SaaS-Export第07天

学习目标

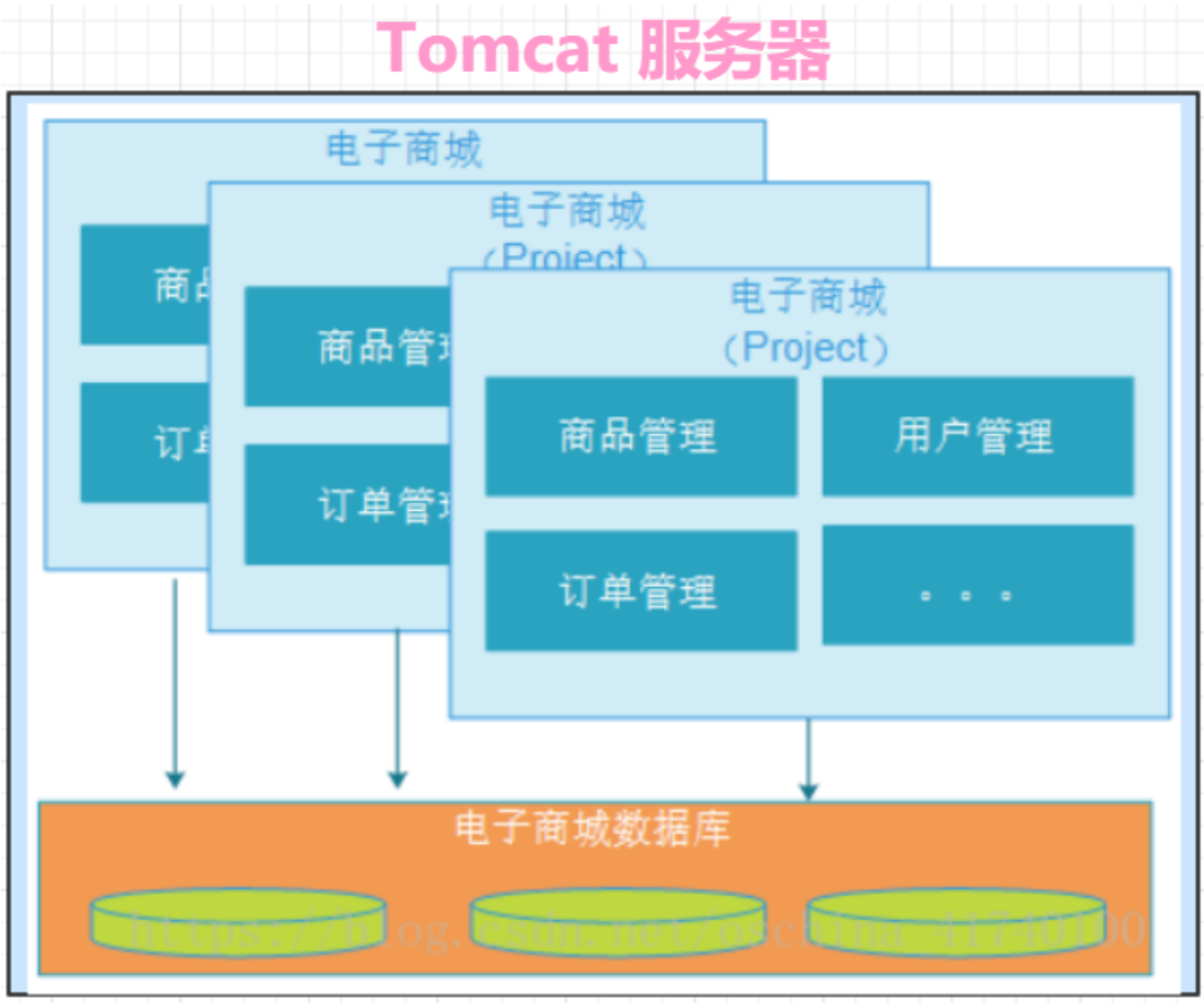
软件架构的发展过程

介绍

软件架构的发展经历了从单体结构（集中式架构）、垂直架构、分布式架构到微服务架构的过程。

集中式架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是影响项目开发的关键。



> 特点：
>
> - 所有的功能集成在一个项目工程中。
> - 所有的功能打一个war包部署到服务器。
> - 应用与数据库分开部署。

- 通过部署应用集群和数据库集群来提高系统的性能。

优点：

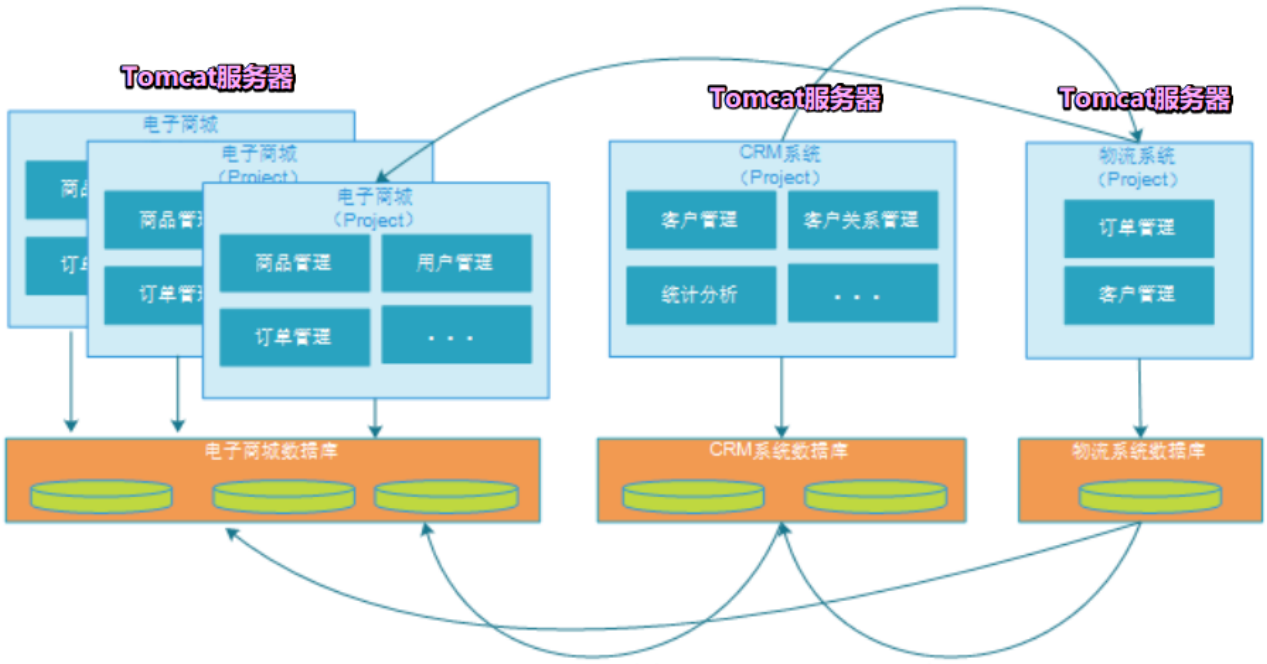
- 代码耦合，开发维护困难
- 无法针对不同模块进行针对性优化
- 无法水平扩展
- 单点容错率低，并发能力差

缺点：

- 全部功能集成在一个工程中，对于大型项目不易开发、扩展及维护。
- 系统性能扩展只能通过扩展集群结点，成本高、有瓶颈。

垂直架构

当访问量逐渐增大，单一应用无法满足需求，此时为了应对更高的并发和业务需求，我们根据业务功能对系统进行拆分。



特点：

- 以单体结构规模的项目为单位进行垂直划分项目即将一个大项目拆分成一个一个单体结构项目。
- 项目与项目之间的存在数据冗余，耦合性较大，比如上图中三个项目都存在客户信息。
- 项目之间的接口多为数据同步功能，如：数据库之间的数据库，通过网络接口进行数据库同步。

优点：

- 系统拆分实现了流量分担，解决了并发问题
- 可以针对不同模块进行优化
- 方便水平扩展，负载均衡，容错率提高
- 系统间相互独立

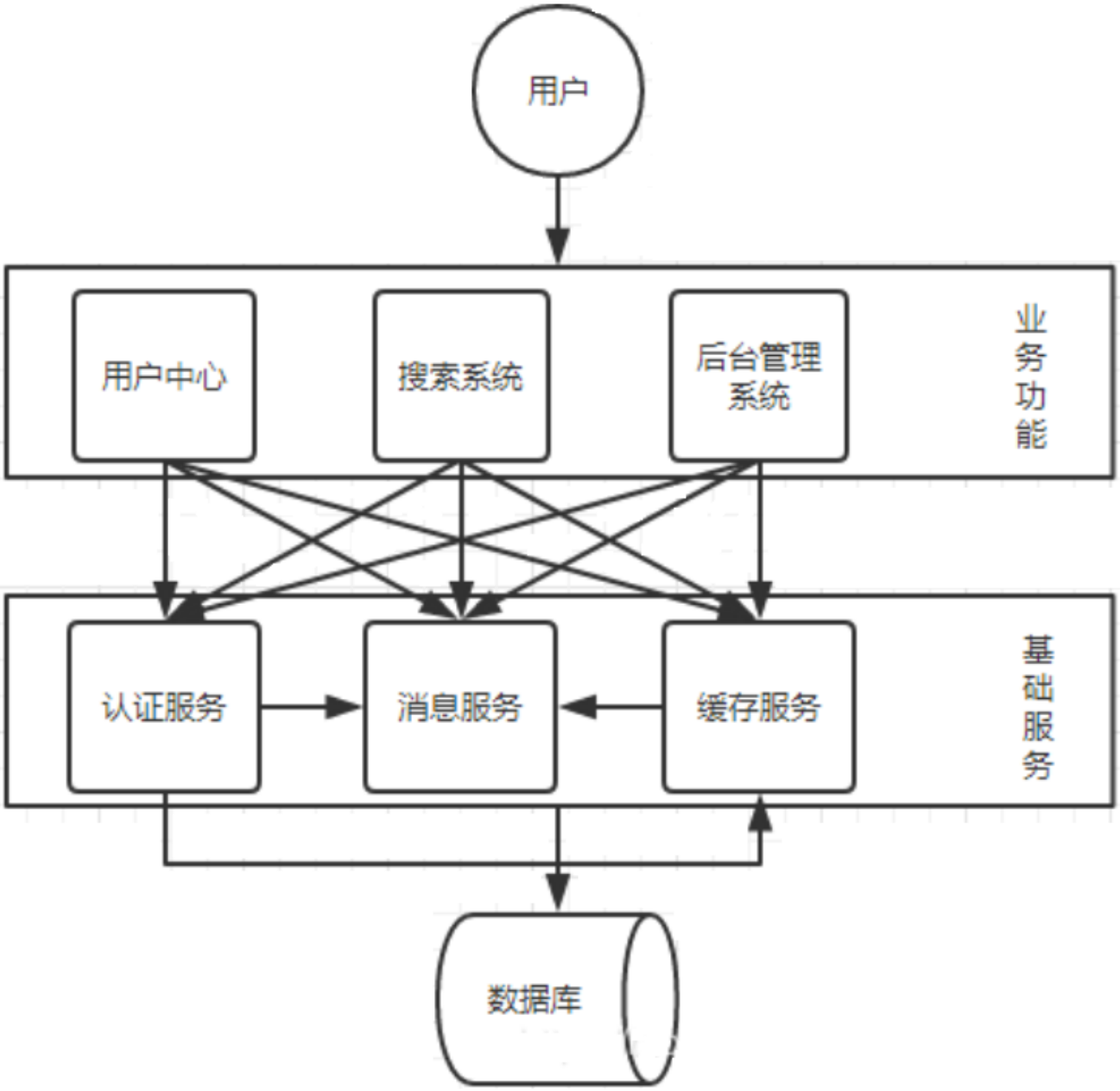
缺点：

- 服务之间相互调用，如果某个服务的端口或者ip地址发生改变，调用的系统得手动改变

- 搭建集群之后，实现负载均衡比较复杂

分布式架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式调用是关键。



优点：

- 将基础服务进行了抽取，系统间相互调用，提高了代码复用和开发效率

缺点：

- 系统间耦合度变高，调用关系错综复杂，难以维护
- 搭建集群之后，负载均衡比较难实现

小结

> 集中式架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。

> 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

> 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

> 流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。

流动计算架构：在分布式计算架构下，添加了监控中心与调度中心。

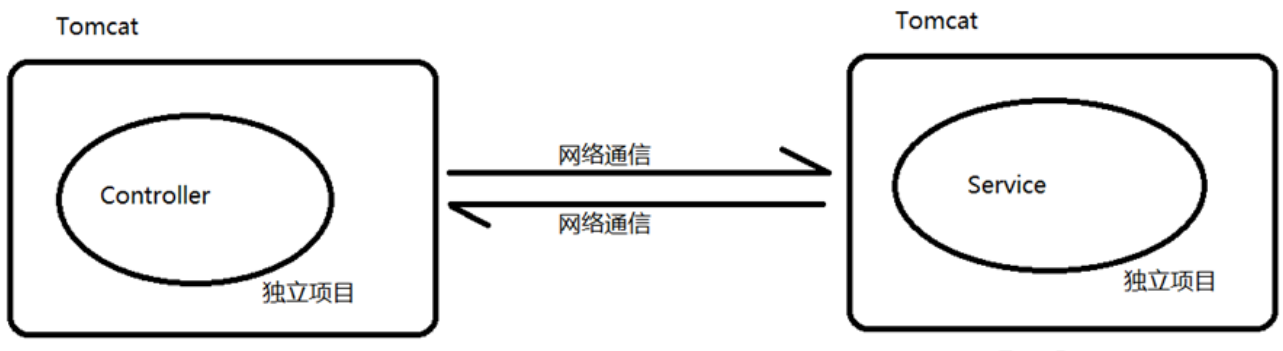
> 什么是SOA?

