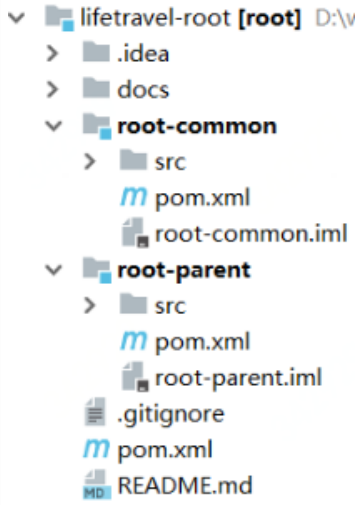
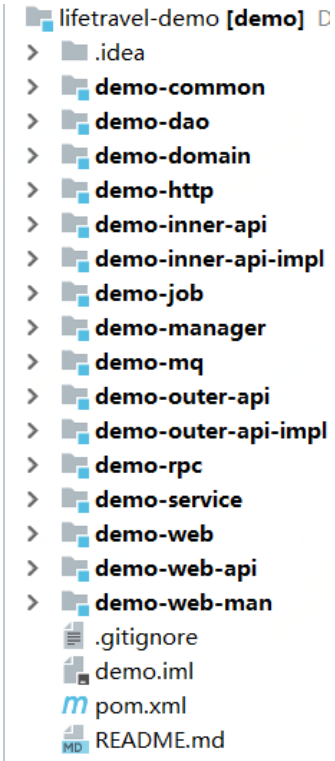
03_工程结构规范

工程结构

工程	截图	描述
lifetravel-root		platform工程目前包含两个module： 1、common：会发布jar包，所有项目的common module都要引用这个jar，如下： <dependency> <groupId>com.jd.lifetravel</groupId> <artifactId>root-common</artifactId> <version>1.0.0-SNAPSHOT</version> </dependency> 2、parent：会发布pom，所有的项目的项层pom都要引用这个pom，如下

		<dependencyManagement> <dependencies> <dependency> <groupId>com.jd.lifetravel</groupId> <artifactId>root-parent</artifactId> <version>1.0.1-SNAPSHOT</version> <type>pom</type> <scope>import</scope> </dependency> </dependencies> </dependencyManagement>	
lifetravel-demo		样例工程，所有的项目都要参考这个工程来创建自己的工程 **module说明：** 1. common：common引用root-common	



2. dao：数据库操作

3. domain：基础实体类
   |--domain
   |--dao：存放DO结尾的数据库实体类，如MonitorLogDO.java
   |--bo：存放BO结尾的业务实体类，如OrderBO.java
   |--enums：存放全局的枚举类。


```
<parent>
  <groupId>com.jd.trip.hotel</groupId>
  <artifactId>trip-hotel-demo</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</parent>

<artifactId>demo-inner-api</artifactId>
<packaging>jar</packaging>

<name>demo-inner-api</name>
<url>http://hotel.jd.com</url>

<dependencies>
  <dependency>
    <groupId>com.jd.trip.hotel</groupId>
    <artifactId>demo-service</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>
</dependencies>
```

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.jd.trip.hotel</groupId>
      <artifactId>hotel-platform-parent</artifact
      <version>1.0.2-SNAPSHOT</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

4. service：基础逻辑处理与公共逻辑处理
5. mq：消息处理
6. job：定时任务
7. rpc：外部或二方服务调用
8. http：外部或二方服务调用
9. inner-api：封装内部JSF接口，给内部服务调用。

inner-api包目录|---|---**domain**

```
|--- dto：DTO实体类
|--- query：用于封装请求参数的实体类
|--- enums：枚举类
|--- constant：常量类
|--- service：服务接口
```

