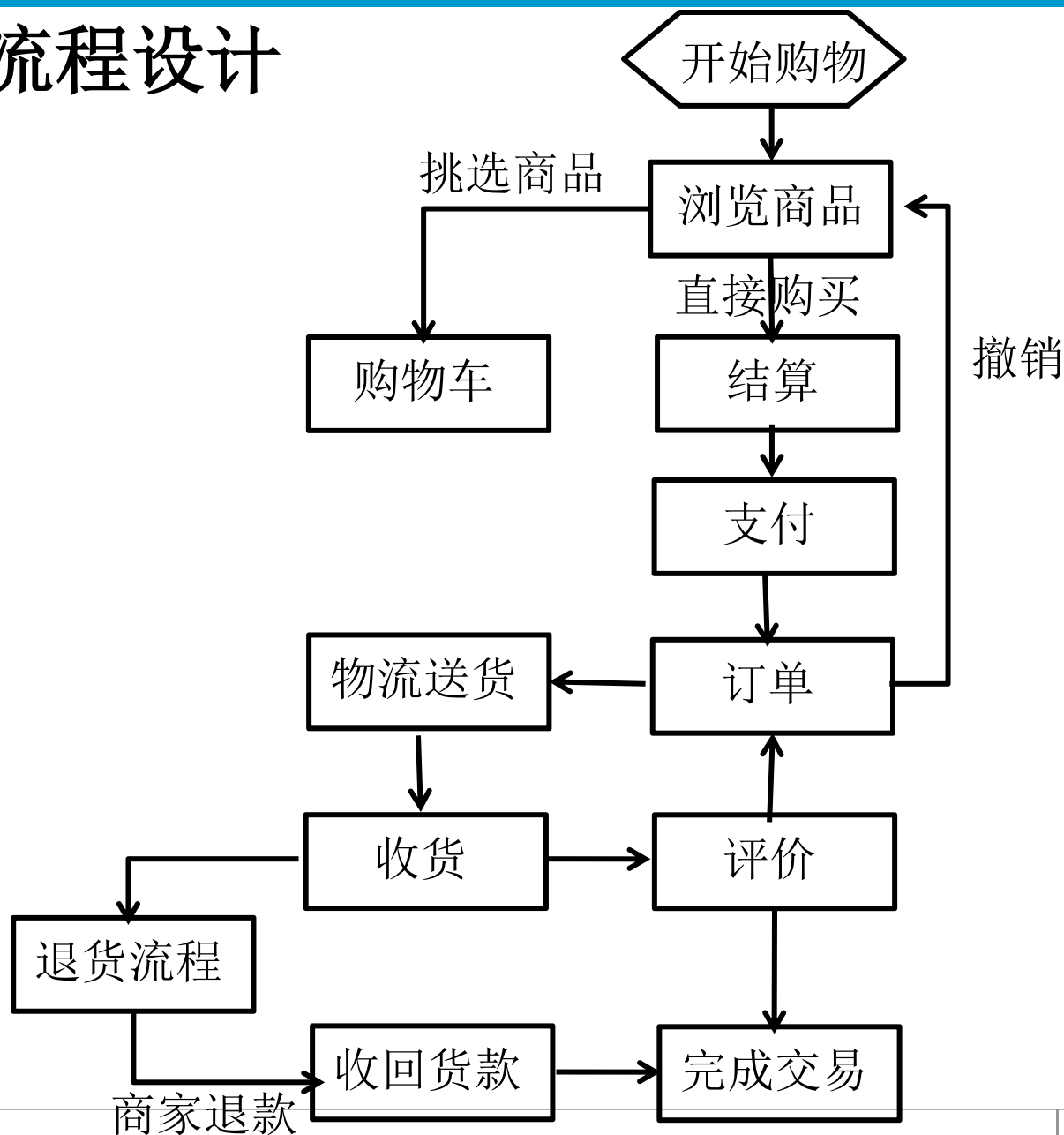


# 微服务架构设计实践

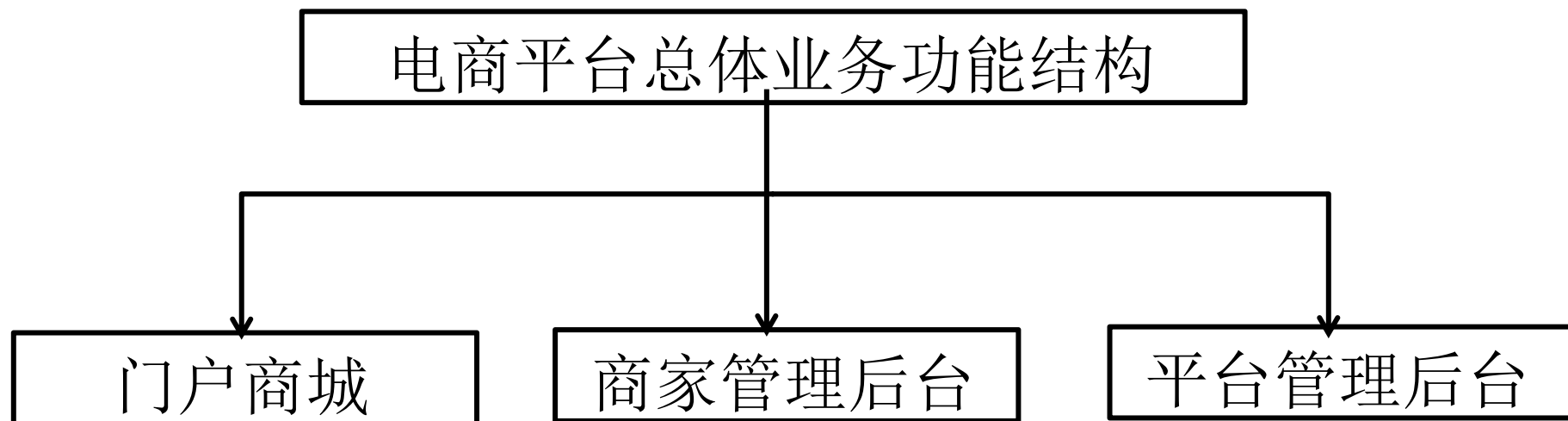
# 微服务架构设计案例

## 电商平台微服务设计实例

# 总体业务流程设计

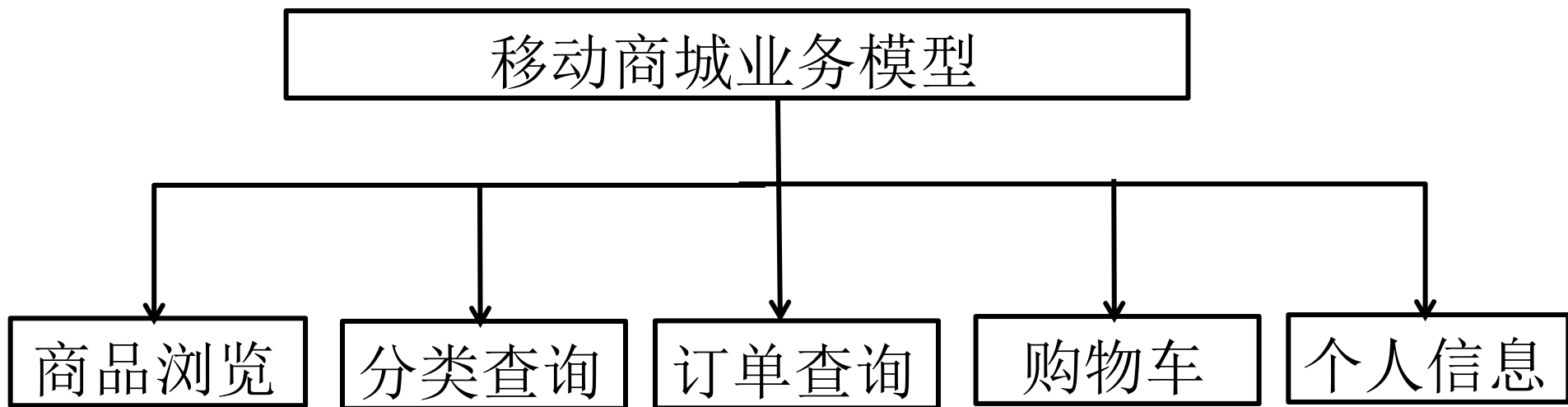


# 总体业务功能设计

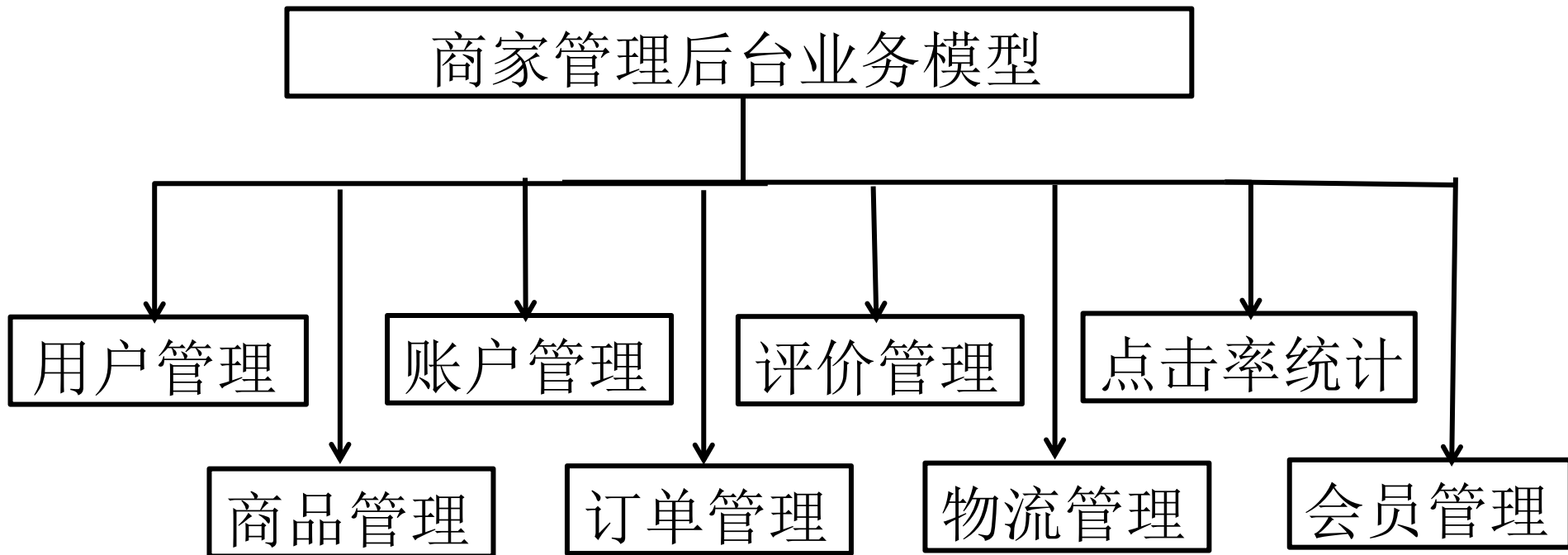


# 电商平台业务模型设计

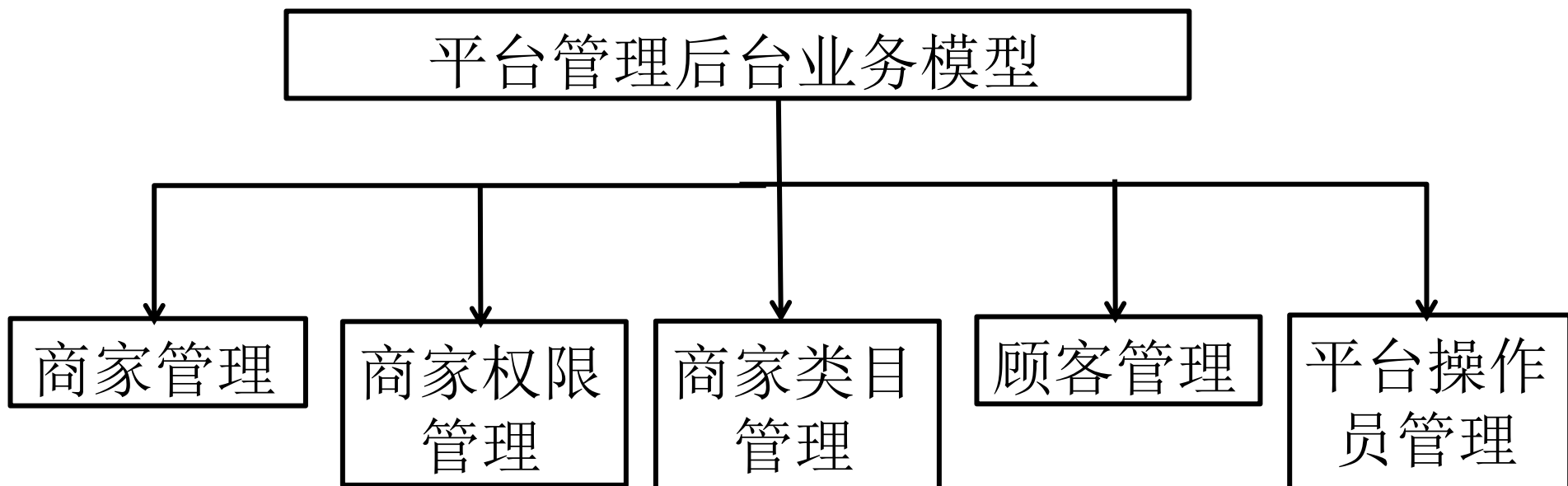
- 移动商城业务模型
- 商家管理后台业务模型
- 平台管理后台业务模型



# 商家管理后台业务模型



# 平台管理后台业务模型



# 微服务划分

- 在水平方向上，按业务功能不同来划分微服务，并把这次划分所创建的微服务称为**Rest API**微服务。**Rest API**微服务负责业务功能的行为设计，主要完成数据管理方面的工作，并通过使用**Rest**协议对外提供接口服务。
- 在垂直方向上，再以**Rest API**微服务为基础实现前后端分离设计，创建**Web UI**微服务。**Web UI**微服务不直接访问数据，而只专注于人机交互界面的设计，它的数据存取将通过调用**Rest API**微服务来完成。

只要使用两种类型的微服务，就可以构成一个复杂的业务系统。



## 创建**RestAPI**微服务，水平划分

- 类目服务
- 商品服务
- 购物车服务
- 订单服务
- 支付服务
- 物流服务
- 评价服务
- 顾客服务
- 会员服务
- 点击率服务
- 商家服务

**RestAPI**微服务使用实体对象进行数据的存取操作，然后对外提供基于**HTTP**协议的**Restful**接口服务

## 创建Web UI微服务，垂直划分

根据每个RestAPI微服务来实现前后端分离设计，从移动商城，商家管理后台和平台管理后台三个方面创建Web UI微服务

### 商家管理后台Web UI微服务

- 用户管理
- 商品管理
- 订单管理
- 物流管理
- 评价查询
- 账户管理
- 会员管理
- 点击率统计

### 移动商城Web UI微服务

- 分类查询
- 商品查询
- 购物车管理
- 订单查询
- 物流跟踪
- 个人信息管理
- 会员卡管理

### 平台管理后台Web UI微服务

- 本地用户管理
- 商家管理
- 商家权限管理

# IDEA中开发微服务项目

Rest API微服务

Web UI微服务

Web UI微服务-->Rest API微服务（用户服务）-->数据库

开发模式

一个工程一个服务

myshop-user

module

pom.xml

myshop-web

module

pom.xml

一个工程多个服务

myshop工程

myshop-user

myshop-web

pom.xml

# IDEA中开发RestAPI微服务

Rest API微服务需要实现两方面的功能：一.对内实现数据管理的功能；二.对外提供API调用

