

虽然之前也有记录简单的分布式架构搭建，但是可能不太全面，本篇以一个实际项目搭建分布式架构力求能稍微详细地介绍分布式架构搭建中的细节。

先介绍下项目，我之家，一个集合资源管理，小功能大全，便利工具大全等等需求的一个为个人服务的项目。

目前项目初步定为这些模块：web模块（前台），admin模块（后台），function模块（小功能大全），resource模块（资源管理），plugin模块（便利工具），用户模块（client），以及调用的公共模块base。

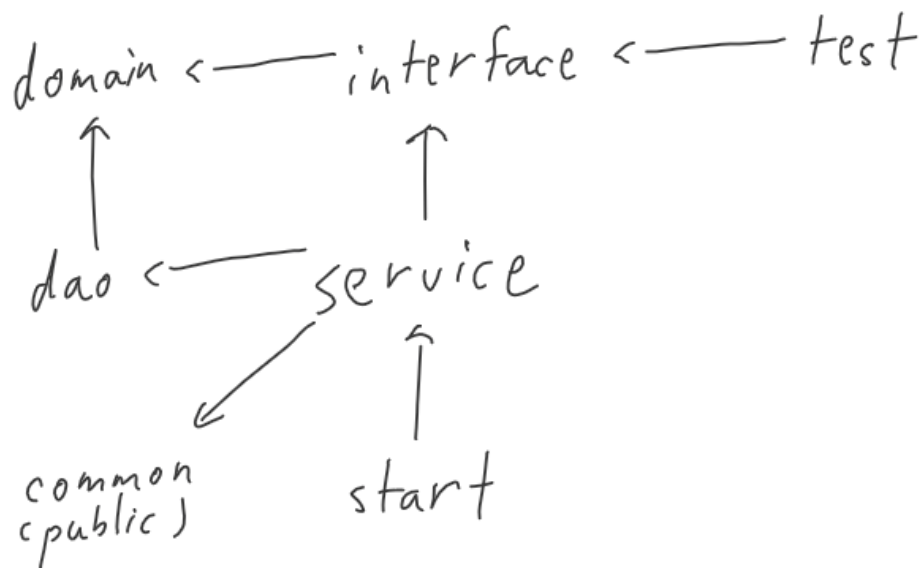
本篇以其中function和web为例讲解，正好一个是服务提供者，一个是服务消费者。

先讲function，目前它的地位对于web来说在dubbo中属于提供者，容我从创建项目开始慢慢谈。

首先先上个项目结构

```
▼ <img alt="IDE icon" data-bbox="125 515 145 530"/> > function [function master]
  > <img alt="Folder icon" data-bbox="145 530 165 545"/> > function-dao
  > <img alt="Folder icon" data-bbox="145 545 165 560"/> > function-domain
  > <img alt="Folder icon" data-bbox="145 560 165 575"/> > function-interface
  > <img alt="Folder icon" data-bbox="145 575 165 590"/> > function-service
  > <img alt="Folder icon" data-bbox="145 590 165 605"/> > function-start
  > <img alt="Folder icon" data-bbox="145 605 165 620"/> > function-test
  <img alt="File icon" data-bbox="145 625 165 640"/> pom.xml
  <img alt="File icon" data-bbox="145 640 165 655"/> README.md
> <img alt="Maven icon" data-bbox="125 660 145 675"/> function-dao
> <img alt="Maven icon" data-bbox="125 675 145 690"/> function-domain
> <img alt="Maven icon" data-bbox="125 690 145 705"/> function-interface
> <img alt="Maven icon" data-bbox="125 705 145 720"/> function-service
> <img alt="Maven icon" data-bbox="125 720 145 735"/> function-start [boot]
> <img alt="Maven icon" data-bbox="125 735 145 750"/> function-test [boot]
```