SOA全称为Service-Oriented Architecture，即面向服务的架构。它可以根据需求通过网络对松散耦合的粗粒度应用组件(服务)进行分布式部署、组合和使用。一个服务通常以独立的形式存在于操作系统进程中。

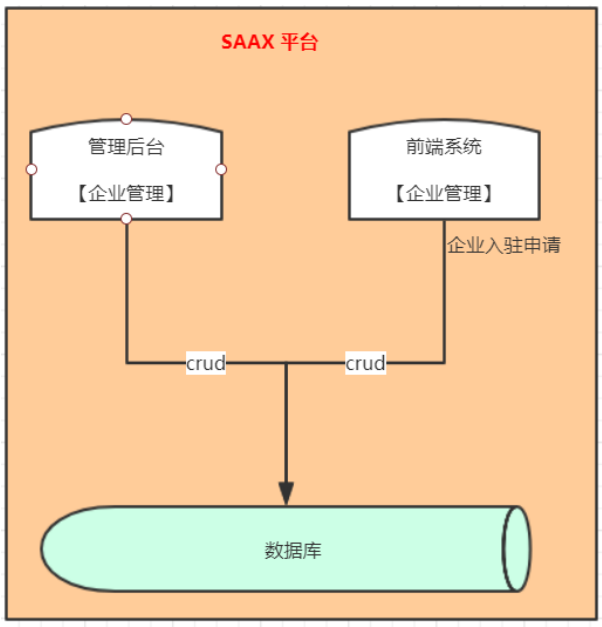
站在功能的角度，把业务逻辑抽象成可复用、可组装的服务，通过服务的编排实现业务的快速再生，目的：把原先固有的业务功能转变为通用的业务服务，实现业务逻辑的快速复用。

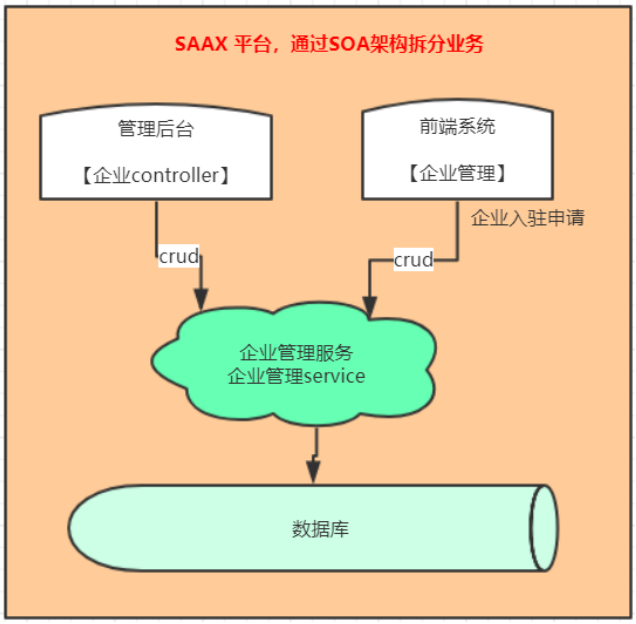
通过上面的描述可以发现SOA有如下几个特点：**分布式、可重用、扩展灵活、松耦合。**

原来的单体工程项目大多分为三层：表现层(Controller)、业务层(Service)、持久层(Dao)，要改为SOA架构，其实就是将业务层提取为服务并且独立部署即可，表现层通过网络和业务层进行通信，如下图：

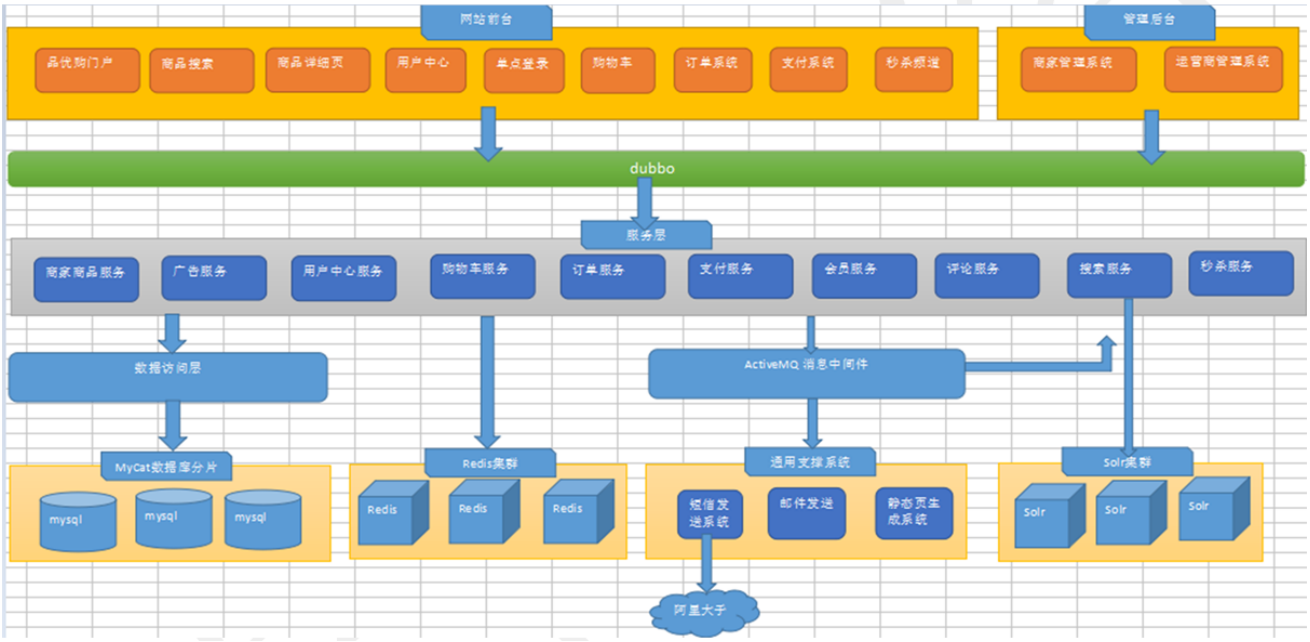


原来的单体项目如何改为SOA架构?





下图以电商系统举例来说明SOA架构：

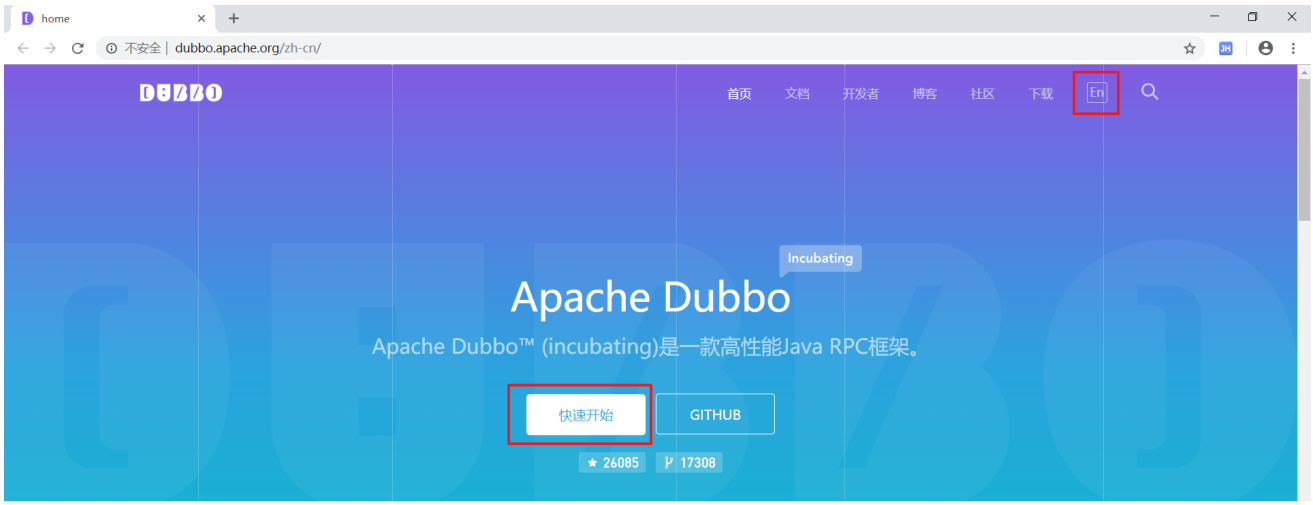


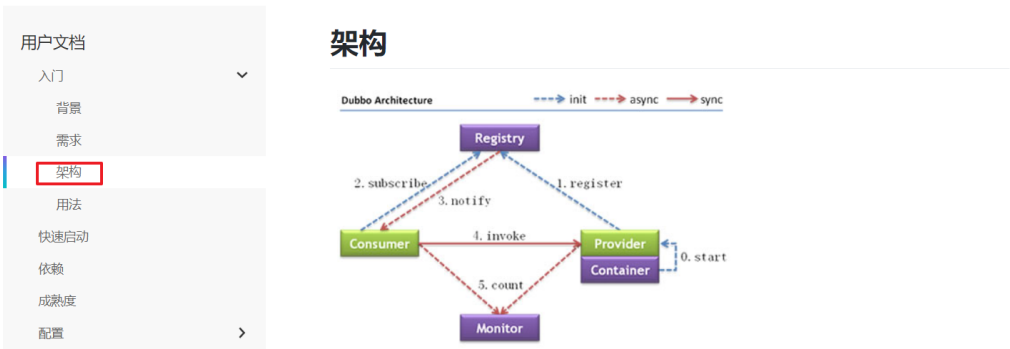
Apache Dubbo（一）介绍

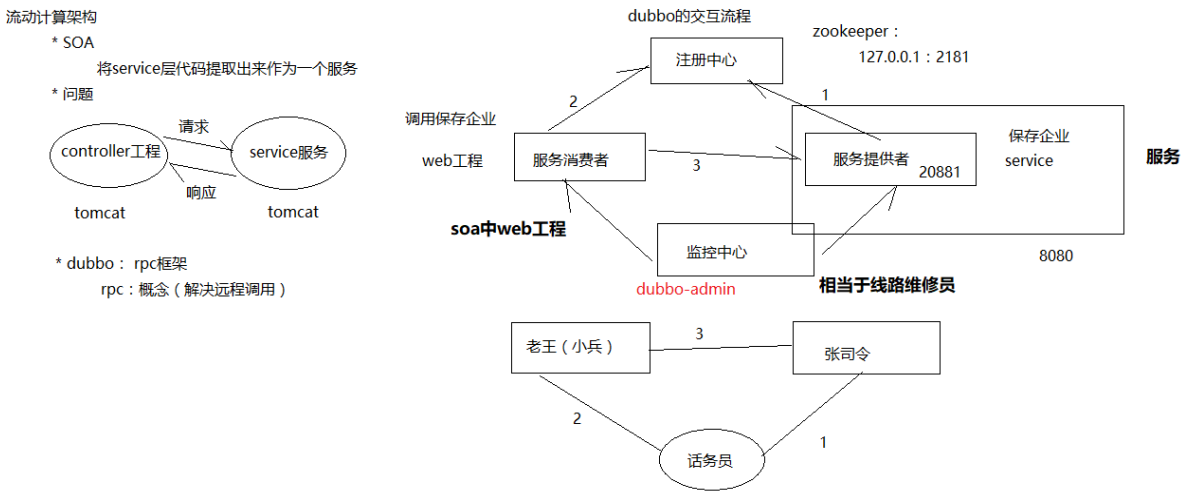
Apache Dubbo是一款高性能的Java RPC框架。其前身是阿里巴巴公司开源的一个高性能、轻量级的开源Java RPC框架，可以和Spring框架无缝集成。【RPC 表示远程调用的意思】

借助Dubbo可以实现基于SOA架构的软件设计。

官网：







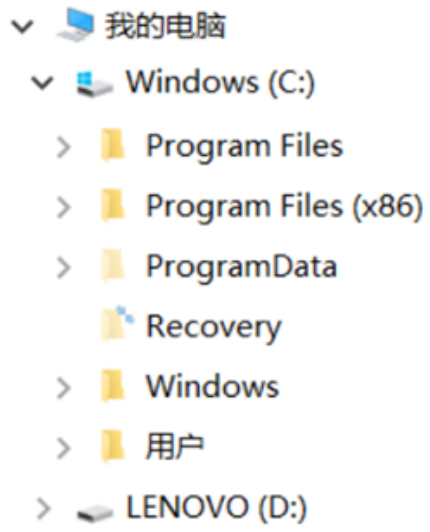
Apache Dubbo（二）服务注册中心Zookeeper

目标

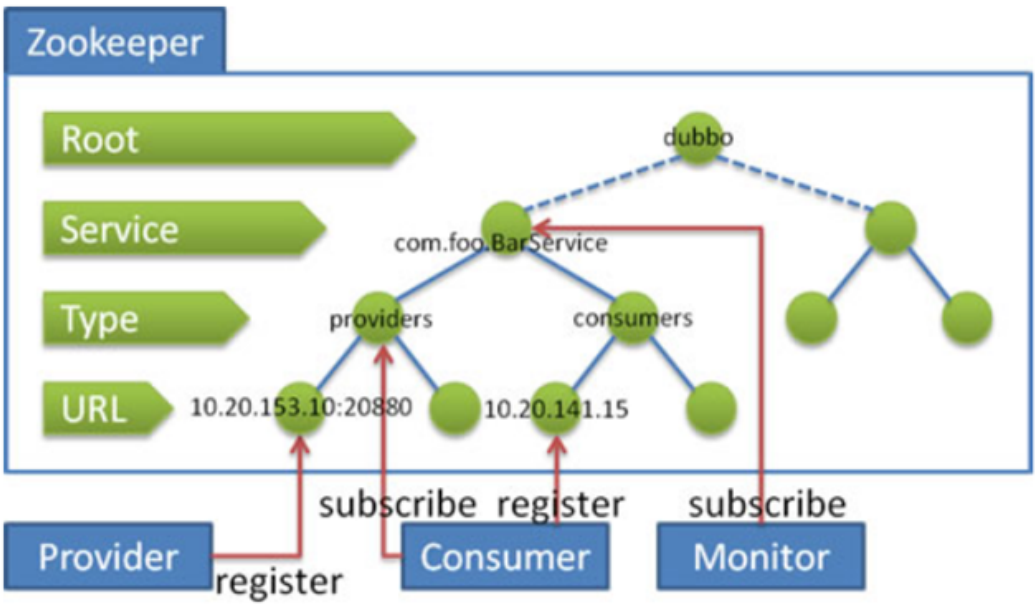
1. 介绍
2. 安装启动

介绍

- Zookeeper 是 Apache Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用 。
- 为了便于理解Zookeeper的树型目录服务，我们先来看一下我们电脑的文件系统(也是一个树型目录结构)：