## 搭建Spring Boot-RestAPI微服务 myshop-user用户微服务

- 1.建立Maven项目
- 2.导入SpringBoot依赖
- 3.编写application.yml 或 application.properties
- 4.编写启动类
- 5.编写UserController

采用**Maven**多模块化方式搭建项目！（中大型项目推荐）

## 1. 建父工程，导入相关依赖

myshop-parent > pom.xml

当前文件

myshop-parent

.idea

.mvn

.gitignore

HELP.md

mvnw

mvnw.cmd

pom.xml

外部库

临时文件和控制台

pom.xml (myshop-parent)

```
<artifactId>myshop-parent</artifactId>
```

```
<version>0.0.1-SNAPSHOT</version>
```

```
<name>myshop-parent</name>
```

```
<description>myshop-parent</description>
```

```
<!--SpringBoot项目必须导入的parent依赖-->
```

```
<parent>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-parent</artifactId>
```

```
<version>2.7.16</version>
```

```
<relativePath/> <!-- lookup parent from repository -->
```

```
</parent>
```

```
<!--定义常用的属性-->
```

```
<properties>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
```

```
<java.version>1.8</java.version>
```

```
</properties>
```

```
<dependencies>
```

```
<!--SpringBoot 对web支持 SpringMVC 相关功能, json转换功能等等-->
```

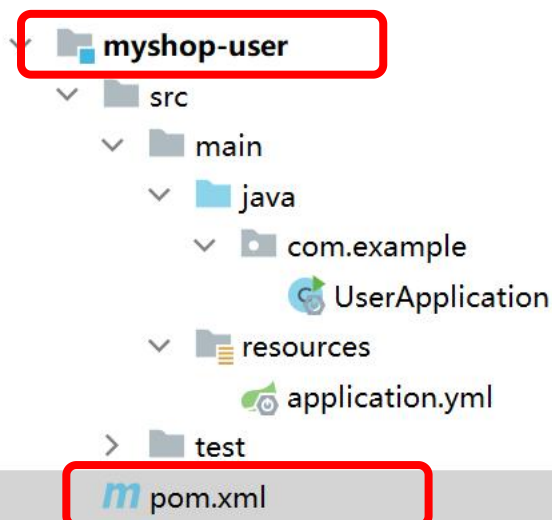
```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

## 2. 建子工程 RestAPI服务，导入依赖



```
xsi:schemaLocation="http://maven.apache
<modelVersion>4.0.0</modelVersion>
<parent>
  <groupId>com.example</groupId>
  <artifactId>myshop-parent</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</parent>