function是一个标准的maven聚合项目，其它都是它的各个子模块，下面上一张手绘的依赖图，理解一下。



介绍一下这几个模块：

interface：服务接口，也是对外的接口

service：服务的实现

domain：实体类以及中间类

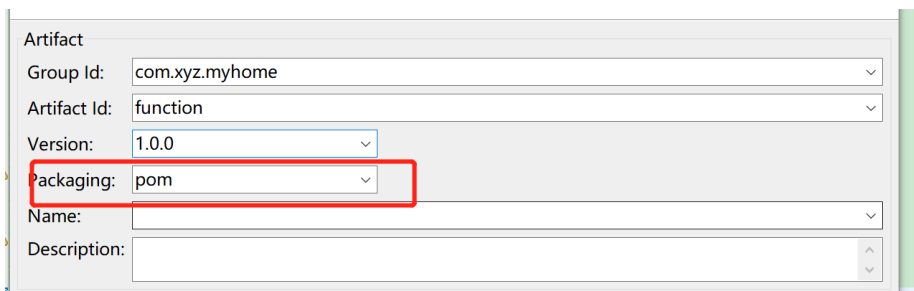
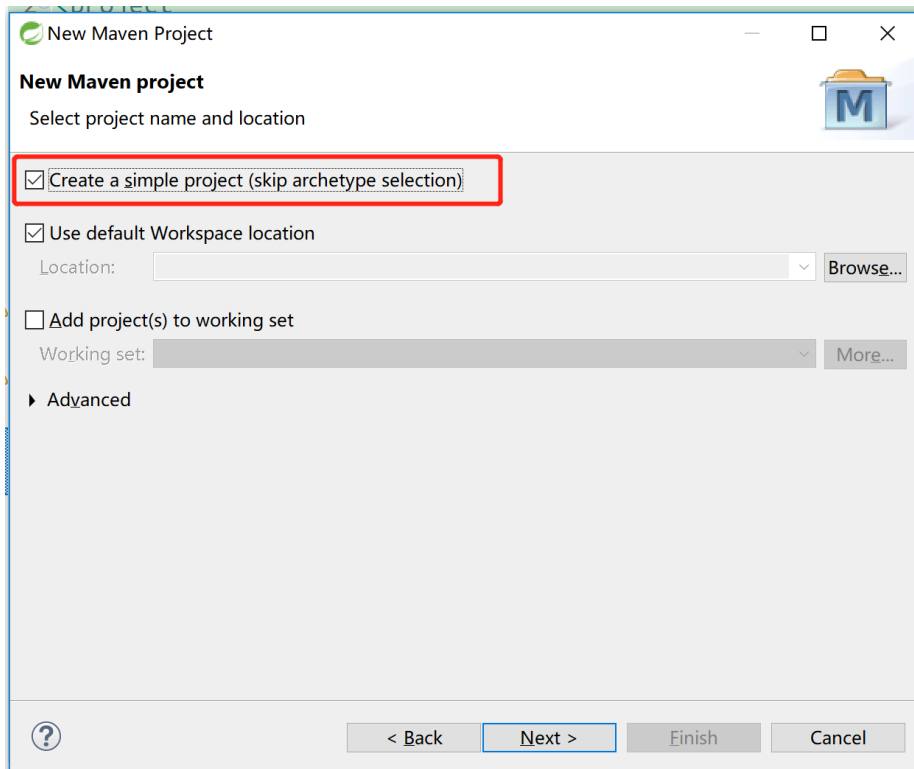
dao：数据访问层

start：function启动

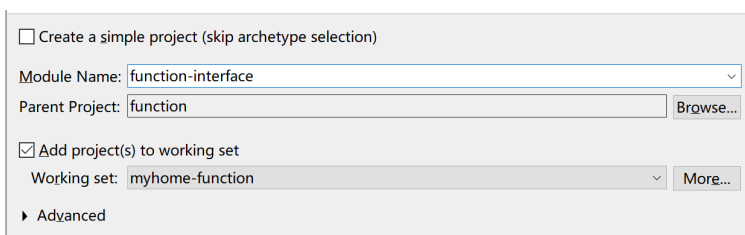
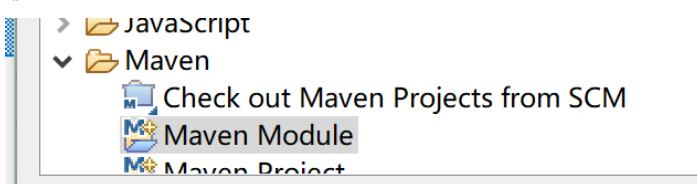
test：测试

common：通用，放在base（public）模块中

首先创建作为父级项目的function



再对function创建模块，以interface为例（有一点还是不清楚，有时候新建模块后，父级项目的pom里会被清空，建议每次新建模块先备份下）



org.apache.maven.archetypes	maven-archetype-promises	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1