- 我的电脑可以分为多个盘符（例如C、D、E等），每个盘符下可以创建多个目录，每个目录下面可以创建文件，也可以创建子目录，最终构成了一个树型结构。通过这种树型结构的目录，我们可以将文件分门别类的进行存放，方便我们后期查找。而且磁盘上的每个文件都有一个唯一的访问路径，例如：C:\Windows\itcast\hello.txt。
- Zookeeper树型目录服务：



- 流程说明：
  1. 服务提供者(Provider)启动时:

     向 `/dubbo/com.foo.BarService/providers` 目录下写入自己的 URL 地址
  2. 服务消费者(Consumer)启动时:

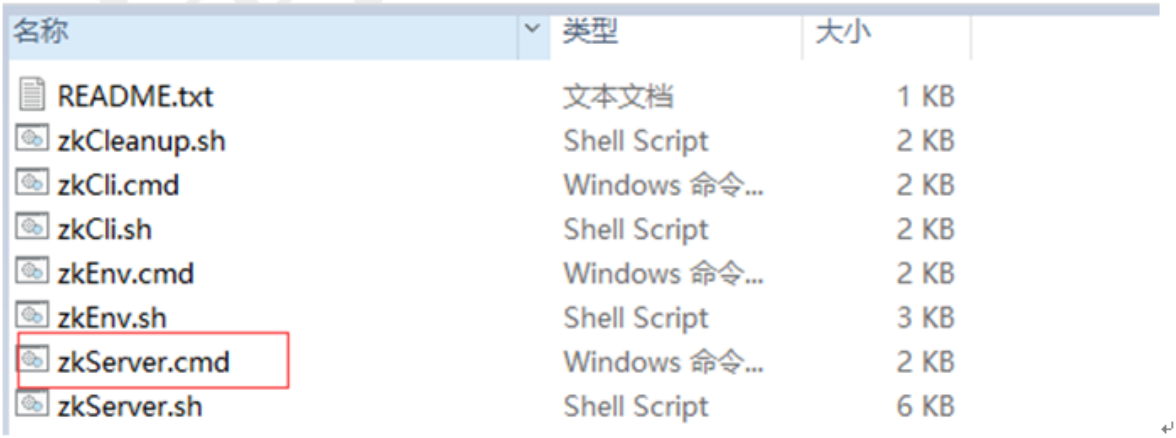
     订阅 `/dubbo/com.foo.BarService/providers` 目录下的提供者 URL 地址。

     并向 `/dubbo/com.foo.BarService/consumers` 目录下写入自己的 URL 地址
  3. 监控中心(Monitor)启动时:

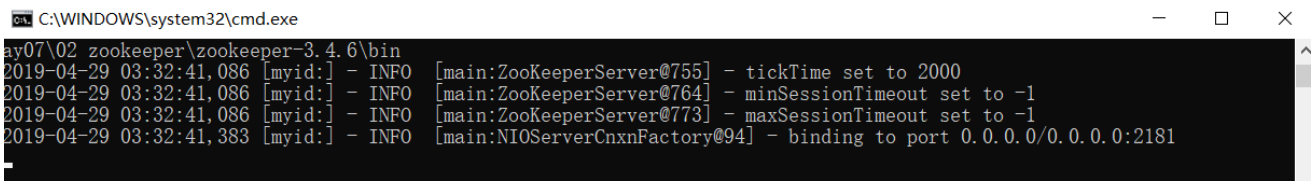
订阅 `/dubbo/com.foo.BarService` 目录下的所有提供者和消费者 URL 地址

windows安装启动

使用资料中提供的 windows 版本 zookeeper 服务器进行安装即可



进入安装路径的 bin 目录，双击 zkServer.cmd 即可启动 zookeeper 服务

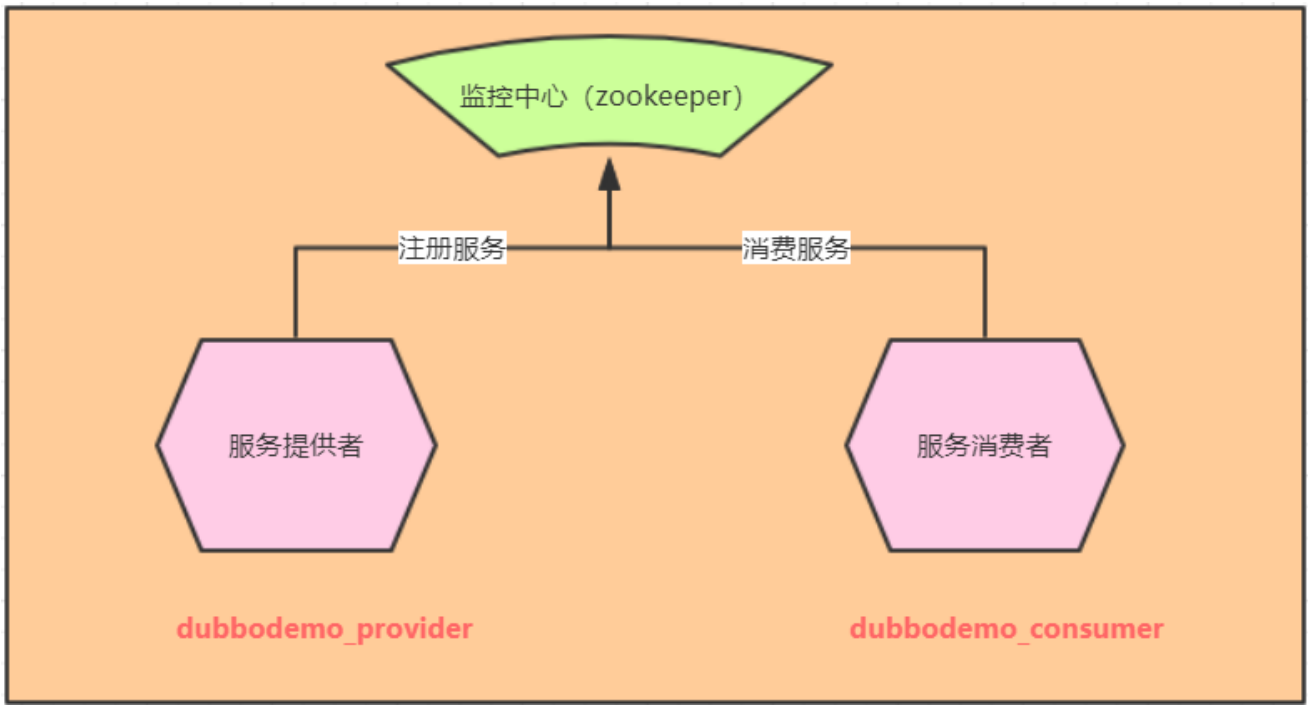


linux安装启动

```
> # 先确保jdk环境已经安全

> # tar -zxvf zookeeper-3.4.6.tar.gz        把压缩包上传到 linux 系统 ，解压压缩包

> # cd zookeeper-3.4.6                      进入zookeeper-3.4.6目录

> # mkdir data                              在zookeeper-3.4.6目录下创建data目录

> # cd conf                                 进入conf目录

> # mv zoo_sample.cfg zoo.cfg               把zoo_sample.cfg 改名为zoo.cfg

> # vi zoo.cfg                              打开zoo.cfg文件，修改dataDir属性:
                                            dataDir=/root/zookeeper-3.4.6/data
                                            (i,修改dataDir=/root/zookeeper-
3.4.6/data
                                            shift + :    ,    wq )


> # ./zkServer.sh start                     进入bin目录，启动服务命令
> # ./zkServer.sh stop                      停止服务命令
> # ./zkServer.sh status                    查看服务状态
```

Dubbo入门案例- 分析

Dubbo作为一个RPC框架，其最核心的功能就是要实现跨网络的远程调用。本小节就是要创建两个应用，一个作为服务的提供者，一个作为服务的消费者。通过Dubbo来实现服务消费者远程调用服务提供者的方法。



Dubbo入门案例- 服务提供者

说明

Dubbo作为一个RPC框架，其最核心的功能就是要实现跨网络的远程调用。当前入门案例就是要创建两个应用，一个作为服务的提供者，一个作为服务的消费者。通过Dubbo来实现服务消费者远程调用服务提供者的方法。

目标

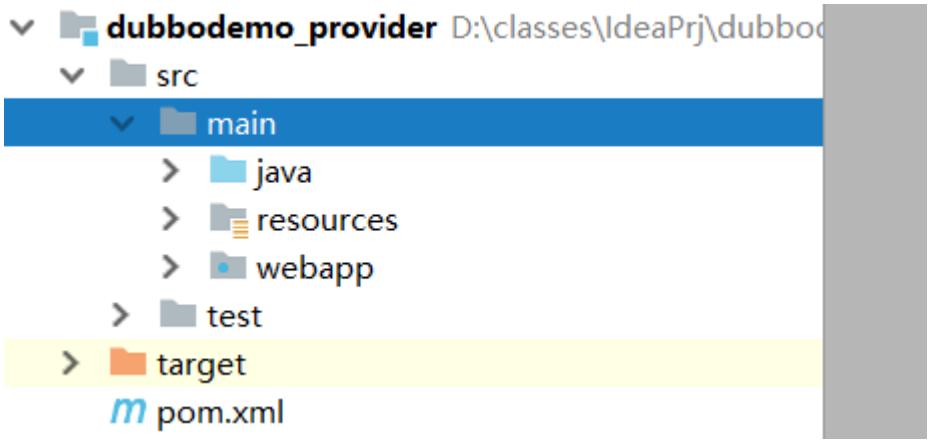
本单元主要实现：创建服务提供者项目、实现发布服务。

步骤

1. 创建web项目: dubbodemo_provider
2. 添加依赖
3. 编写web.xml
4. 编写dubbo-provider.xml
5. 编写服务接口
6. 编写服务实现
7. 启动tomcat，发布项目

实现

1. 创建web项目: dubbodemo_provider



2. 添加依赖

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

<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">
  <modelVersion>4.0.0</modelVersion>
  <groupId>cn.itcast</groupId>
  <artifactId>dubbodemo_provider</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <properties>
    <spring.version>5.0.2.RELEASE</spring.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
```

```
      <artifactId>spring-aspects</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jms</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context-support</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <!-- dubbo相关 -->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>dubbo</artifactId>
      <version>2.6.6</version>
    </dependency>
    <dependency>
      <groupId>io.netty</groupId>
      <artifactId>netty-all</artifactId>
      <version>4.1.32.Final</version>
    </dependency>
    <dependency>
      <groupId>org.apache.curator</groupId>
      <artifactId>curator-framework</artifactId>
      <version>4.0.0</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.zookeeper</groupId>
          <artifactId>zookeeper</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.apache.zookeeper</groupId>
      <artifactId>zookeeper</artifactId>
      <version>3.4.7</version>
    </dependency>
    <dependency>
      <groupId>com.github.sgroschupf</groupId>
      <artifactId>zkclient</artifactId>
      <version>0.1</version>
    </dependency>
  </dependencies>
</project>
```

3. 编写web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
```

```
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
        version="2.5">

    <!-- 监听器监听其他的spring配置文件 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:dubbo-provider.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

</web-app>
```

4. 编写dubbo-provider.xml

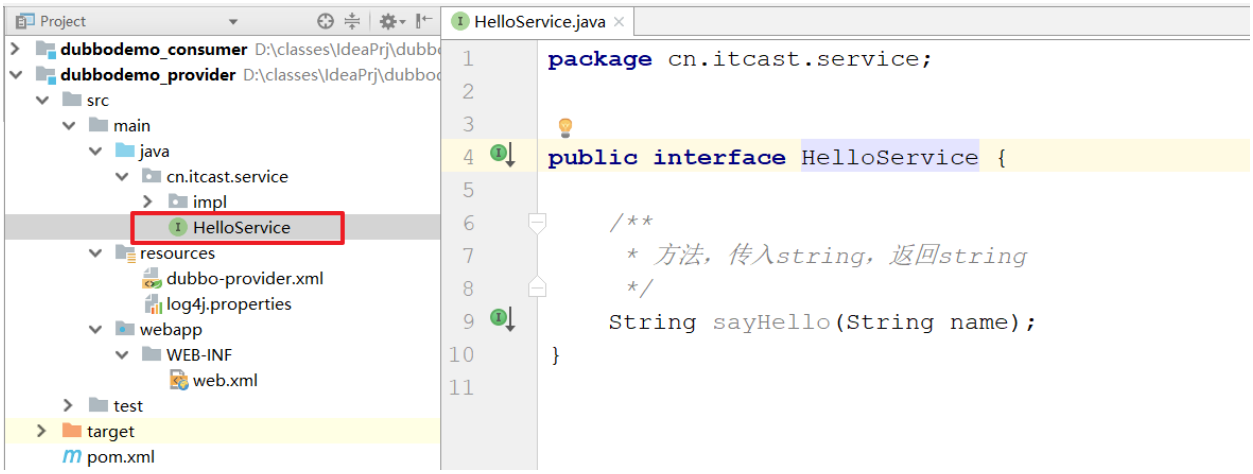
图1：

图2：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://code.alibabatech.com/schema/dubbo
       http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!--指定服务提供者名称，通常为项目名称。可以随意定义，唯一即可。-->
    <dubbo:application name="dubbodemo_provider"/>

    <!--配置注册中心地址-->
    <dubbo:registry address="zookeeper://192.168.12.132:2181"/>
    <!--
     配置请求协议
            name 指定的是传输协议的名称，
                 值列表范围如：dubbo rmi hessian webservice http
            port：服务提供者的真实请求端口
    -->
    <dubbo:protocol name="dubbo" port="20881"/>

```

```
    <!--配置dubbo服务提供者的包扫描-->
    <dubbo:annotation package="cn.itcast.service"/>