<artifactId>myshop-user</artifactId>

<properties>
  <maven.compiler.source>8</maven.compiler
```

## 3.编写配置文件application.yml

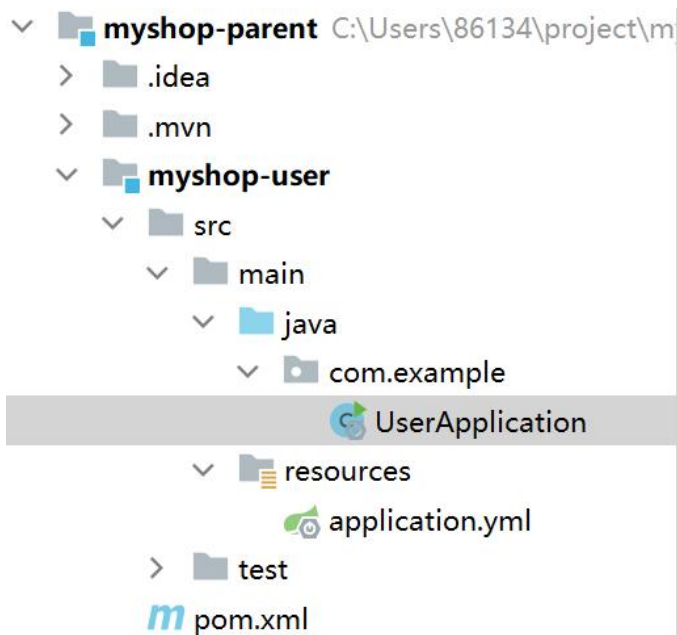


The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The project structure on the left includes a folder named **myshop-parent** (highlighted with a red box) which contains subfolders **.idea**, **.mvn**, and **myshop-user**. The **myshop-user** folder contains **src**, **main**, **java**, **com.example**, **UserApplication**, and **resources**. The **resources** folder contains the file **application.yml** (also highlighted with a red box). The code editor on the right shows the content of **application.yml** with line numbers 1 through 5. The configuration is as follows:

```
1 server:
2   port: 9001
3   spring: #服务名称, 暂时没有用, 讲到SpringCloud服务调用的时候才会有用。
4     application:
5       name: myshop-user
```



## 4.编写启动类



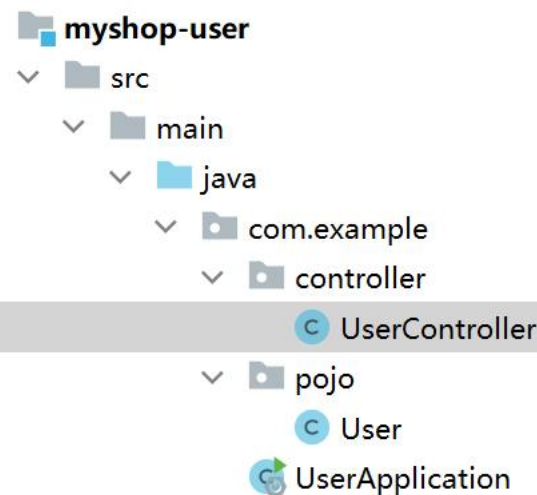
```
1 package com.example;
2
3 import ...
4
5
6 /**
7  * 用户微服务
8  */
9 @SpringBootApplication
10 public class UserApplication {
11
12     public static void main(String[] args) {
13
14         SpringApplication.run(UserApplication.class, args);
15     }
16 }
```



## 5.编写UserController及User实体类

```
/**
 * UserController
 */
@RequestMapping("/user")
@RestController // @RestController=@RequestMapping + @Respo
public class UserController {
    /**
     * 查询所有用户
     */
    @RequestMapping(method = RequestMethod.GET)
    public List<User> findAll() {
        // 模拟用户数据
        List<User> list = new ArrayList<User>();
        list.add(new User( id: 1, username: "张三", password: "123456", sex: "男", money: 1000.0));
        list.add(new User( id: 2, username: "李四", password: "123456", sex: "女", money: 2000.0));
        list.add(new User( id: 3, username: "陈五", password: "123456", sex: "男", money: 2500.0));

        return list;
    }
}
```



## 5.编写UserController及User实体类

| 请求方法   | 用途                 |
|--------|--------------------|
| POST   | 创建一个新的资源并返回新资源     |
| GET    | 查询特定资源             |
| PUT    | 修改资源的全部属性并返回修改后的资源 |
| PATCH  | 修改资源的特定属性并返回修改后的资源 |
| DELETE | 删除资源               |

# 微服务架构设计案例

```
package com.example.pojo;
```

```
import java.io.Serializable;
```

```
/**
```

```
 * 用户实体
```

```
 */
```

7 个用法

```
public class User implements Serializable{
```

3 个用法

```
private Integer id; // 编号
```

3 个用法

```
private String username; // 用户名
```

3 个用法

```
private String password; // 密码
```

3 个用法

```
private String sex; // 性别
```

3 个用法

```
private Double money; // 余额
```

0 个用法

```
public User() {
```

```
}
```

## 微服务架构设计案例

```
public User(Integer id, String username, String password, String sex, Double money) {  
    this.id = id;  
    this.username = username;  
    this.password = password;  
    this.sex = sex;  
    this.money = money;  
}
```

0 个用法

```
public Integer getId() { return id; }
```

0 个用法

```
public void setId(Integer id) { this.id = id; }
```

0 个用法

```
public String getUsername() { return username; }
```

0 个用法

```
public void setUsername(String username) { this.username = username; }
```

0 个用法

```
public String getPassword() { return password; }
```

0 个用法

```
public void setPassword(String password) { this.password = password; }
```

0 个用法

```
public String getSex() { return sex; }
```

## 6.测试 Postman



The screenshot shows an IDE with a project named 'myshop-parent'. The file explorer on the left shows the project structure: 'myshop-parent' (C:\Users\86134\project\m) containing '.idea', '.mvn', 'myshop-user', 'src', and 'main'. The main editor displays the code for 'UserController.java' (partially visible), 'User.java', and 'UserApplication.java'. The 'UserApplication.java' code is as follows:

```
7 * 用户微服务
8 */
9 @SpringBootApplication
10 public class UserApplication {
11
12     public static void main(String[] args) {
13 
```

Below the code editor, the '运行' (Run) tab is active, showing the output of the 'UserApplication' class. The console output is as follows:

```
运行: UserApplication x
控制台 Actuator
: Starting UserApplication using Java 1.8.0_333 on TF-laptop with PID 1
: No active profile set, falling back to 1 default profile: "default"
lebServer : Tomcat initialized with port(s): 9001 (http)
Service : Starting service [Tomcat]
rdEngine : Starting Servlet engine: [Apache Tomcat/9.0.80]
[/] : Initializing Spring embedded WebApplicationContext
.onContext : Root WebApplicationContext: initialization completed in 878 ms
lebServer : Tomcat started on port(s): 9001 (http) with context path ''
: Started UserApplication in 1.853 seconds (JVM running for 3.011)
```

The line 'Tomcat initialized with port(s): 9001 (http)' is highlighted with a red box.



# 微服务架构设计案例

# Postman



≡

←

→

Home

Workspaces ▾

API Network ▾

Explore

Search Postman

Invite

Settings

Notifications

Upgrade ▾

—

□

Overview

POST http://localhost:9001/

GET http://localhost:9001/u

+

...

No Environment ▾

HTTP

http://localhost:9001/user

Save ▾

✎

💬

GET ▾

http://localhost:9001/user

Send ▾

Params

Authorization

Headers (8)

Body ●

Pre-request Script

Tests

Settings

Cookies

Query Params

|  | Key | Value | Description | ... | Bulk Edit |
|--|-----|-------|-------------|-----|-----------|
|  | Key | Value | Description |     |           |

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 152 ms

Size: 393 B

Save as Example

...

Pretty

Raw

Preview

Visualize

JSON ▾

↻

```
1  [
2    {
3      "id": 1,
4      "username": "张三",
5      "password": "123456",
6      "sex": "男",
7      "money": 1000.0
8    },
9    {
10     "id": 2,
11     "username": "李四",
12     "password": "123456",
13     "sex": "女",
14     "money": 2000.0
15   }
16 ]
```

# IDEA中开发Web UI微服务

Web UI微服务需要实现两方面的功能：一.实现对Rest API微服务的调用；二.用户界面设计

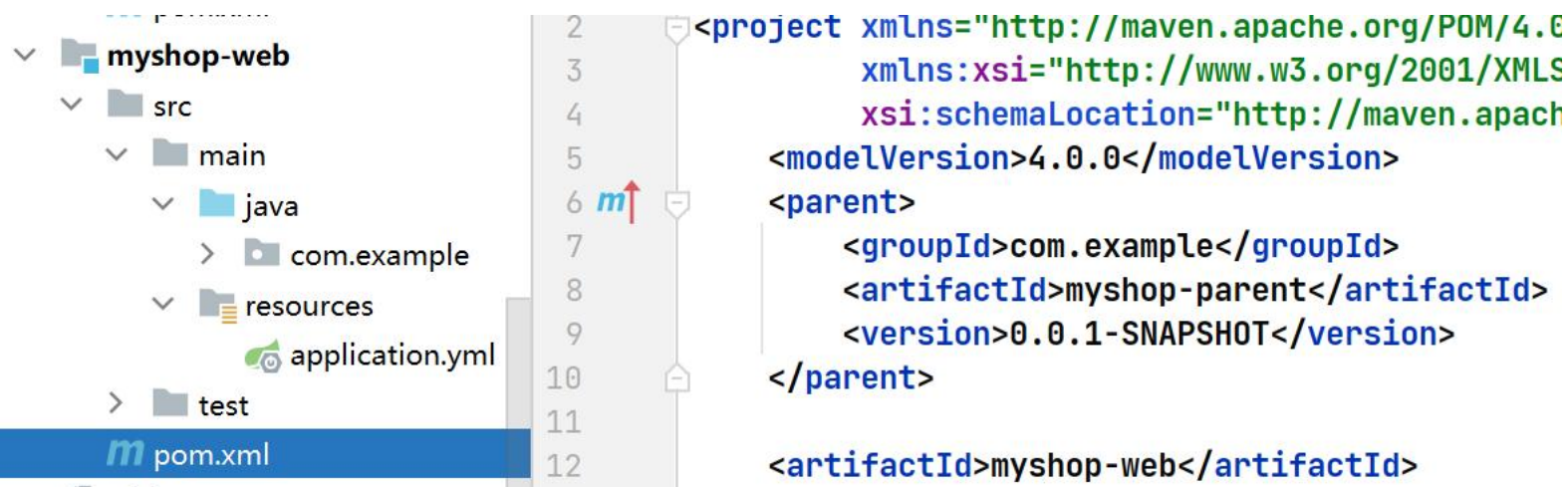
## 快速搭建Spring Boot-Web UI微服务

过程和user微服务类似，只是不需要管理数据库中数据

根据需求，web微服务需要远程调用用户微服务的方法（根据id查询用户信息）。

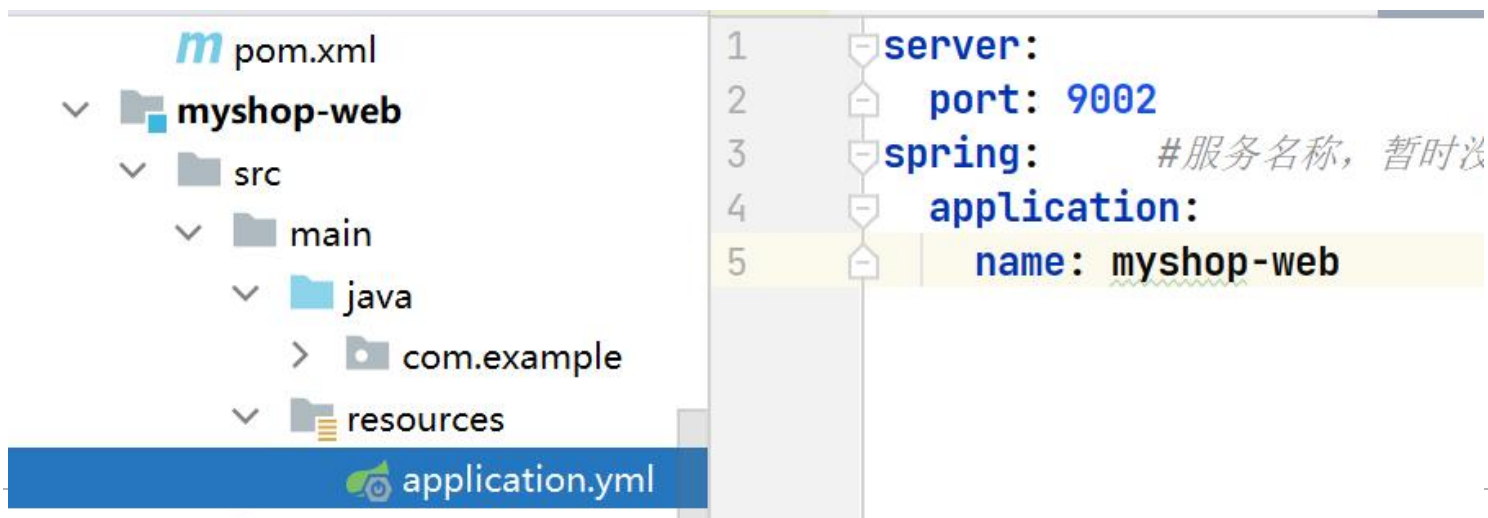
这时候，怎么办？ 继续学习服务通信课程

## pom.xml