讲解实例就以目前最新的一个需求“游戏梦想储蓄罐”为例，详解各个子模块的工作。

首先我们展示一下dao层和domain层。

```
▼ function-dao [boot]
  ▼ src/main/java
    ▼ com.xyz.myhome.dao
      ▼ extend
        > GdbboxMapper.java
        > GdbboxMainMapper.xml
      ▼ generator
        > GdbboxJournalMapper.java
        > GdbboxMainMapper.java
        > GdbboxJournalMapper.xml
        > GdbboxMainMapper.xml
```

```
<?xml version="1.0"?>
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>function</artifactId>
    <version>1.0.0</version>
  </parent>
  <groupId>com.xyz.myhome</groupId>
  <artifactId>function-dao</artifactId>
  <version>1.0.0</version>
  <name>function-dao</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.xyz.myhome</groupId>
      <artifactId>function-domain</artifactId>
      <version>1.0.0</version>
    </dependency>

    <!-- Spring boot整合mybatis包 -->
    <dependency>
      <groupId>org.mybatis.spring.boot</groupId>
      <artifactId>mybatis-spring-boot-starter</artifactId>
      <version>1.3.2</version>
    </dependency>
```

```

        <!-- MySQL JDBC驱动包 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
    </dependencies>
</project>

```

generator里面存放自动生成的mapper，extend中存放扩展的自己写的mapper。

```

v myhome-function [function master]
  > function [function master]
  > function-dao [boot]
  v function-domain
    v src/main/java
      v com.xyz.myhome
        v domain
          > GdboxJournal.java
          > GdboxJournalExample.java
          > GdboxMain.java
          > GdboxMainExample.java
        v dto
          v request
            > GdboxRequest.java
          v response
            > GdboxResponse.java
      src/test/java
    > JRE System Library [JavaSE-1.8]
    > Maven Dependencies
    > src
    > target
    > pom.xml

```

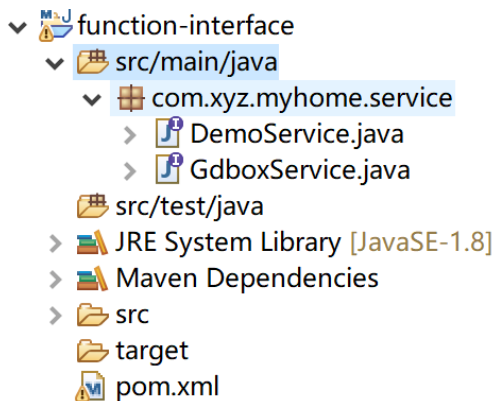
```

<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.xyz.myhome</groupId>
        <artifactId>function</artifactId>
        <version>1.0.0</version>
    </parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>function-domain</artifactId>
    <version>1.0.0</version>
    <name>function-domain</name>
    <url>http://maven.apache.org</url>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
    </dependencies>
</project>












```

domain中存放自动生成的实体类，dto里面存放由具体需求决定的实体类。

下面展示对外的接口层interface和服务的实现层service



```
<?xml version="1.0"?>
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>function</artifactId>
    <version>1.0.0</version>
  </parent>
  <groupId>com.xyz.myhome</groupId>
  <artifactId>function-interface</artifactId>
  <version>1.0.0</version>
  <name>function-interface</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.xyz.myhome</groupId>
      <artifactId>function-domain</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
</project>
```

- ▼  function-service [boot]
  - ▼  src/main/java
    - ▼  com.xyz.myhome.service.impl
      - >  DemoServiceImpl.java
      - >  GdboxServiceImpl.java
    -  src/test/java
    - >  JRE System Library [JavaSE-1.8]
    - >  Maven Dependencies
    - >  src
    -  target
    -  pom.xml

```
<?xml version="1.0"?>
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>function</artifactId>
    <version>1.0.0</version>
  </parent>
  <groupId>com.xyz.myhome</groupId>
  <artifactId>function-service</artifactId>
  <version>1.0.0</version>
  <name>function-service</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.xyz.myhome</groupId>
      <artifactId>function-interface</artifactId>
      <version>1.0.0</version>
    </dependency>


















    <dependency>
      <groupId>com.xyz.myhome</groupId>
      <artifactId>function-dao</artifactId>
      <version>1.0.0</version>
    </dependency>

    <dependency>
      <groupId>com.xyz.public</groupId>
      <artifactId>base-common</artifactId>
      <version>1.0.0</version>
    </dependency>

  </dependencies>
</project>
```