</beans>
```

5. 编写服务接口



6. 编写服务实现。注意这里的@Service引入的包。

7. 启动tomcat，发布项目

小结

注意：@Service所在的包，要引入dubbo支持包。不要引入spring的包

Dubbo入门案例- 服务消费者

目标

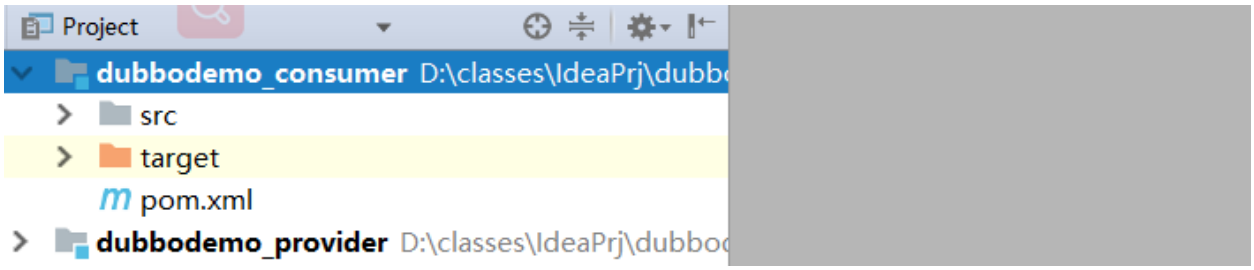
创建服务消费者项目，实现服务的消费。也叫做调用服务。

步骤

1. 创建web项目: dubbodemo_consumer
2. 添加依赖
3. 编写web.xml
4. 编写dubbo-consumer.xml
5. 编写服务接口
6. 编写控制器，调用服务
7. 启动tomcat，测试，观察服务是否调用成功。

实现

1. 创建web项目: dubbodemo_consumer



2. 添加依赖

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

<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">
  <modelVersion>4.0.0</modelVersion>
  <groupId>cn.itcast</groupId>
  <artifactId>dubbodemo_consumer</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <properties>
    <spring.version>5.0.2.RELEASE</spring.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
```

```
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aspects</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jms</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context-support</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <!-- dubbo相关 -->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>dubbo</artifactId>
      <version>2.6.6</version>
      <exclusions>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-web</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>io.netty</groupId>
      <artifactId>netty-all</artifactId>
      <version>4.1.32.Final</version>
    </dependency>
    <dependency>
      <groupId>org.apache.curator</groupId>
      <artifactId>curator-framework</artifactId>
      <version>4.0.0</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.zookeeper</groupId>
          <artifactId>zookeeper</artifactId>
        </exclusion>
```

```
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.apache.zookeeper</groupId>
      <artifactId>zookeeper</artifactId>
      <version>3.4.7</version>
    </dependency>
    <dependency>
      <groupId>com.github.sgroschupf</groupId>
      <artifactId>zkclient</artifactId>
      <version>0.1</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>
  </dependencies>
</project>
```

3. 编写web.xml

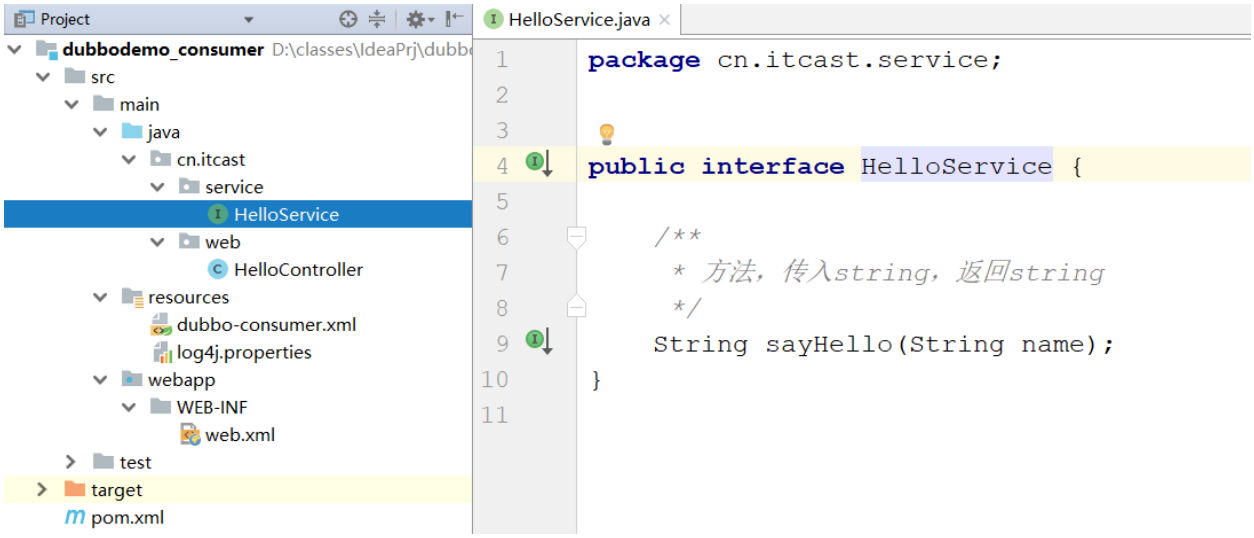
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:dubbo-consumer.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

4. 编写dubbo-consumer.xml

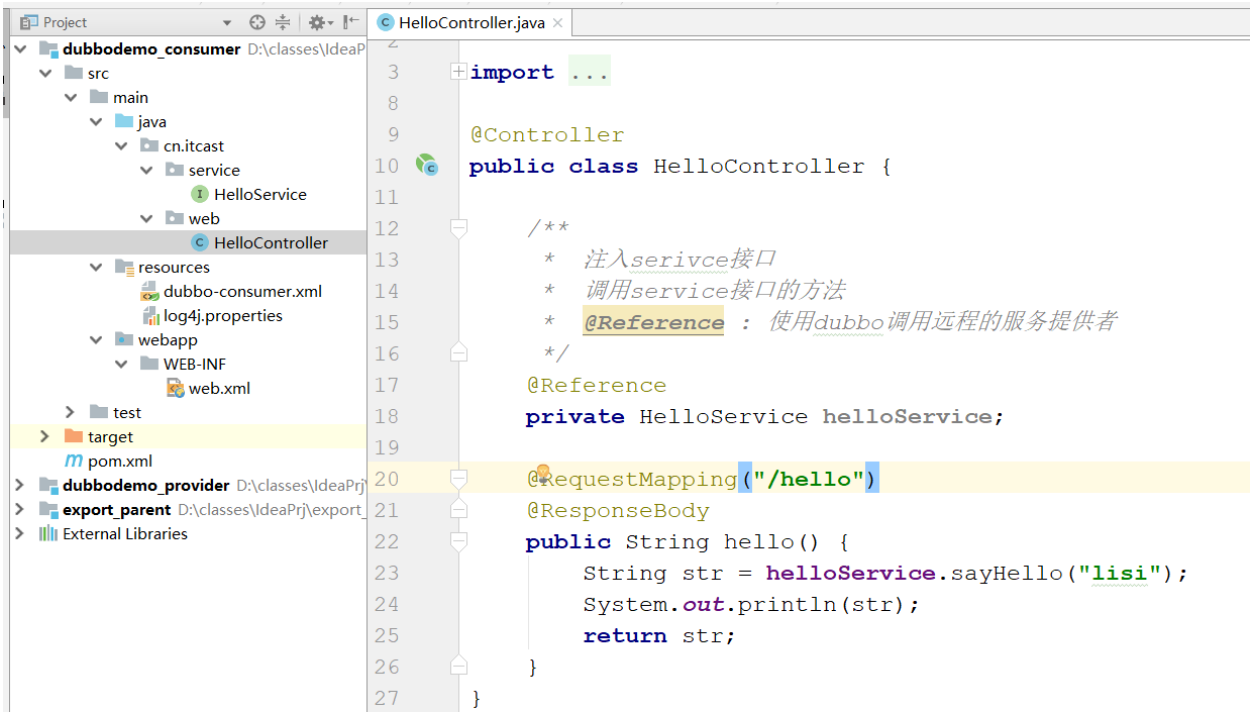
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
```

```
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!--开启SpringMVC注解扫描，扫描@Controller注解-->
    <context:component-scan base-package="cn.itcast.web"/>
    <!--SpringMVC注解驱动-->
    <mvc:annotation-driven/>


    <!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->
    <dubbo:application name="dubbodemo_consumer">
        <dubbo:parameter key="qos.enable" value="false"/>
    </dubbo:application>

    <!--配置注册中心地址-->
    <dubbo:registry address="zookeeper://192.168.12.132:2181"/>

    <!--开启dubbo注解扫描(@Reference注解)-->
    <dubbo:annotation package="cn.itcast.web"/>
</beans>
```

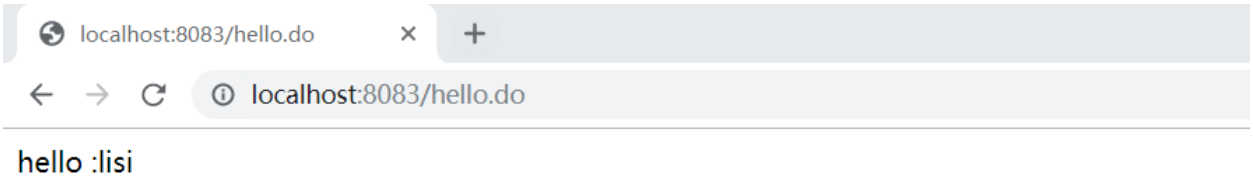
5. 编写服务接口。 注意：这里的接口类名称要与服务端接口名称一致；路径也要一致。



6. 编写控制器，调用服务



7. 启动tomcat，测试，观察服务是否调用成功。



小结

1. 客户端接口类名称要与服务端接口名称一致；路径也要一致。
2. @Reference 用来注入dubbo的服务对象，用的时候不要导错包。

```
import com.alibaba.dubbo.config.annotation.Reference;
```

Dubbo细节（一）通过main函数启动服务

目标

启动服务的两种方式：

1. 部署到tomcat启动，适合正式项目
2. 通过main函数启动，适合开发阶段

之前已经实现tomcat启动服务，现在实现通过man函数启动

实现

1. 前面是通过tomcat启动的服务，在实际的开发中肯定是用这种方式。其实，还可以通过main函数启动服务
2. 查看dubbo官方介绍

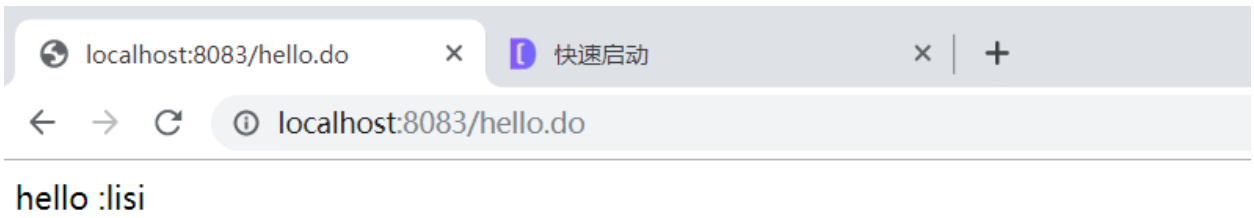
加载 Spring 配置

Provider.java：

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Provider {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new String[]
        context.start();
        System.in.read(); // 按任意键退出
    }
}
```

3. 代码实现如下

4. 访问测试



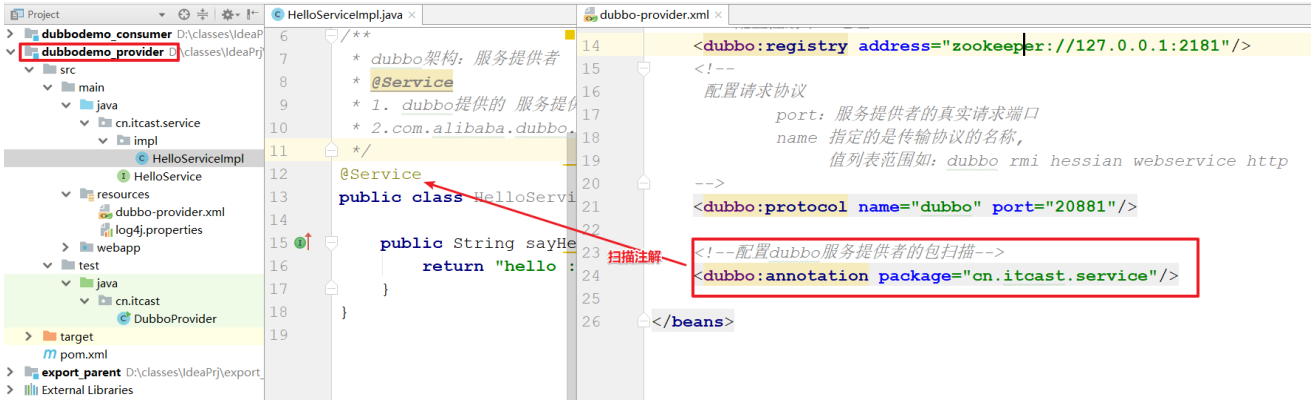
Dubbo细节（二）dubbo配置说明

服务提供者-包扫描配置

> 包扫描配置：

```
服务提供者和服务消费者都需要配置，表示包扫描，作用是扫描指定包(包括子包)下的类
<dubbo:annotation package="com.itheima.service" />
```

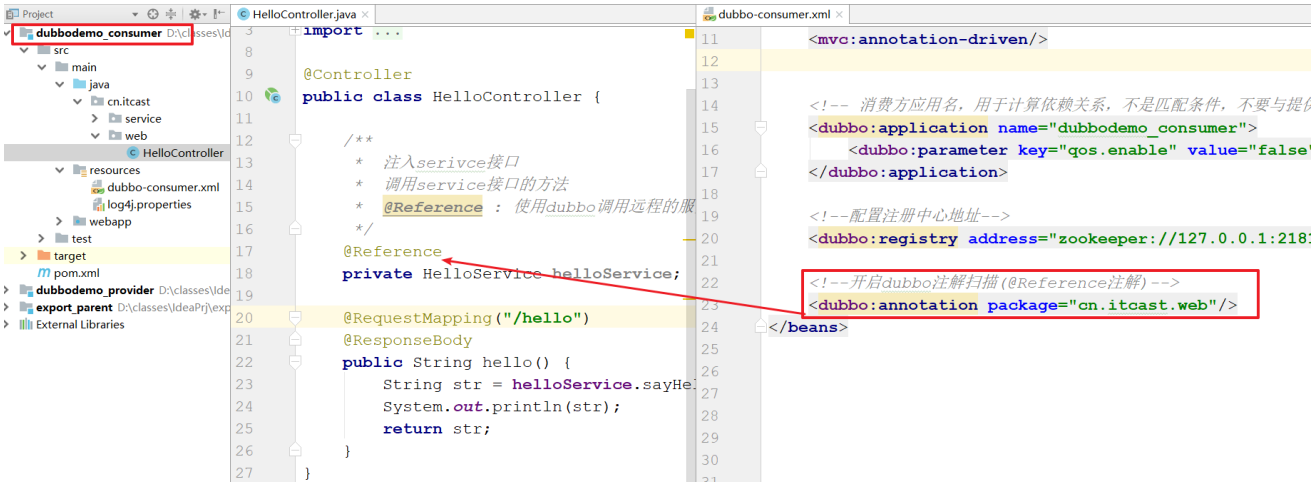
> 服务提供者配置



服务提供者-不使用包扫描的配置

```
如果不使用包扫描，也可以通过如下配置的方式来发布服务
<bean id="helloService" class="com.itheima.service.impl.HelloServiceImpl" />
<dubbo:service interface="com.itheima.api.HelloService" ref="helloService" />
```