```
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3
4   <modelVersion>4.0.0</modelVersion>
5
6   <parent>
7     <groupId>com.example</groupId>
8     <artifactId>myshop-parent</artifactId>
9     <version>0.0.1-SNAPSHOT</version>
10  </parent>
11
12  <artifactId>myshop-web</artifactId>
```

## application.yml



```
1 server:
2   port: 9002
3   spring: #服务名称, 暂时没
4     application:
5       name: myshop-web
```



# 编写WebController

```
package com.example.controller;
import ...

/**
 * web购票Controller
 */
@RequestMapping("/web")
@RestController
public class WebController {

    /**
     * 购票方法
     */
    @RequestMapping(value = "/order", method = RequestMethod.GET )
    public String order(){
        // 模拟当前用户
        Integer id = 2;
        // 查询用户微服务，获取用户具体信息
        System.out.println("==正在购票...");
        return "购票成功";
    }
}
```

## Postman测试

The screenshot displays an IDE interface with a project explorer on the left, a code editor in the center, and a console window at the bottom.

**Project Explorer:** Shows a project named `mvshon-web` with sub-projects `pojo` and `UserApplication`. The `resources` folder is expanded, showing `target` and `pom.xml`.

**Code Editor:** Displays the `WebApplication` class, which is annotated with `@SpringBootApplication`. The code is as follows:

```
9  @SpringBootApplication
10 public class WebApplication {
11     public static void main(String[] args) {
12
13         SpringApplication.run(WebApplication.class, args)
14     }
```

**Console Window:** Shows the output of the application startup. The tabs are labeled "控制台" (Console) and "Actuator". The output is as follows:

```
plication : Starting WebApplication using Java 1.8.0_333 on TF-laptop with
plication : No active profile set, falling back to 1 default profile: "def
tomcat.TomcatWebServer : Tomcat initialized with port(s): 9002 (http)
.core.StandardService : Starting service [Tomcat]
```

# 微服务架构设计案例



http://localhost:9002/web/order

POST



http://localhost:9002/web/order

Params

Authorization

Headers (8)

Body ●

Pre-request Script

T

## Query Params

|  | Key | Value |
|--|-----|-------|
|  | Key | Value |

Body

Cookies

Headers (5)

Test Results

Pretty

Raw

Preview

Visualize

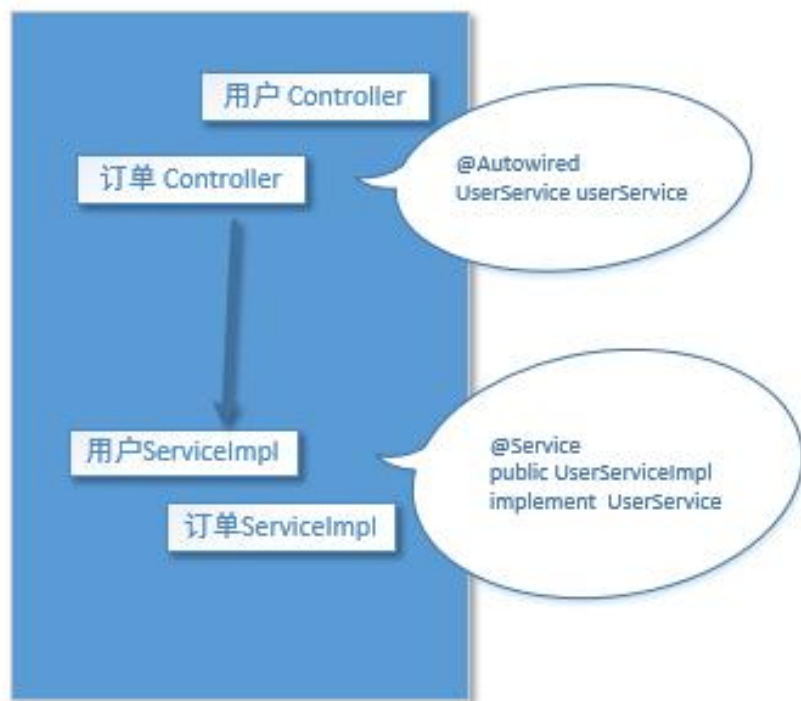
Text



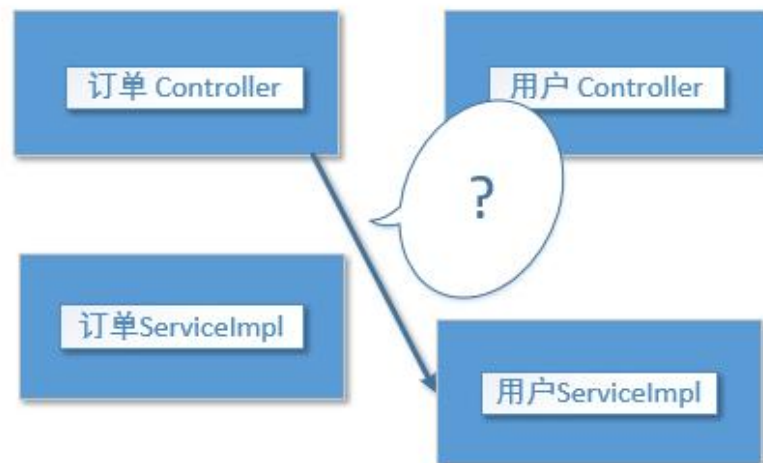
1 购票成功

# 问题：服务间调用

在同一工程下利用Spring注入获得Service



在不同工程之间如何调用



# 设计依赖关系

## 将公共包统一管理

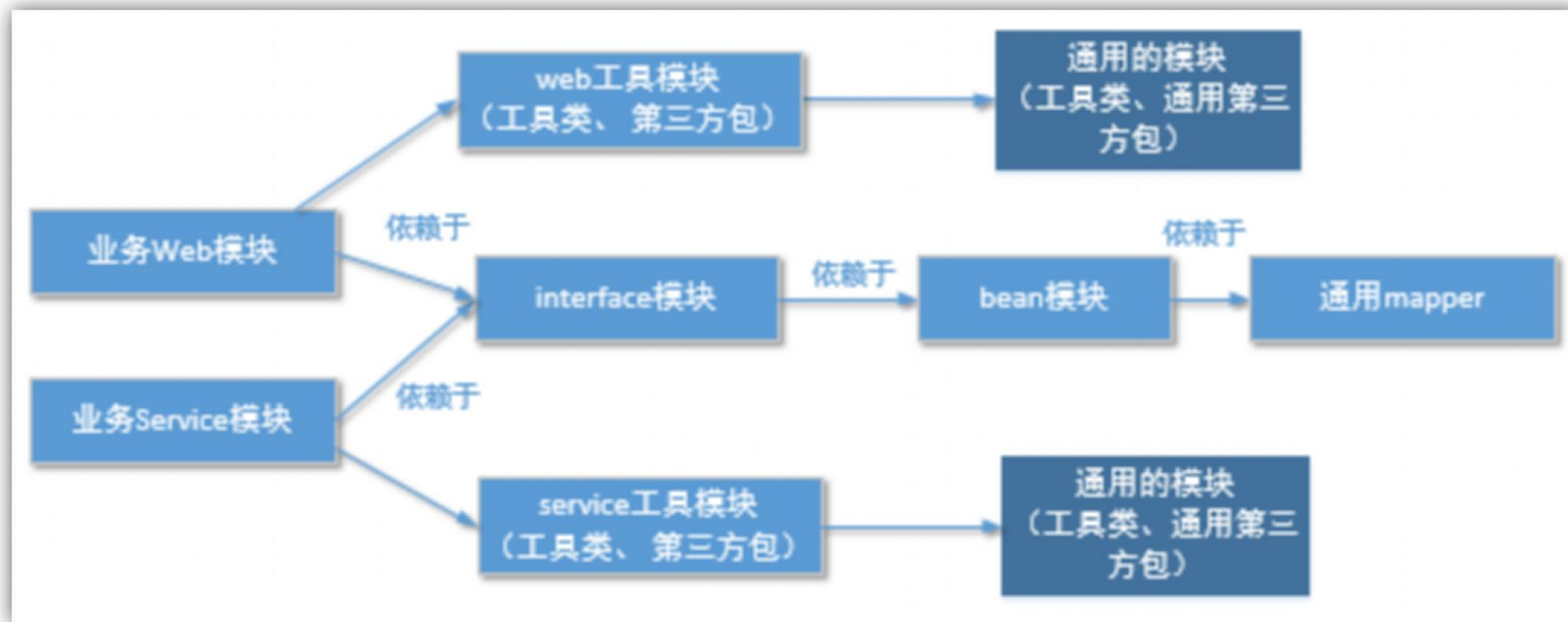


## 第三方依赖包

第三方依赖包分为三种：

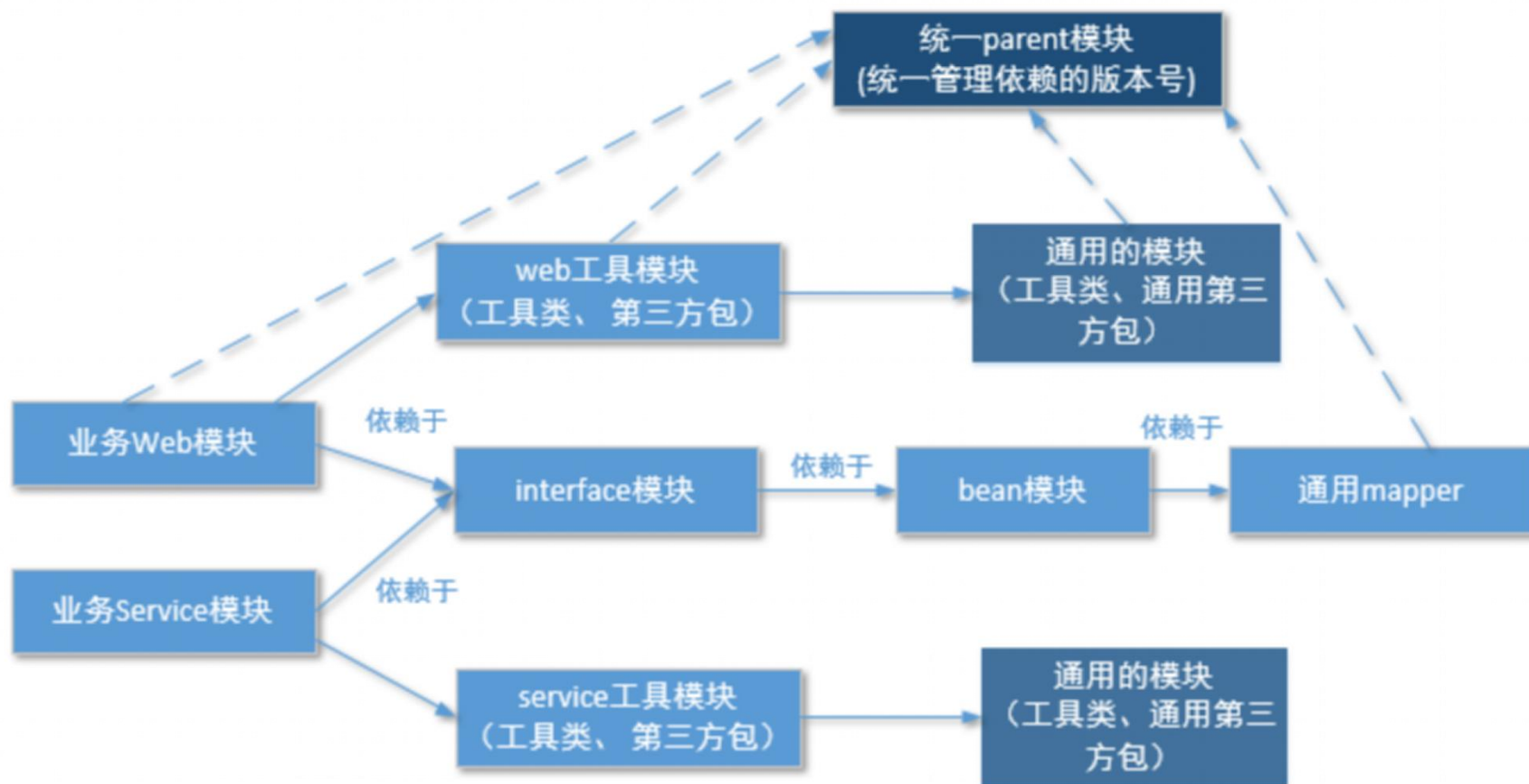
- 1、web业务模块用到的第三方包,比如文件上传客户端、页面渲染工具、操作cookie的工具类等等。
- 2、service业务模块用到的第三方包，比如jdbc、mybatis、jedis、activemq工具包等等。
- 3、通用型的第三方包，比如fastjson、httpclient、apache工具包等等。

基于以上三种情况我们可以搭建如下的依赖结构：



## parent

让所有的模块都继承这个parent模块，由这个parent模块来管理版本





## 创建各模块

myshop-user-manage

> .idea

> .mvn

> myshop-bean

> myshop-common-util

> myshop-interface

myshop-parent

> .mvn

.gitignore

HELP.md

mvnw

mvnw.cmd

pom.xml

> myshop-service-util

> myshop-user-service

> myshop-web-util

.gitignore

HELP.md

mvnw

mvnw.cmd

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org
3     xsi:schemaLocation="http://ma
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.7.16</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>myshop-parent</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>myshop-parent</name>
15    <description>myshop-parent</description>
16    <packaging>pom</packaging>
17    <properties>
18        <java.version>1.8</java.version>
19    </properties>
20    <dependencies>
21        <dependency>
22            <groupId>org.springframework.boot</groupId>
23            <artifactId>spring-boot-starter-web</artifactId>
24        </dependency>
```