这两个层就可以理解为以前的service接口和service实现类。

接下来就是系统的启动层。

- ▼  function-start [boot]
  - ▼  src/main/java
    - ▼  com.xyz.myhome
      - >  logtest
      - ▼  starter
        - >  ApplicationStarter.java
  - ▼  src/main/resources
    -  application.properties
    -  dubbo-service.xml
    -  logback.xml
  -  src/test/java
  - >  JRE System Library [JavaSE-1.8]
  - >  Maven Dependencies
  -  log
  - >  src
  -  target
  -  pom.xml

```
<?xml version="1.0"?>
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>function</artifactId>
    <version>1.0.0</version>
  </parent>
  <groupId>com.xyz.myhome</groupId>
  <artifactId>function-start</artifactId>
  <version>1.0.0</version>
  <name>function-start</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.xyz.myhome</groupId>
      <artifactId>function-service</artifactId>
      <version>1.0.0</version>
    </dependency>

    <!-- Spring boot启动支持 -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
```



```

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

    <!-- Spring boot 集成dubbo -->
    <dependency>
        <groupId>com.alibaba.spring.boot</groupId>
        <artifactId>dubbo-spring-boot-starter</artifactId>
        <version>1.0.0</version>
    </dependency>

    <!-- zookeeper 客户端jar -->
    <dependency>
        <groupId>com.101tec</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.10</version>
    </dependency>
</dependencies>
</project>

```

基本上这一个聚合项目的主体就这些了，测试层待会单独讲。

接下来我以一个具体的业务流程叙述一下。

业务流程描述：web传递参数返回相关的数据列表。

```

package com.xyz.myhome.service.impl;

import java.util.Date;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.xyz.myhome.dao.extend.GdboxMapper;
import com.xyz.myhome.dao.generator.GdboxJournalMapper;
import com.xyz.myhome.dao.generator.GdboxMainMapper;
import com.xyz.myhome.domain.GdboxMain;
import com.xyz.myhome.dto.request.GdboxRequest;
import com.xyz.myhome.dto.response.GdboxResponse;
import com.xyz.myhome.service.GdboxService;

public class GdboxServiceImpl implements GdboxService {

    @Autowired
    private GdboxMainMapper gdboxMainMapper;

    @Autowired
    private GdboxJournalMapper gdboxJournalMapper;

    @Autowired
    private GdboxMapper gdboxMapper;

    @Override
    public List<GdboxResponse> getDetail(GdboxRequest gdboxRequest) {

```

```

        Long id = gdboxRequest.getMid();

        List<GdboxResponse> list = gdboxMapper.selectDetail(id);

        return list;
    }

    @Override
    public boolean changeBal(GdboxRequest gdboxRequest) {

        Long id = gdboxRequest.getMid();
        Integer balance = gdboxRequest.getBalance();

        GdboxMain gdboxMain = new GdboxMain();
        gdboxMain.setId(id);
        gdboxMain.setBalance(balance);
        gdboxMain.setUptime(new Date());

        return gdboxMainMapper.updateByPrimaryKey(gdboxMain) > 0;
    }

    @Override
    public boolean changeListing(GdboxRequest gdboxRequest) {

        Long id = gdboxRequest.getMid();
        String listing = gdboxRequest.getListing();

        GdboxMain gdboxMain = new GdboxMain();
        gdboxMain.setId(id);
        gdboxMain.setListing(listing);
        gdboxMain.setUptime(new Date());

        return gdboxMainMapper.updateByPrimaryKey(gdboxMain) > 0;
    }
}

```

可以看到没有用注解表明service身份，因为会在dubbo配置文件中注册为bean。

我们看看启动类。

```

package com.xyz.myhome.starter;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ImportResource;

/**

```

```

* Spring boot启动类（未配置的前提下必须放在需要扫描的类的上级或平级目录,或者想这里手动配置扫描）
*
* @author pc
*
*/
@SpringBootApplication // 该类转型为Spring boot启动类
@ImportResource(value = { "classpath:dubbo-service.xml" })
@MapperScan("com.xyz.myhome.dao") //扫描该路径注册为mapper
public class ApplicationStarter {

    public static void main(String[] args) {
        SpringApplication.run(ApplicationStarter.class, args);
        System.out.println("function服务启动成功.....");
    }
}