10. inner-api-impl：对inner-api的具体实现
11. outer-api：封装内部JSF接口，给网关或外部系统调用。

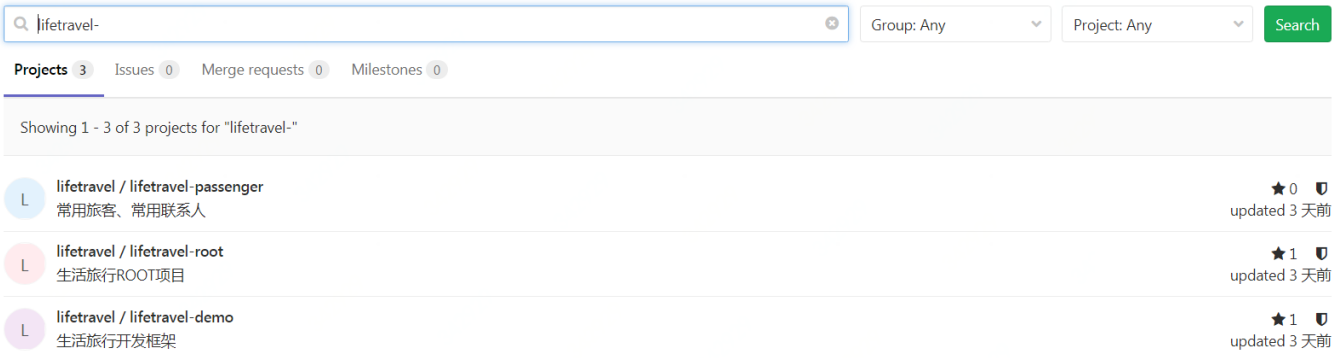
outer-api包目录
|--|--|--|--dto：DTO实体类
|--|--|--|--query：用于封装请求参数的实体类
|-

		12.	-|--|--|--enums:枚举类|--|--|--|--constant:常量类|--|--**service**:服务接口outer-api-impl:对outer-api的具体实现