> myshop-bean

> myshop-common-util

> myshop-interface

myshop-parent

> myshop-service-util

> myshop-user-service

> myshop-web-util

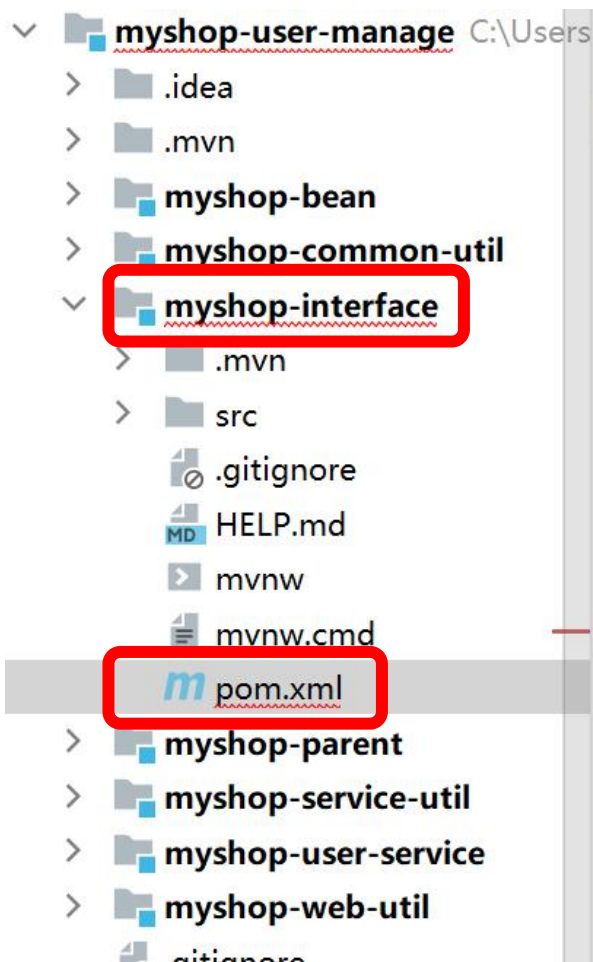
## myshop-bean服务

The image shows a screenshot of an IDE with the following components:

- Project Structure (Left):**
  - myshop-bean
    - .mvn
    - src
      - main
        - java
          - com.example.m
            - MyshopBean
            - UserInfo
      - resources
      - test
    - .gitignore
    - HELP.md
    - mvnw
    - mvnw.cmd
    - pom.xml
- Code Editor (Right):**

```
4 <modelVersion>4.0.0</modelVersion>
5 <parent>
6   <groupId>com.example</groupId>
7   <artifactId>myshop-parent</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
9   <relativePath/> <!-- lookup parent from repository -->
10 </parent>
11 <groupId>com.example</groupId>
12 <artifactId>myshop-bean</artifactId>
13 <version>0.0.1-SNAPSHOT</version>
14 <name>myshop-bean</name>
15 <description>myshop-bean</description>
16 <properties>
17   <java.version>1.8</java.version>
18 </properties>
19 <dependencies>
20   <dependency>
```

## myshop-interface服务



```
<modelVersion>4.0.0</modelVersion>
<parent>
  <groupId>com.example</groupId>
  <artifactId>myshop-parent</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>myshop-interface</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>myshop-interface</name>
<description>myshop-interface</description>
<properties>
  <java.version>1.8</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>com.example</groupId>
    <artifactId>myshop-bean</artifactId>
  </dependency>
</dependencies>
```



# myshop-user-service服务

- > myshop-bean
- > myshop-common-util
- > myshop-interface
- > myshop-parent
- > myshop-service-util
- ▼ myshop-user-service

- > .mvn
- > src
- ⊗ .gitignore
- MD HELP.md
- > mvnw
- ≡ mvnw.cmd

m pom.xml

- > myshop-web-util
- ⊗ .gitignore
- MD HELP.md
- > mvnw
- ≡ mvnw.cmd
- m pom.xml

```
<groupId>com.example</groupId>
<artifactId>myshop-parent</artifactId>
<version>0.0.1-SNAPSHOT</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
```

```
<groupId>com.example</groupId>
<artifactId>myshop-user-service</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>myshop-user-service</name>
<description>myshop-user-service</description>
<properties>
  <java.version>1.8</java.version>
</properties>
```

```
<dependencies>
  <dependency>
    <groupId>com.example</groupId>
    <artifactId>myshop-interface</artifactId>
  </dependency>
```

```
<dependency>
  <groupId>com.example</groupId>
  <artifactId>myshop-service-util</artifactId>
</dependency>
```