```

三行注解第一行和第三行就不说了，第三行提一下，start引入了service依赖自然也引入了dao依赖，也就是start中是可以找到这个路径下的mapper的。

第二行自然就是我们的主角，引入dubbo配置文件，配置文件长啥样我们来瞧瞧。

```

<?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">
    <!-- 配置可参考 http://dubbo.io/User+Guide-zh.htm -->
    <!-- 服务提供方应用名，用于计算依赖关系 -->
    <dubbo:application name="myhome-function" owner="myhome-function" />
    <!-- 定义 zookeeper 注册中心地址及协议 -->
    <dubbo:registry protocol="zookeeper" address="127.0.0.1:2181"
        client="zkclient" />
    <!-- 定义 Dubbo 协议名称及使用的端口，dubbo 协议缺省端口为 20880，如果配置为 -1 或者没有配置 port，则会分配一个没有被占用的端口 -->
    <dubbo:protocol name="dubbo" port="-1" />

    <!-- 声明需要暴露的服务接口 -->
    <dubbo:service interface="com.xyz.myhome.service.DemoService"
        ref="demoService" timeout="10000" />
    <!-- 和本地 bean 一样实现服务 -->
    <bean id="demoService" class="com.xyz.myhome.service.impl.DemoServiceImpl" />

    <dubbo:service interface="com.xyz.myhome.service.GdboxService"

```

```
        ref="gdboxService" timeout="10000" />
        <bean id="gdboxService" class="com.xyz.myhome.service.impl.GdboxServiceImpl" />
    </beans>
```

主体看下面几行注册服务的，很显然，将我们的GdboxServiceImpl注册为了bean，并且dubbo将GdboxService接口指向该实现类，之后通过该接口即可使用实现类。

现在来讲讲我们的test层，这层存在的必要就是各模块自己的业务在不去对接web的情况下先自己测试，因为在dubbo环境中，实现它的难点主要还在如何对接注册在zookeeper上的function服务。

先展示一下。

```
function-test [boot]
├── src/main/java
├── src/test/resources
│   └── dubbo.xml
├── src/test/java
│   └── com.xyz.myhome.function
│       ├── BaseTest.java
│       ├── DemoServiceTest.java
│       └── GdboxServiceTest.java
├── JRE System Library [JavaSE-1.8]
├── Maven Dependencies
├── src
├── target
├── test-output
└── pom.xml
```

```
<dependencies>
    <dependency>
        <groupId>com.xyz.myhome</groupId>
        <artifactId>function-interface</artifactId>
        <version>1.0.0</version>
    </dependency>

    <!-- Spring boot 集成dubbo -->
    <dependency>
        <groupId>com.alibaba.spring.boot</groupId>
        <artifactId>dubbo-spring-boot-starter</artifactId>
        <version>1.0.0</version>
    </dependency>

    <!-- zookeeper 客户端jar -->
    <dependency>
        <groupId>com.101tec</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.10</version>
    </dependency>
</dependencies>
```

```

</dependency>

<!-- Spring boot 单元测试相关 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

</dependencies>

```

以GdboxServiceTest为例。

```

package com.xyz.myhome.function;

import java.util.List;

import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;

import com.xyz.myhome.dto.request.GdboxRequest;
import com.xyz.myhome.dto.response.GdboxResponse;
import com.xyz.myhome.service.GdboxService;

public class GdboxServiceTest extends BaseTest{

    @Autowired
    private GdboxService gdboxService;

    @Test
    public void testGetDetail() {
        GdboxRequest gdboxRequest = new GdboxRequest();
        gdboxRequest.setMid(1L);

        List<GdboxResponse> list = gdboxService.getDetail(gdboxRequest);
        System.out.println(list.size());
    }
}

```

看似不能理解这个是怎么工作的，玄机就在继承的BaseTest。

```

package com.xyz.myhome.function;

import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:dubbo.xml"})
public class BaseTest {

```

```
}
```

这两个注解将测试类与dubbo挂钩了，我们去看看dubbo的配置文件。

```
<?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" >
  <!-- 配置可参考 http://dubbo.io/User+Guide-zh.htm -->
  <!-- 服务提供方应用名，用于计算依赖关系 -->
  <dubbo:application name="myhome_function_test" owner="myhome_function_test" />

  <!-- 定义 zookeeper 注册中心地址及协议 -->
  <dubbo:registry protocol="zookeeper" address="127.0.0.1:2181"
    client="zkclient" />

  <dubbo:reference id="demoService"
interface="com.xyz.myhome.service.DemoService" />

  <dubbo:reference id="gdboxService"
interface="com.xyz.myhome.service.GdboxService" />
</beans>
```



