服务消费者-包扫描配置



服务消费者-不使用包扫描的配置

```
作为服务消费者，可以通过如下配置来引用服务：
<!-- 生成远程服务代理，可以和本地bean一样使用helloService -->
<dubbo:reference id="helloService" interface="com.itheima.api.HelloService" />
```

小结

```
不使用包扫描的方式发布和引用服务，一个配置项(dubbo:service、dubbo:reference)只能发布或者引用一个服务，如果有多个服务，这种方式就比较繁琐了。推荐使用包扫描方式。
```

Dubbo细节（三）协议

```
<dubbo:protocol name="dubbo" port="20881"/>
```

> 协议一般在服务提供者一方配置，可以指定使用的协议名称和端口号。

> 其中Dubbo支持的协议有：dubbo、rmi、hessian、http、webservice、rest、redis等。

> 推荐使用的是dubbo协议。

dubbo 协议采用单一长连接和 NIO 异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。

> 也可以在同一个工程中配置多个协议，不同服务可以使用不同的协议，例如：

```
<!-- 多协议配置 -->
<dubbo:protocol name="dubbo" port="20880" />
<dubbo:protocol name="rmi" port="1099" />

<!-- 使用dubbo协议暴露服务 -->
<dubbo:service interface="com.itheima.api.HelloService" ref="helloService"
protocol="dubbo" />

<!-- 使用rmi协议暴露服务 -->
<dubbo:service interface="com.itheima.api.DemoService" ref="demoService"
protocol="rmi" />
```

Dubbo细节（四）启动时检查

```
<dubbo:consumer check="false"/>
```

> 配置位置：

配置在服务消费者一方，如果不配置默认check值为true。

> 作用：

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，以便上线时，能及早发现问题。可以通过将check值改为false来关闭检查。

> 应用场景：

建议在开发阶段将check值设置为false，在生产环境下改为true。

Dubbo细节（五）负载均衡

什么是负载均衡?

> 负载均衡（Load Balance）：其实就是将请求分摊到多个操作单元上进行执行，从而共同完成工作任务。

> 在集群负载均衡时，Dubbo 提供了多种均衡策略（包括随机、轮询、最少活跃调用数、一致性Hash），缺省为random随机调用。

> 配置负载均衡策略，既可以在服务提供者一方配置，也可以在服务消费者一方配置

> 配置在服务消费者一方：

```
@Controller
public class HelloController {
    //在服务消费者一方配置负载均衡策略
```

```
    @Reference(check = false,loadbalance = "random")
    private HelloService helloService;

    @RequestMapping("/hello")
    @ResponseBody
    public String getName(String name){
        //远程调用
        String result = helloService.sayHello(name);
        System.out.println(result
        return result;
    }
}
```

> 配置在服务提供者一方：

```
//在服务提供者一方配置负载均衡
@Service(interfaceClass = HelloService.class,loadbalance = "random")
public class HelloServiceImpl implements HelloService {
    public String sayHello(String name) {
        return "hello " + name;
    }
}
```

可以通过启动多个服务提供者来观察Dubbo负载均衡效果。注意：因为我们是在一台机器上启动多个服务提供者，所以需要修改tomcat的端口号和Dubbo服务的端口号来防止端口冲突。在实际生产环境中，多个服务提供者是分别部署在不同的机器上，所以不存在端口冲突问题。

Dubbo管理控制台

介绍

我们在开发时，需要知道Zookeeper注册中心都注册了哪些服务，有哪些消费者来消费这些服务。我们可以通过部署一个管理中心来实现。其实管理中心就是一个web应用，部署到tomcat即可。

安装

安装步骤：

> （1）将资料中的dubbo-admin-2.6.0.war文件复制到tomcat的webapps目录下

> （2）启动tomcat，此war文件会自动解压

> （3）修改WEB-INF下的dubbo.properties文件，注意dubbo.registry.address对应的值需要对应当前使用的Zookeeper的ip地址和端口号

```
dubbo.registry.address=zookeeper://192.168.134.129:2181
dubbo.admin.root.password=root
dubbo.admin.guest.password=guest
```

> （4）重启tomcat

使用

操作步骤：

> （1）访问http://localhost:6080/dubbo-admin-2.6.0/，输入用户名(root)和密码(root)

> （2）启动服务提供者工程和服务消费者工程，可以在查看到对应的信息

图1：在监控中心找到服务菜单

图2：点击下面的服务

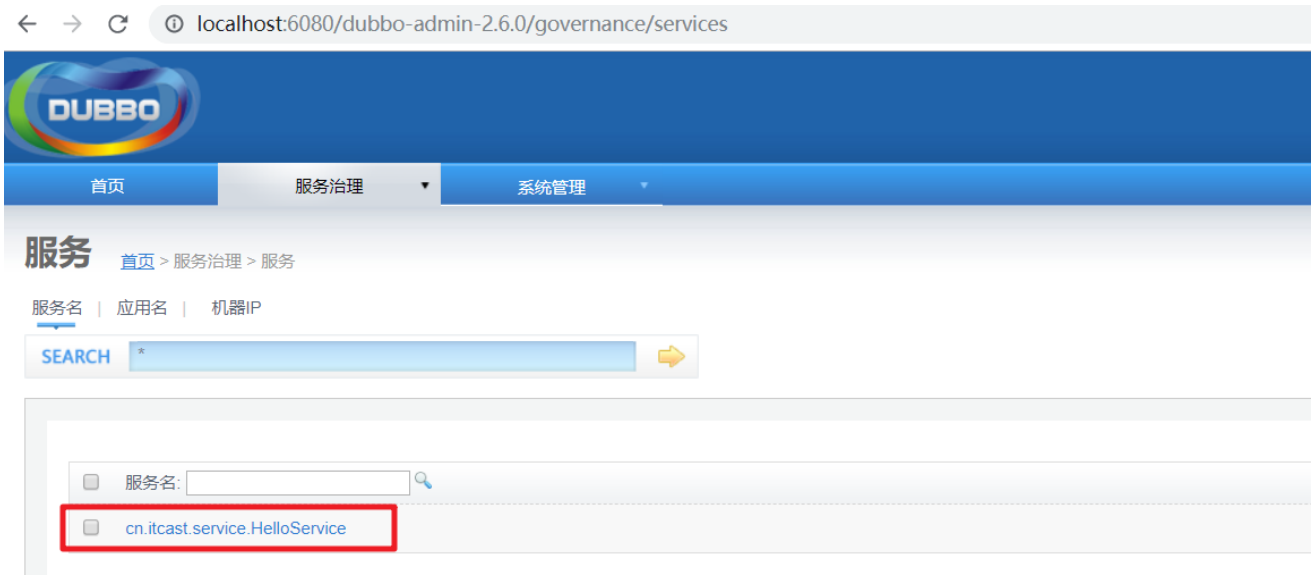


图3：可以查看有哪些服务提供者、消费者

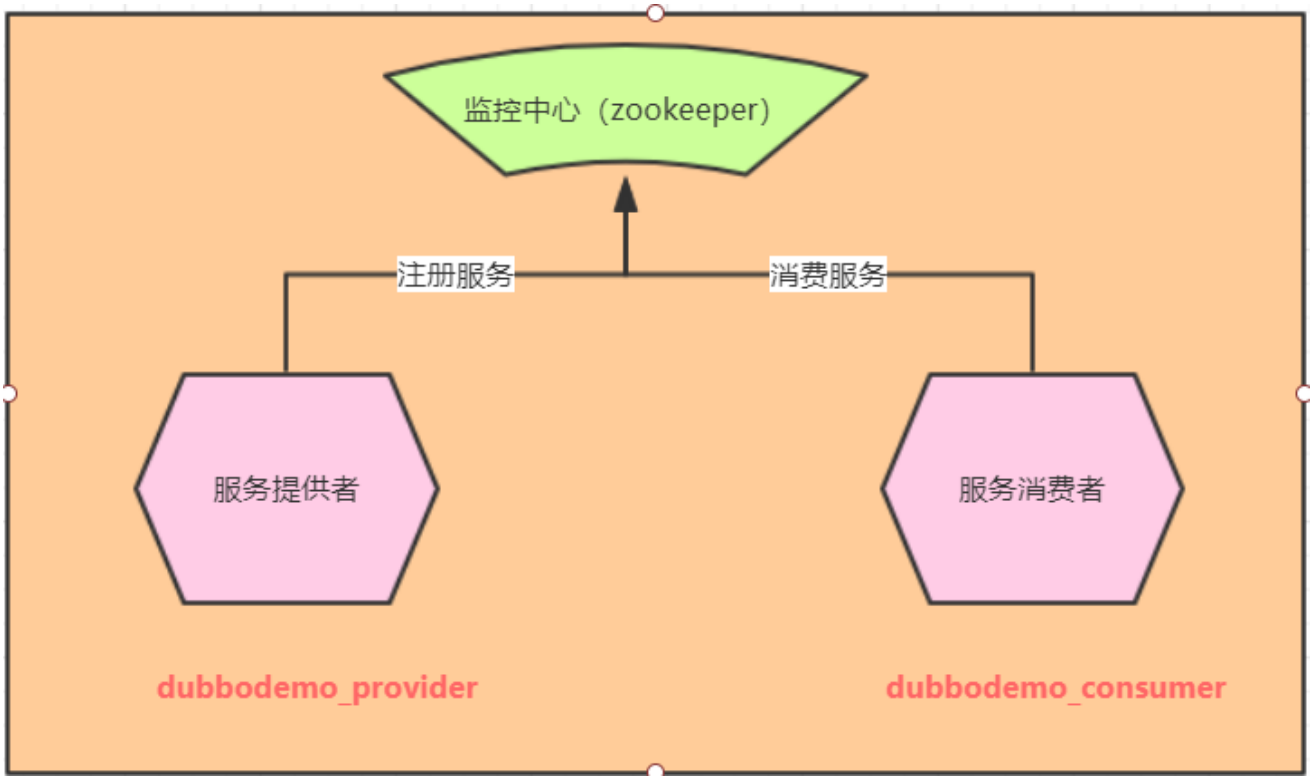
Dubbo 工程模块关系分析、重构入门案例设计

目标

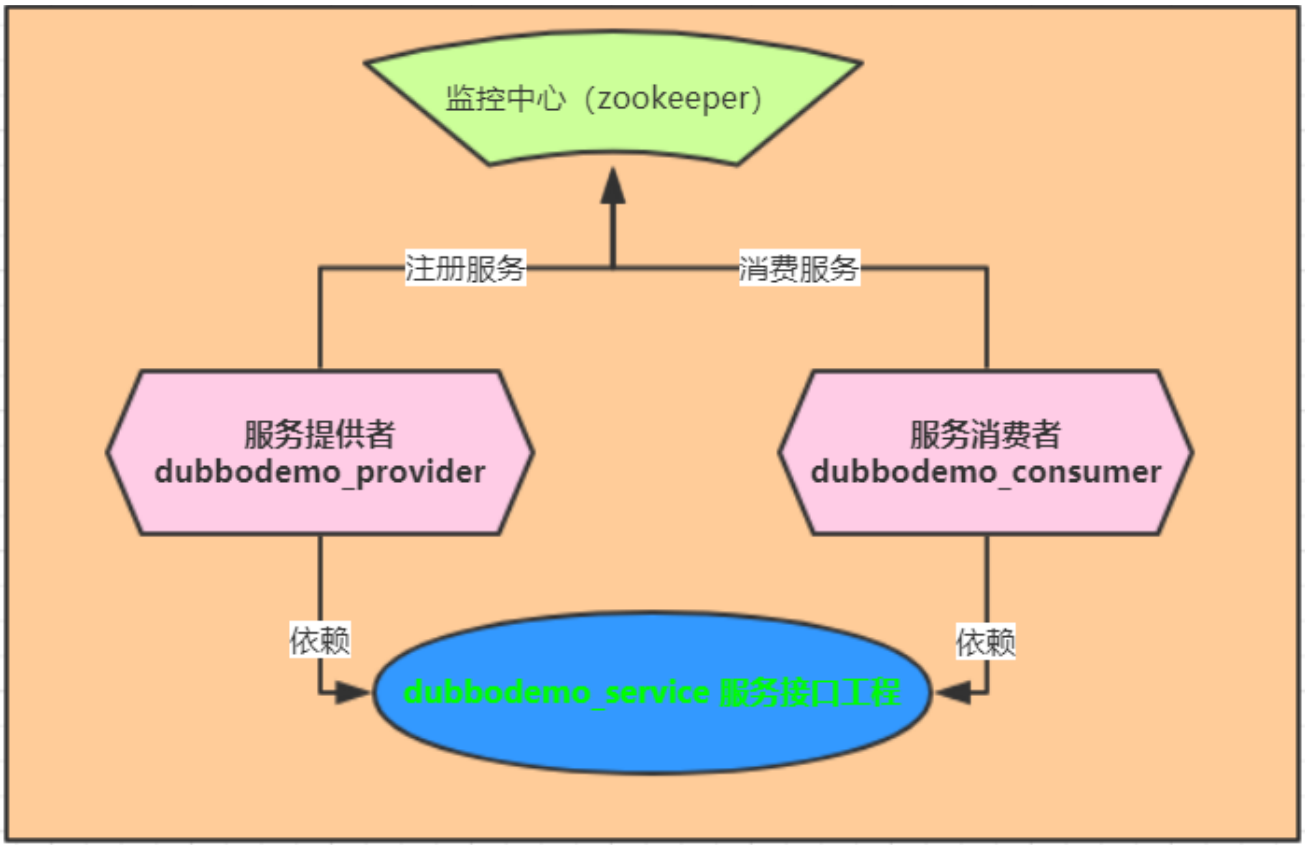
1. 分析Dubbo 工程模块关系
2. 重构入门案例设计

分析Dubbo 工程模块关系

> 目前工程关系为：① 服务提供者，编写服务接口；②服务消费者编写服务接口，调用服务



> 会发现服务提供者与服务消费者都要编写服务接口，这里会存在编码冗余，不利于后期维护，所以现在重构案例的设计如下：



步骤

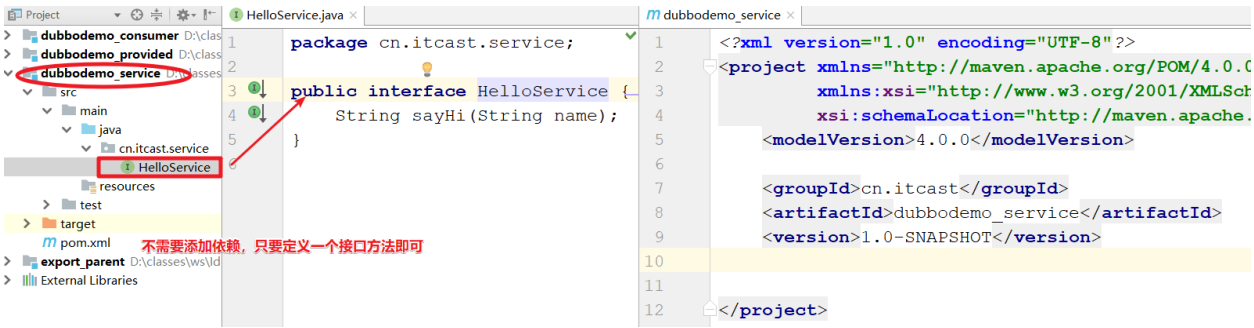
1. 编写服务接口工程：dubbodemo_service
2. 修改dubbodemo_provider服务提供者工程：依赖接口工程、删除service接口