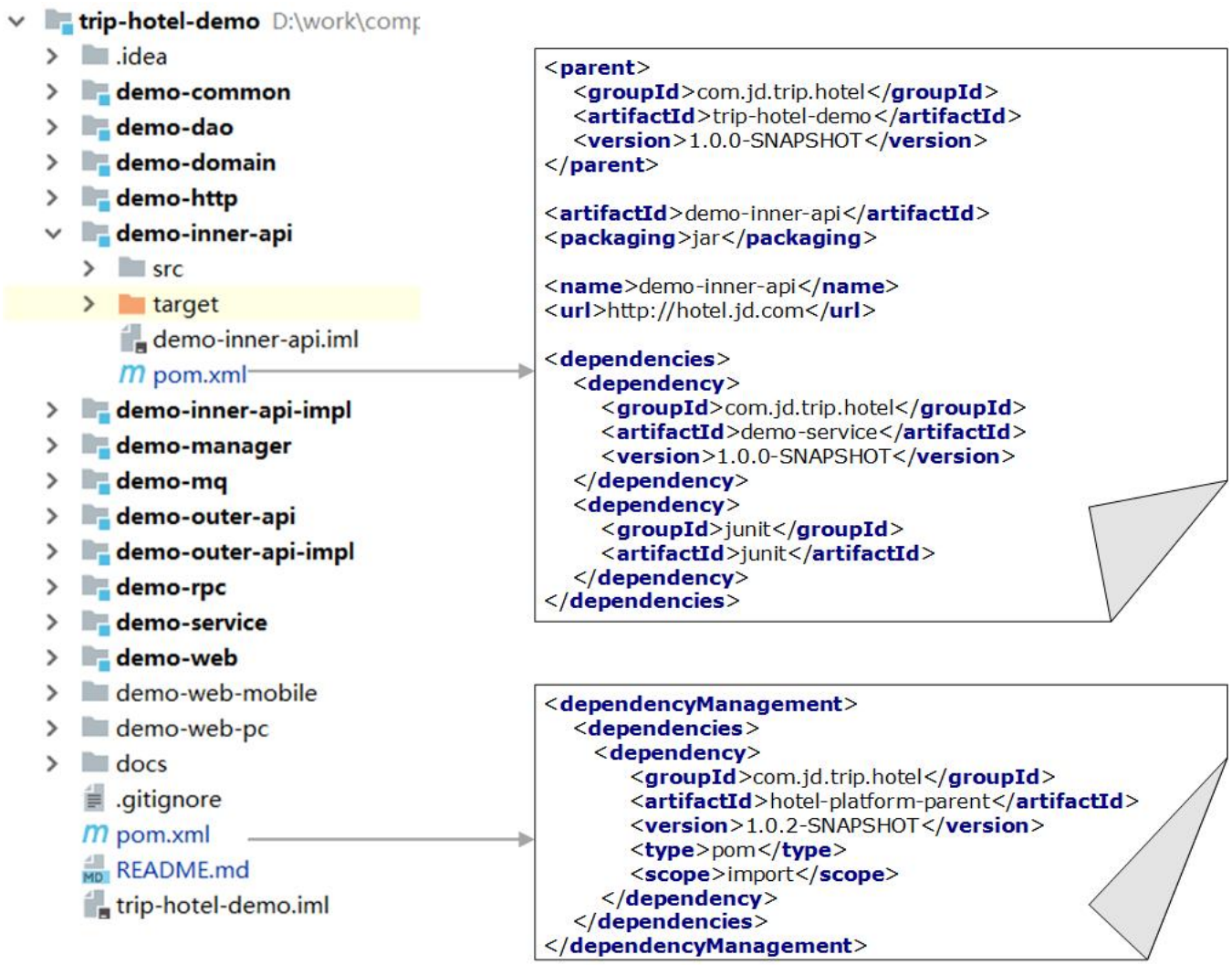
工程命名规范

所有工程都要以“业务名-”作为前缀，如lifetravel-root、lifetravel-passenger

访问git.jd.com，搜索lifetravel-可以查到所有公共平台的项目工程，如下图所示：



Maven管理



【强制】严格按照上面的结构进行pom依赖引用

【强制】线上应用不要依赖SNAPSHOT版本（安全包除外）。 说明：不依赖SNAPSHOT版本是保证应用发布的幂等性。另外，也可以加快编译时的打包构建。

【强制】依赖于一个二方库、三方库群时，必须定义一个统一的版本变量，避免版本号不一致。说明：依赖springframework-core,-context,-beans，它们都是同一个版本，可以定义一个变量来保存版本：${spring.version}，定义依赖的时候，引用该版本。

【强制】禁止在子项目的pom依赖中出现相同的GroupId，相同的ArtifactId，但是不同的Version。 说明：在本地调试时会使用各子项目指定的版本号，但是合并成一个war，只能有一个版本号出现在最后的lib目录中。可能出现线下调试是正确的，发布到线上却出故障的问题。

【强制】二方库的新增或升级，保持除功能点之外的其它jar包仲裁结果不变。如果有改变，必须明确评估和验证，进行dependency:resolve前后信息比对，如果仲裁结果完全不一致，那么通过dependency:tree命令，找出差异点，进行<excludes>排除jar包。

代码实体类定义规范

DO（Data Object）	与数据库表结构一一对应，通过DAO层向上传输数据源对象
DTO（Data Transfer Object）	数据传输对象，Service和Manager向外传输的对象
BO（Business Object）	业务对象。可以由Service层输出的封装业务逻辑的对象
Query	数据查询对象，各层接收上层的查询请求。注：超过2个参数的查询封装，禁止使用Map类来传输