可以看到这边就引入了GdboxService，即刚刚注册的接口。这样测试类就与dubbo挂钩并且能够调用function中的具体服务了。

接下来我们就长话短说直接开始介绍我们的消费者web，只有两个子模块web-controller和web-starter。

```
▼ > web [web master]
  > > web-controller
  > > web-starter
  > > pom.xml
  > > README.md
  > > web-controller [boot]
  > > web-starter [boot]
```

web-controller负责视图渲染以及服务调用，web-starter负责服务的启动。

先来讲web-controller

- ▼  web-controller [boot]
  - ▼  src/main/java
    - ▼  com.xyz.myhome.controller
      - >  function
      -  plugins
      - >  PageController.java
  - ▼  src/main/resources
    - ▼  static
      - >  bootstrap
      - >  css
      - >  demo
        -  doc
      - >  img
      - >  js
      - >  page
      - >  third
    - ▼  templates
      - ▼  views
        - >  function
        -  demo.html
  -  src/test/java
  - >  JRE System Library [JavaSE-1.8]
  - >  Maven Dependencies
  - >  src
  -  target
  -  pom.xml

```
<?xml version="1.0"?>
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>web</artifactId>
    <version>1.0.0</version>
  </parent>
  <groupId>com.xyz.myhome</groupId>
  <artifactId>web-controller</artifactId>
  <version>1.0.0</version>
  <name>web-controller</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <!-- thymeleaf -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- Spring boot 集成dubbo -->
    <dependency>
      <groupId>com.alibaba.spring.boot</groupId>
```

```

        <artifactId>dubbo-spring-boot-starter</artifactId>
        <version>1.0.0</version>
    </dependency>

    <!-- zookeeper 客户端jar -->
    <dependency>
        <groupId>com.101tec</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.10</version>
    </dependency>

    <!-- 接入function的接口 -->
    <dependency>
        <groupId>com.xyz.myhome</groupId>
        <artifactId>function-interface</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>

<build>
    <finalName>web-controller</finalName>
    <plugins>
    </plugins>
</build>
</project>

```

这边视图渲染采用了thymeleaf模板，具体的项目结构见上图，针对resources下的static和templates两个文件夹的作用不在本篇幅详述，其它也没啥讲的，详细配置也在starter中。

### web-starter

```

v web-starter [boot]
v src/main/java
  v com.xyz.myhome
    > config
    > starter
v src/main/resources
  application.properties
  dubbo.xml
  src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > src
  target
  pom.xml

```

```

<?xml version="1.0"?>
<project
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"

```



```
    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.xyz.myhome</groupId>
    <artifactId>web</artifactId>
    <version>1.0.0</version>
  </parent>
  <groupId>com.xyz.myhome</groupId>
  <artifactId>web-starter</artifactId>
  <version>1.0.0</version>
  <name>web-starter</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <!-- Spring boot启动支持 -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Spring boot 集成dubbo -->
    <dependency>
      <groupId>com.alibaba.spring.boot</groupId>
      <artifactId>dubbo-spring-boot-starter</artifactId>
      <version>1.0.0</version>
    </dependency>

    <!-- zookeeper 客户端jar -->
    <dependency>
      <groupId>com.101tec</groupId>
      <artifactId>zkclient</artifactId>
      <version>0.10</version>
    </dependency>

    <dependency>
      <groupId>com.xyz.myhome</groupId>
      <artifactId>web-controller</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>web-starter</finalName>
    <plugins>

    </plugins>
  </build>
</project>
```

这边也主要就是看看消费者的dubbo配置文件，其实上文也讲过了，function的test其实也是一个即时性的消费者。

```
<?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" >
  <!-- 配置可参考 http://dubbo.io/User+Guide-zh.htm -->
  <!-- 服务提供方应用名，用于计算依赖关系 -->
  <dubbo:application name="myhome_web" owner="myhome_web" />
  <!-- 定义 zookeeper 注册中心地址及协议 -->
  <dubbo:registry protocol="zookeeper" address="127.0.0.1:2181"
    client="zkclient" />

  <dubbo:reference id="demoService"
interface="com.xyz.myhome.service.DemoService" />

  <dubbo:reference id="gdboxService"
interface="com.xyz.myhome.service.GdboxService"> </dubbo:reference>
</beans>
```