   因为service接口已经在服务接口工程厂统一抽取了，所以这里就不需要写了。
3. 修改dubbodemo_consumer服务消费者工程：依赖接口工程、删除service接口

   因为service接口已经在服务接口工程统一抽取了，所以这里就不需要写了。

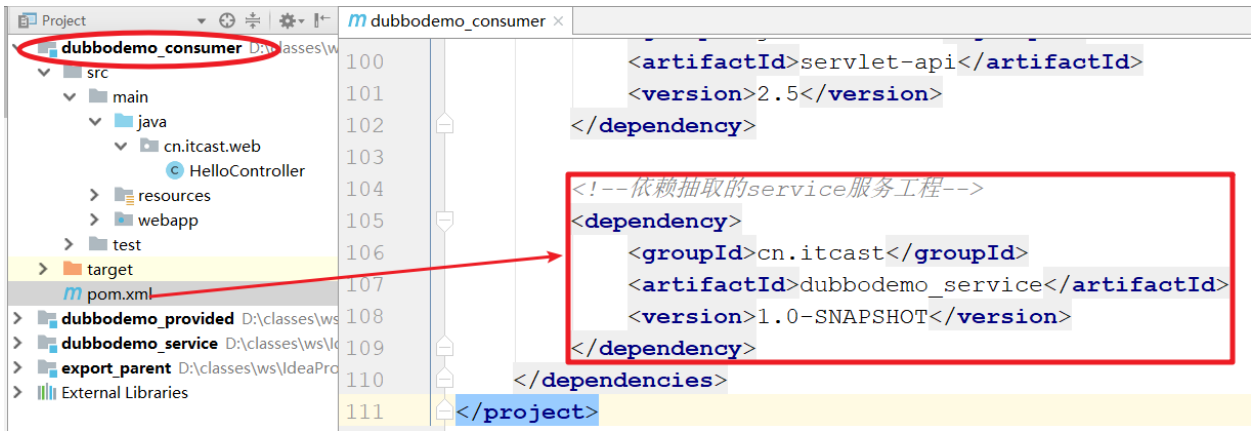
实现

1. 编写服务接口工程：dubbodemo_service



2. 修改dubbodemo_provider服务提供者工程：依赖接口工程、删除service接口

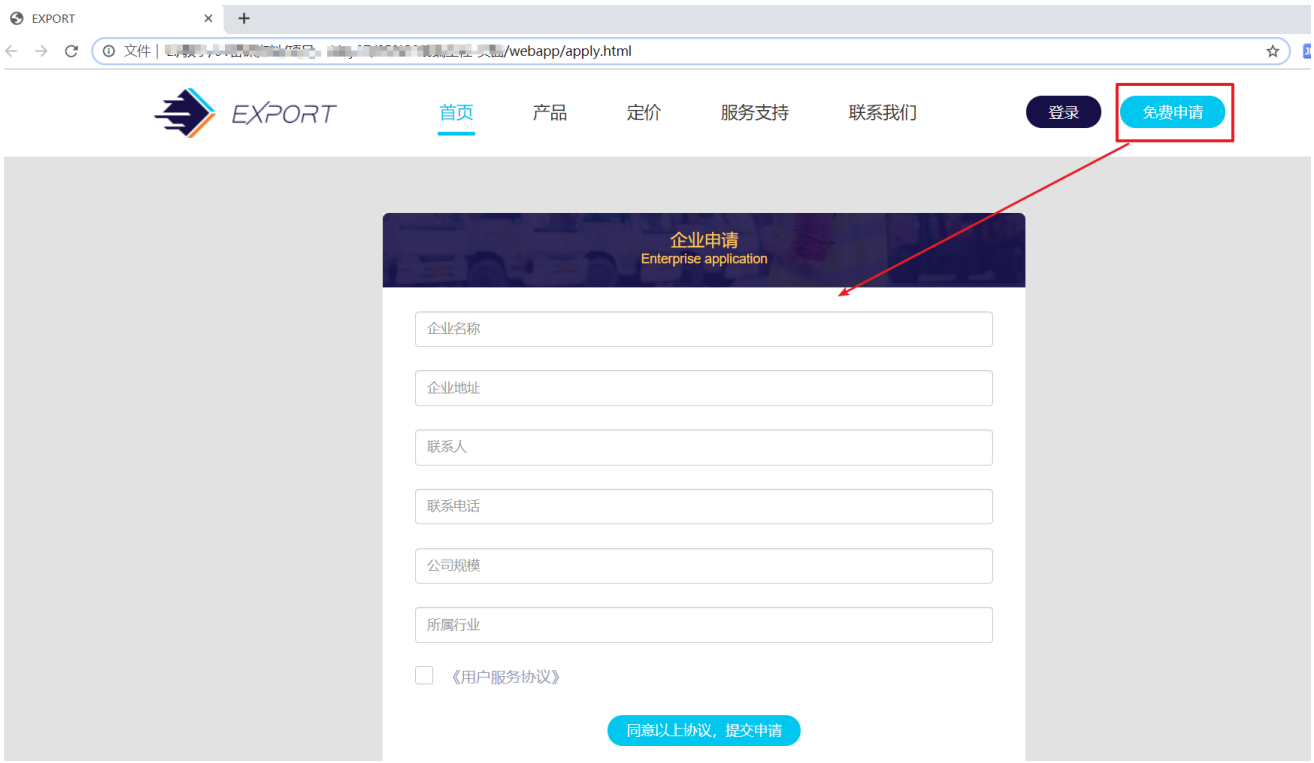
3. 修改dubbodemo_consumer服务消费者工程：依赖接口工程、删除service接口



案例：改造企业管理（一）分析

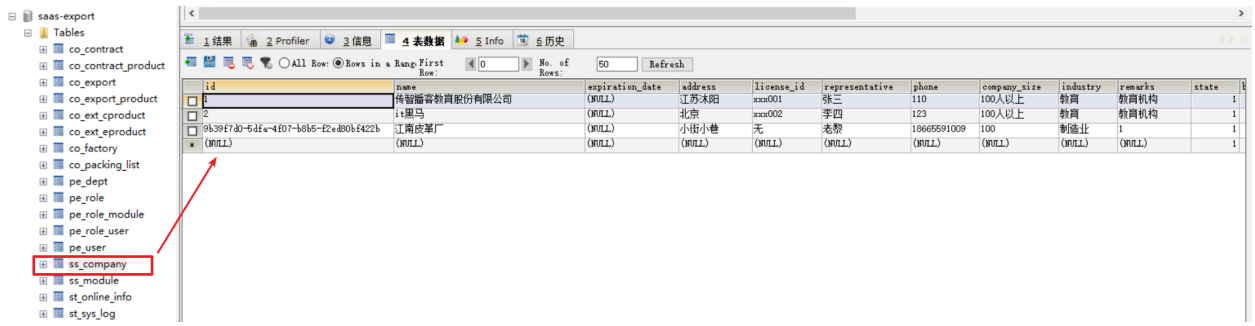
需求

完成网站前端企业申请工作：

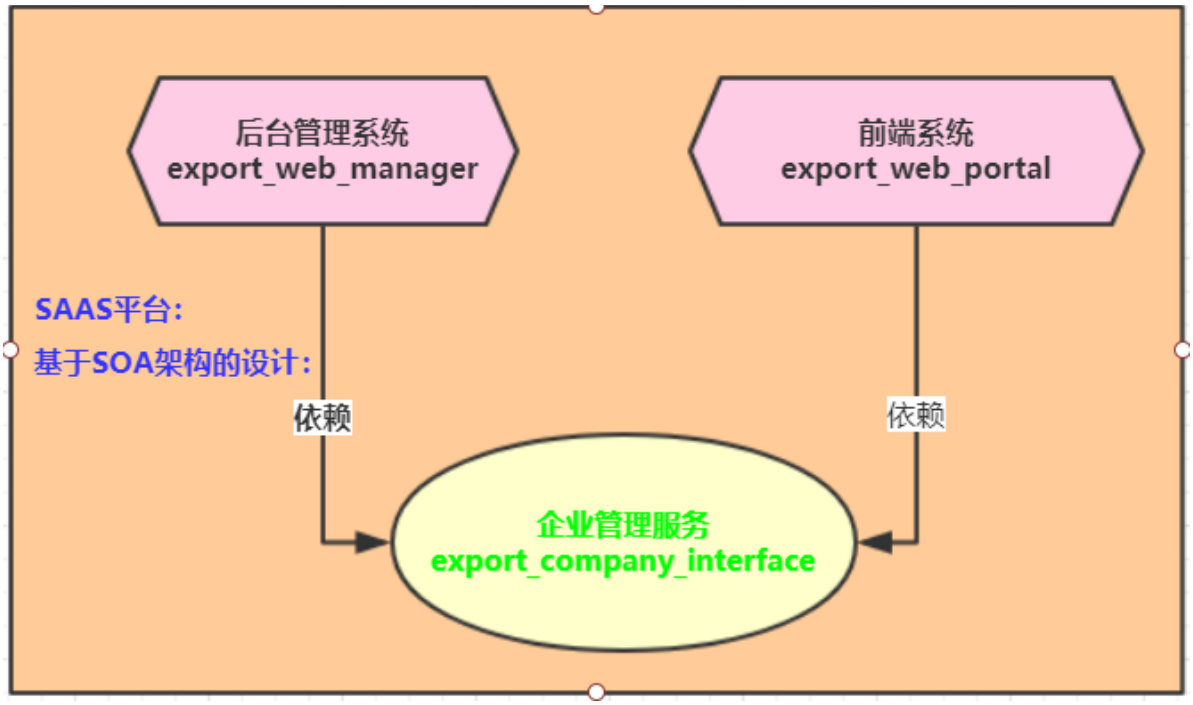


分析

1. 目前，后台系统中已经完成添加企业工程
2. 现在，需要开发前端系统，完成企业入驻申请功能，与后台功能一样。都是往企业表保存数据：



3. 如何解决前端系统企业入驻功能实现?
    1. 方案1： 前端系统自己实现企业入驻，此时需要编写domain、dao、service、controller等。
    2. 方案2：通过公共的dubbo服务完成统一的企业操作



案例：改造企业管理（二）创建Dubbo服务接口工程

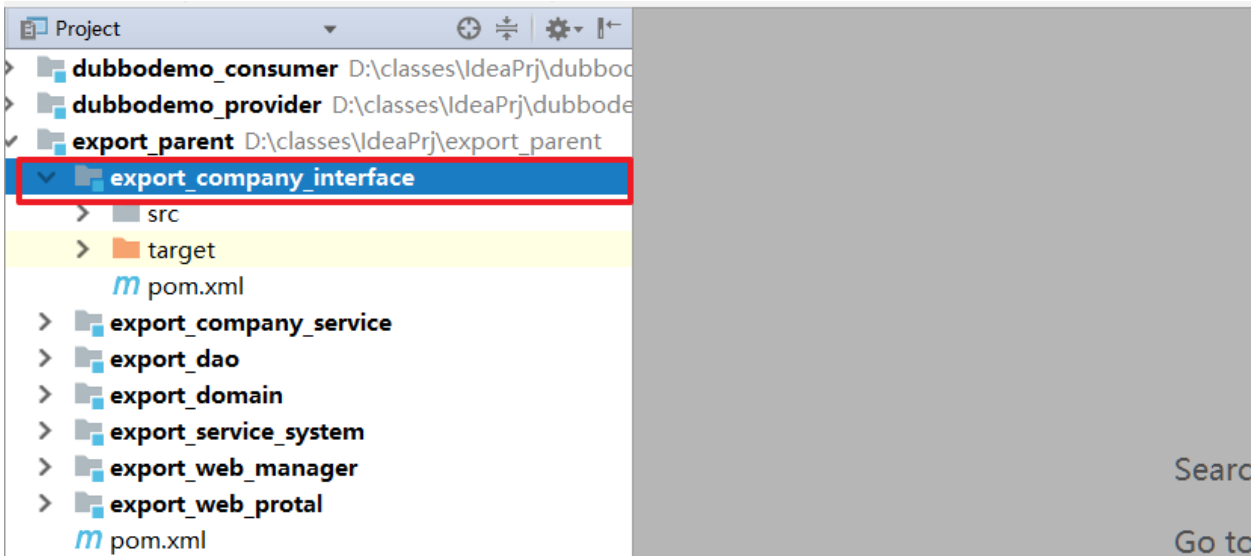
步骤

1. 创建项目：export_company_interface
2. 添加依赖
3. 编写对外发布的服务的接口

实现

1. 创建项目：export_company_interface

   图1:

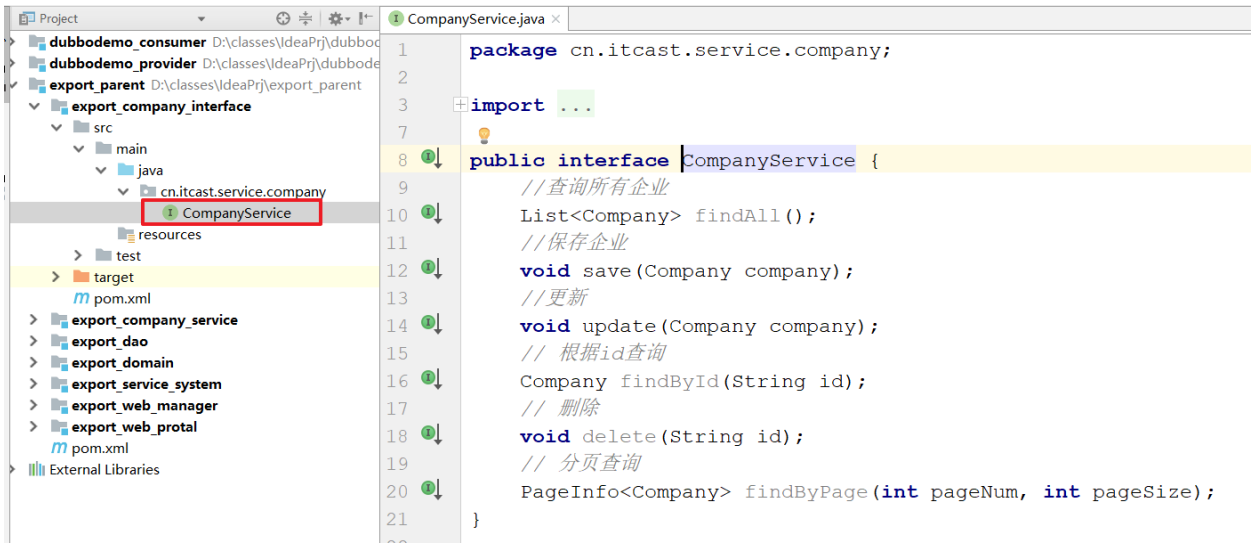


2. 添加依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
    <parent>
        <artifactId>export_parent</artifactId>
        <groupId>cn.itcast</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>export_company_interface</artifactId>

    <dependencies>
        <dependency>
            <groupId>cn.itcast</groupId>
            <artifactId>export_domain</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

3. 编写对外发布的服务的接口



案例：改造企业管理（三）接口实现工程

目标

创建dubbo服务接口实现工程，实现接口。

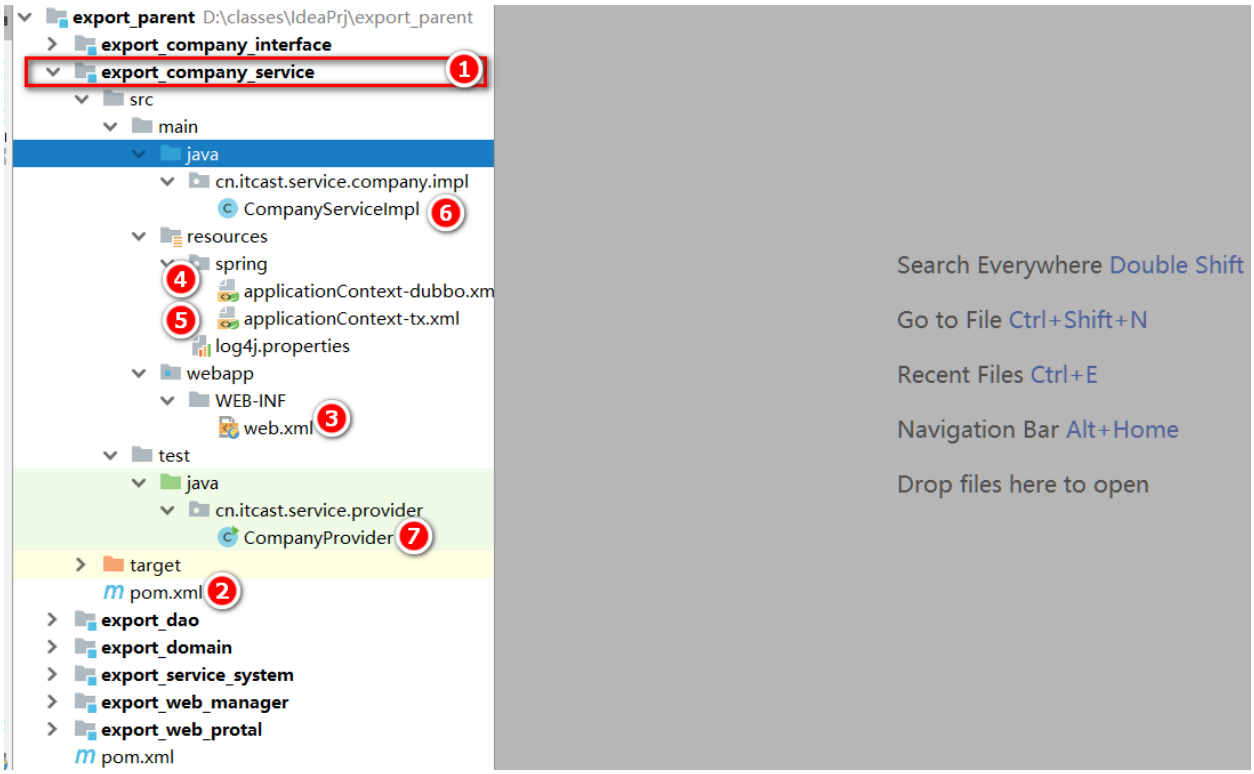
步骤

1. 创建项目：export_company_service
2. 添加依赖：服务接口工程、dao工程、dubbo相关依赖
3. 配置web.xml
4. 配置dubbo：applicationContext-dubbo.xml
5. 配置事务：applicationContext-tx.xml
6. 编写服务接口实现类

7. 通过main函数启动服务

实现

1. 创建项目：export_company_service



2. 添加依赖：依赖服务接口工程、dao工程、dubbo相关

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

<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">
  <parent>
    <artifactId>export_parent</artifactId>
    <groupId>cn.itcast</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>export_company_service</artifactId>
  <packaging>war</packaging>

  <dependencies>
    <!--依赖服务接口-->
    <dependency>
      <groupId>cn.itcast</groupId>
      <artifactId>export_company_interface</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    <!--依赖dao-->
    <dependency>
```

```
      <groupId>cn.itcast</groupId>
      <artifactId>export_dao</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <!--dubbo相关-->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>dubbo</artifactId>
      <version>2.6.6</version>
      <exclusions>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-web</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>io.netty</groupId>
      <artifactId>netty-all</artifactId>
      <version>4.1.32.Final</version>
    </dependency>
    <dependency>
      <groupId>org.apache.curator</groupId>
      <artifactId>curator-framework</artifactId>
      <version>4.0.0</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.zookeeper</groupId>
          <artifactId>zookeeper</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.apache.zookeeper</groupId>
      <artifactId>zookeeper</artifactId>
      <version>3.4.7</version>
    </dependency>
    <dependency>
      <groupId>com.github.sgroschupf</groupId>
      <artifactId>zkclient</artifactId>
      <version>0.1</version>
    </dependency>
  </dependencies>
</project>
```

3. 配置web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

    <!-- 监听器监听其他的spring配置文件 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:spring/applicationContext-*.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
</web-app>
```

4. 配置dubbo：applicationContext-dubbo.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!-- 应用名称：随便写（保证唯一:和项目名一致）  -->
    <dubbo:application name="export_company_service">
        <!--qos 监控 -->
        <dubbo:parameter key="qos.enable" value="false"></dubbo:parameter>
    </dubbo:application>

    <!--配置注册中心  注册中心的地址-->
    <dubbo:registry address="zookeeper://192.168.12.132:2181"></dubbo:registry>

    <!--
        配置请求协议  此dubbo服务的请求端口（和tomcat端无关：不能一致）
            port（端口）：服务提供者的真实请求端口
     -->
    <dubbo:protocol name="dubbo" port="20881"></dubbo:protocol>

    <!--配置dubbo服务提供者的包扫描-->
    <dubbo:annotation package="cn.itcast.service"></dubbo:annotation>
</beans>
```

5. 配置事务：applicationContext-tx.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
       http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
      http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!--包扫描：开启注解支持-->
    <context:component-scan base-package="cn.itcast.service"/>

    <!--配置事务-->
    <!--1.事务管理器-->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <!--2.事务通知-->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="find*" read-only="true"/>
            <tx:method name="save*" read-only="false" propagation="REQUIRED"/>
            <tx:method name="update*" read-only="false"
propagation="REQUIRED"/>
            <tx:method name="delete*" read-only="false"
propagation="REQUIRED"/>
            <tx:method name="*" read-only="false" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>
    <!--3.aop-->
    <aop:config>
        <aop:pointcut id="pt" expression="execution( *
cn.itcast.service.*.impl.*.*(..))"></aop:pointcut>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="pt"></aop:advisor>
    </aop:config>


</beans>
```

6. 编写服务接口实现类

```java
package cn.itcast.service.company.impl;
```

```
2
3  import cn.itcast.dao.company.CompanyDao;
4  import cn.itcast.domain.company.Company;
5  import cn.itcast.service.company.CompanyService;
6  import com.alibaba.dubbo.config.annotation.Service;
7  import com.github.pagehelper.PageHelper;
8  import com.github.pagehelper.PageInfo;
9  import org.springframework.beans.factory.annotation.Autowired;
10
11 import java.util.List;
12 import java.util.UUID;
13
14 // import com.alibaba.dubbo.config.annotation.Service;
15 @Service
16 public class CompanyServiceImpl implements CompanyService {
17
18     @Autowired
19     private CompanyDao companyDao;
20
21     public List<Company> findAll() {
22         return companyDao.findAll();
23     }
24
25     @Override
26     public void save(Company company) {
27         //设置企业id
28         company.setId(UUID.randomUUID().toString());
29         companyDao.save(company);
30     }
31
32     @Override
33     public void update(Company company) {
34         companyDao.update(company);
35     }
36
37     @Override
38     public Company findById(String id) {
39         return companyDao.findById(id);
40     }
41
42     @Override
43     public void delete(String id) {
44         companyDao.delete(id);
45     }
46
47     @Override
48     public PageInfo<Company> findByPage(int pageNum, int pageSize) {
49         // 开始分页，PageHelper组件会自动对其后的第一条查询查询分页
50         PageHelper.startPage(pageNum,pageSize);
51         // 调用dao查询
52         List<Company> list = companyDao.findAll();
53         // 创建PageInfo对象封装分页结果，传入查询集合。会自动计算分页参数
54         PageInfo<Company> pageInfo = new PageInfo<>(list);
```

```
        return pageInfo;
    }
}
```

7. 通过main函数启动服务

```
package cn.itcast.service.provider;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

/**
 * 基于main方法启动提供者
 */
public class CompanyProvider {

   public static void main(String[] args) throws IOException {
      //1.加载配置文件
      ClassPathXmlApplicationContext ac =
            new
ClassPathXmlApplicationContext("classpath*:spring/applicationContext-*.xml");
      //2.启动
      ac.start();
      //3.输入后停止
      System.in.read();
   }
}
```

前端系统部署、企业入驻

目标

实现前端系统部署、企业入驻功能。

步骤

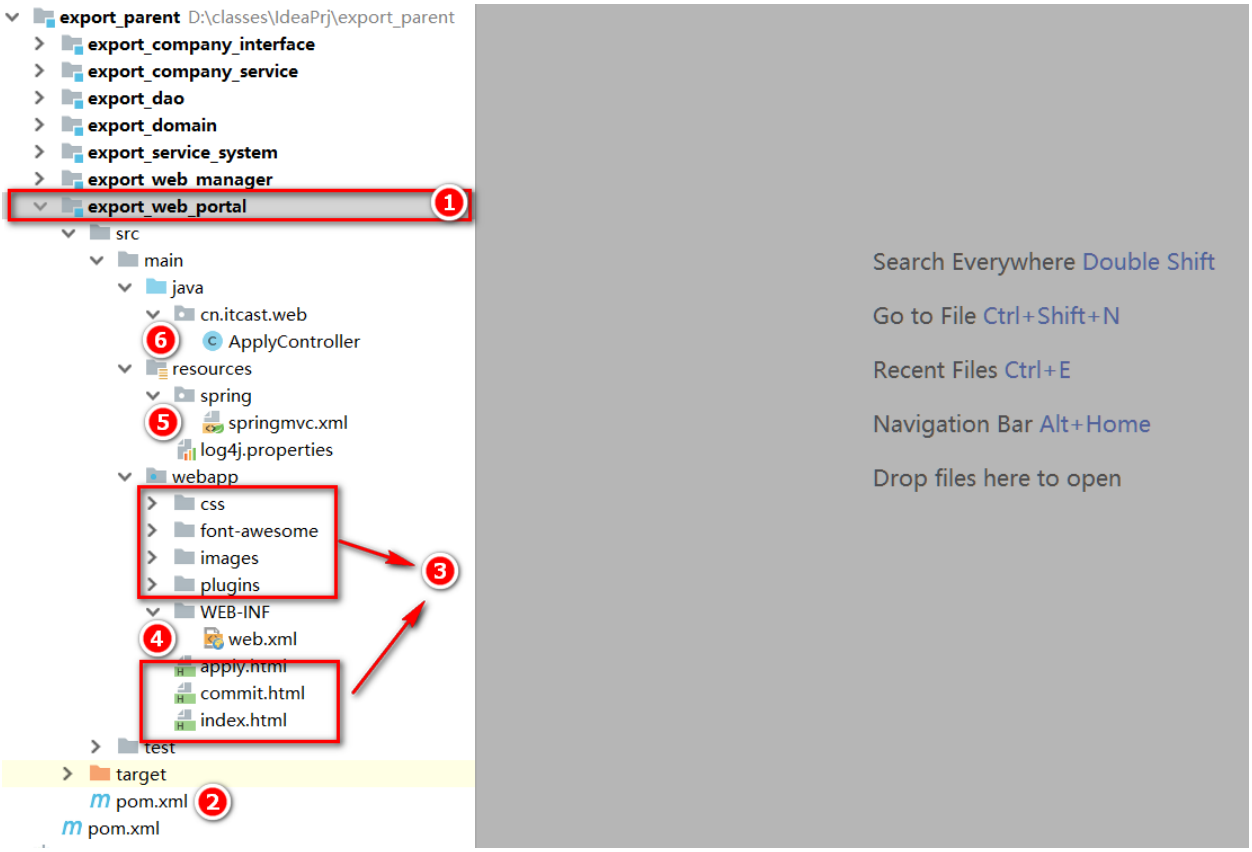
1. 创建项目：export_web_portal
2. 添加依赖： 依赖企业管理service实现工程、依赖dubbo
3. 部署UI资源
4. 配置web.xml
5. 配置springmvc.xml
6. 编写控制器
7. 测试