VO（View Object）	显示层对象，通常是Web向模板渲染引擎层传输的对象

1）结果实体类：

（1）service层以BO结尾

（2）mananger层DO结尾

（3）dao层以DO结尾

（4）jsf传输层以DTO结尾

（5）controller层传递给页面以VO结尾

2）请求实体类：

上层向下层提交请求的时候，如果没有封装成对象，则用Param结尾的实体，负责为BO或DO，具体如下：

写请求：

（1）controller层Param结尾

（2）service层BO结尾

（3）mananger层DO结尾

（4）dao层DO结尾

查请求：

（1）controller层Param结尾

（2）service层Query结尾

（3）mananger层Query结尾

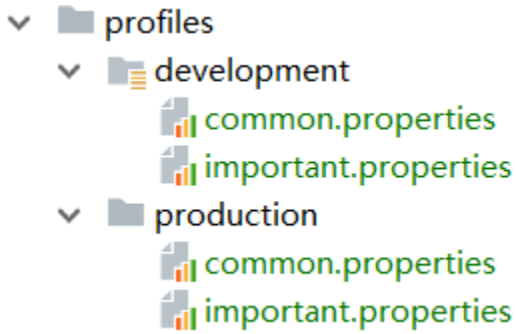
（4）dao层Query结尾

实体类转换工具BeanUtils.copyProperties

```
public OrderBO getById(Integer id) {
    OrderDO orderDO = orderManager.getById(id);
    if (orderDO == null) {
        return null;
    }
    OrderBO orderBO = new OrderBO();
    BeanUtils.copyProperties(orderDO, orderBO);
    jimCache.set("trip::hotel::tianziyu", "");
    logger.info(":" + jimCache.get("trip::hotel::tianziyu"));
    return orderBO;
}
```

```
public class BeanUtils extends org.springframework.beans.BeanUtils {

    public static void copyProperties(Object source, Object target) throws BeansException {
        copyProperties(source, target, null, (String[]) null);
    }


    private static void copyProperties(Object source, Object target, Class<?> editable, String...
ignoreProperties)
            throws BeansException {

        Assert.notNull(source, "Source must not be null");
        Assert.notNull(target, "Target must not be null");

        Class<?> actualEditable = target.getClass();
        if (editable != null) {
            if (!editable.isInstance(target)) {
                throw new IllegalArgumentException("Target class [" + target.getClass().getName() +
                        "] not assignable to Editable class [" + editable.getName() + "]");
            }
            actualEditable = editable;
        }
        PropertyDescriptor[] targetPds = getPropertyDescriptors(actualEditable);
        List<String> ignoreList = (ignoreProperties != null) ? Arrays.asList(ignoreProperties) : null;

        for (PropertyDescriptor targetPd : targetPds) {
            Method writeMethod = targetPd.getWriteMethod();
            if (writeMethod != null && (ignoreProperties == null || (!ignoreList.contains(targetPd.getName()))))
{
                PropertyDescriptor sourcePd = getPropertyDescriptor(source.getClass(), targetPd.getName());
                if (sourcePd != null) {
                    Method readMethod = sourcePd.getReadMethod();
                    if (readMethod != null &&
                            ClassUtils.isAssignable(writeMethod.getParameterTypes()[0], readMethod.
getReturnType())) {
                        try {
                            if (!Modifier.isPublic(readMethod.getDeclaringClass().getModifiers())) {
                                readMethod.setAccessible(true);
                            }
                            Object value = readMethod.invoke(source);
                            if (value != null) {
                                if (!Modifier.isPublic(writeMethod.getDeclaringClass().getModifiers())) {
                                    writeMethod.setAccessible(true);
                                }
                                writeMethod.invoke(target, value);
                            }
                        }
                        catch (Exception ex) {
                            throw new FatalBeanException("Could not copy property '" + targetPd.getName() + "'
from source to target", ex);
                        }
                    }
                }
            }
        }
    }
}
```

配置规范



这里没有增加config.properties文件是因为所有运行态的配置信息统一放到配置中心中。

把启动态的配置文件用properties表示而不是写在pom中的原因如下

1、 写在目录资源文件中，可以通过maven type war继承的方式继承所有properties文件

2、 将信息配置在properties文件中，就可以对properties进行加密