   易错点：Company一定要实现Serializable接口，否则报错。

```
public class Company implements Serializable{}

dubbo传输的对象如果没有实现可序列化接口，报错：
Caused by: Serialized class Company must implement java.io.Serializable
```

实现

1. 创建项目：export_web_portal



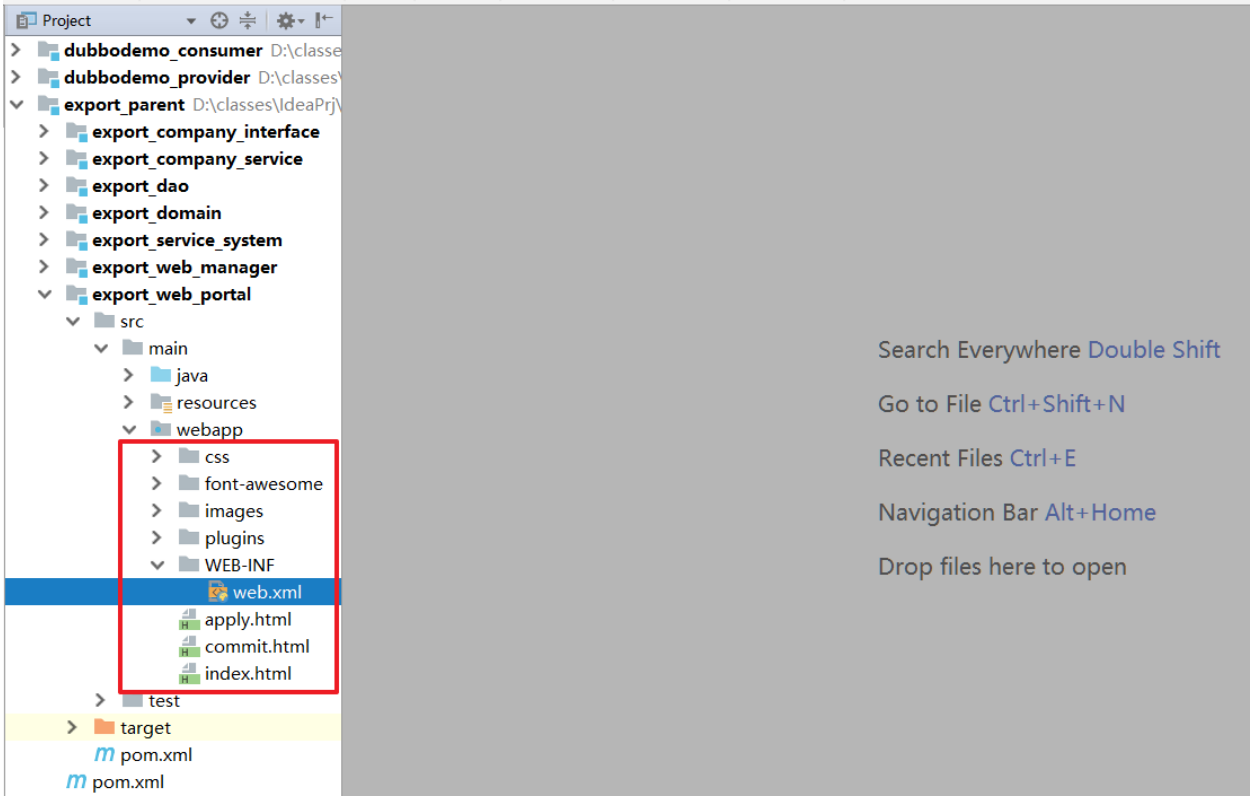
2. 添加依赖： 依赖企业管理service实现工程、依赖dubbo

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

<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">
  <parent>
    <artifactId>export_parent</artifactId>
    <groupId>cn.itcast</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>export_web_portal</artifactId>
  <packaging>war</packaging>


  <dependencies>
    <!--依赖service接口工程-->
```

```
    <dependency>
      <groupId>cn.itcast</groupId>
      <artifactId>export_company_interface</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <!--dubbo支持包-->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>dubbo</artifactId>
      <version>2.6.6</version>
      <exclusions>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-web</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>io.netty</groupId>
      <artifactId>netty-all</artifactId>
      <version>4.1.32.Final</version>
    </dependency>
    <dependency>
      <groupId>org.apache.curator</groupId>
      <artifactId>curator-framework</artifactId>
      <version>4.0.0</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.zookeeper</groupId>
          <artifactId>zookeeper</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.apache.zookeeper</groupId>
      <artifactId>zookeeper</artifactId>
      <version>3.4.7</version>
    </dependency>
    <dependency>
      <groupId>com.github.sgroschupf</groupId>
      <artifactId>zkclient</artifactId>
      <version>0.1</version>
    </dependency>

```

```
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>

  </dependencies>
</project>
```

3. 部署UI资源: 拷贝资料中的ui资源到项目的webapp目录下。



4. 配置web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

    <!-- 解决post乱码 -->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>utf-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
```

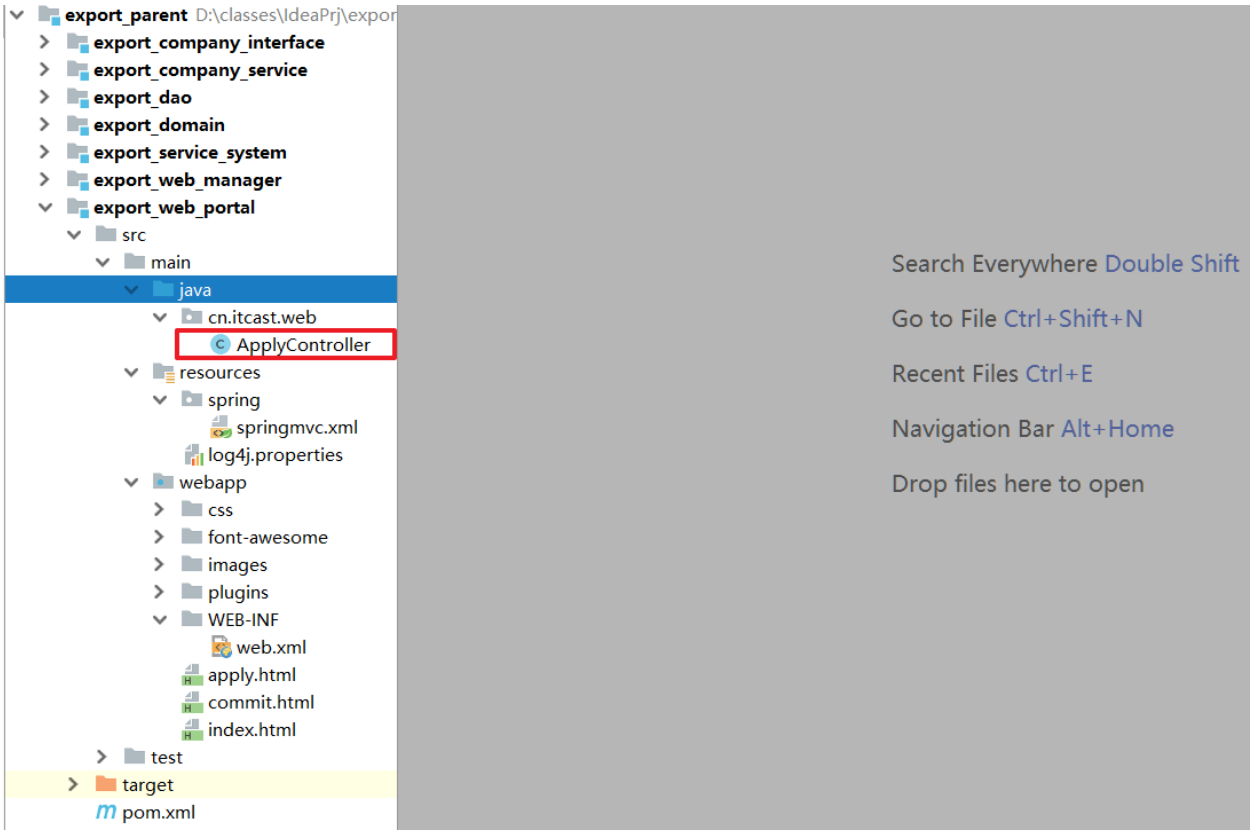
```
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 指定加载的配置文件 ，通过参数contextConfigLocation加载-->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring/springmvc.xml</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

5. 配置springmvc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
   xmlns:mvc="http://www.springframework.org/schema/mvc"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
           http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">


   <!--SpringMVC配置-->
    <context:component-scan base-package="cn.itcast.web"></context:component-
scan>
   <mvc:annotation-driven></mvc:annotation-driven>

   <!--Dubbo配置-->
    <!-- 当前应用名称，用于注册中心计算应用间依赖关系，注意：消费者和提供者应用名不要一样 -->
    <dubbo:application name="export_web_protal" />
    <!-- 连接服务注册中心zookeeper ip为zookeeper所在服务器的ip地址-->
    <dubbo:registry address="zookeeper://192.168.12.132:2181"/>
    <!-- 开启dubbo注解扫描(@Reference注解)-->
```

```
    <dubbo:annotation package="cn.itcast.web"/>
</beans>
```

6. 编写控制器



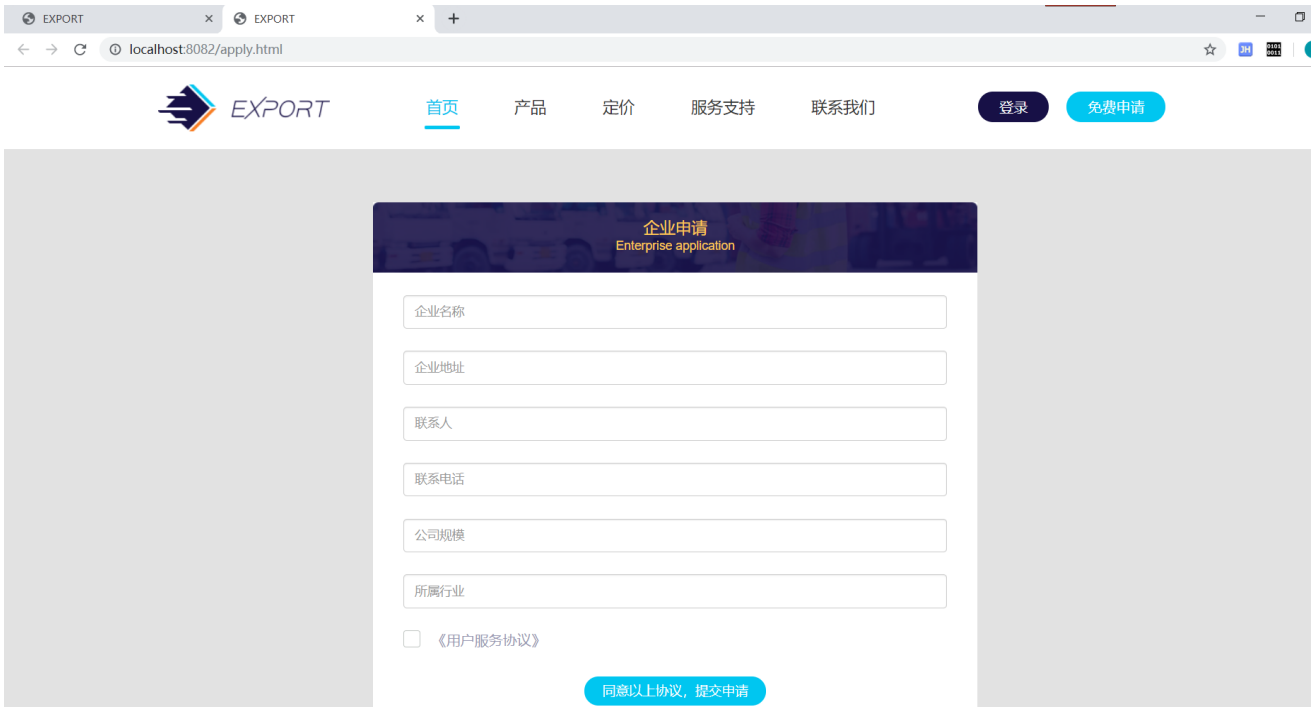

```java
package cn.itcast.web;

import cn.itcast.domain.company.Company;
import cn.itcast.service.company.CompanyService;
import com.alibaba.dubbo.config.annotation.Reference;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class ApplyController {

    /**
     * @Reference(retries = 0)
     *      retries:配置重试次数。
     */
    @Reference(retries = 0)
    private CompanyService companyService;

    /**
     * 企业入驻申请，保存
     */
    @RequestMapping("/apply")
    @ResponseBody
    public String apply(Company company) {
```

```
26          try{
27              company.setState(0);
28              companyService.save(company);
29              return "1";
30          }catch (Exception e) {
31              e.printStackTrace();
32              return "2";
33          }
34      }
35  }
```

7. 测试

第一步：访问前端系统首页

第二步：点击免费申请



第三步： 填写数据，保存。保存后的页面如下：


EXPORT
localhost:8082/commit.html
EXPORT
首页
产品
定价
服务支持
联系我们
登录
免费申请
1
提交资料
2
审核
3
完成
资料已提交成功，我们将尽快安排为您审核，请您耐心等待!
如长时间未审核通过您可以联系您的业务员进行开通或者联系销售助理进行开通
联系电话：400-618-4000