FSTP开发手册（v0.1.0）

# 目录

[目录 1](#_Toc518482446)

[修订（开发）记录 2](#_Toc518482447)

[一．环境搭建 3](#_Toc518482448)

[二．FSTP架构 3](#_Toc518482449)

[（一） Core层 4](#_Toc518482450)

[1）Entity 4](#_Toc518482451)

[2）IPC 4](#_Toc518482452)

[3）Cache 4](#_Toc518482453)

[4）Persist 5](#_Toc518482454)

[（二） Core-Service层 5](#_Toc518482455)

[1） System-core-service 5](#_Toc518482456)

[2） User-core-service 5](#_Toc518482457)

[3） Other-service 5](#_Toc518482458)

[（三） Business-Service层 5](#_Toc518482459)

[三．JAVA版本示例 6](#_Toc518482460)

[（一）Pub/Sub服务 6](#_Toc518482461)

[（二）Query 9](#_Toc518482462)

[（三）QNS 10](#_Toc518482463)

[（四） 持久化到数据库 12](#_Toc518482464)

# 修订（开发）记录

| 日期 | 版本 | 修订说明 | 修订者 |
| --- | --- | --- | --- |
| 2018-4-17 | v0.0.1 | 创建文档，加入pub和sub的示例程序。  目前程序可供使用的模块：   1. RPC：pub/sub，QNS，Query/Respond(目前不可用) 2. Cache：重新封装，弃用CacheClient，代替为FCache 3. DB：采用Spring-Data-jpa动态加载 4. BO：采用proto封装，可操作类后缀为\_OTW 5. MonitorService目前功能： 6. 侦测其他服务，以及心跳 7. 管理各个服务 8. 解析QNS请求   下一步计划做的工作：   1. 完成python、C/C++的接口开发工作，即开发python客户端时，任何service继承自PService，完成两者之间的兼容 2. Query/Respond模式的修复（因为BO序列化的原因，该功能需要重新设计） 3. 写python端的数据接口，对接tushare、Vnpy，以及量化平台 | Hans |
| 2018-7-4 | V0.1.0 | Java更新内容：   1. Entity：protobuf序列化，支持bo的继承，可以多层继承，但不可以多继承 2. IPC：支持pub/sub，QNS，Query/Respond(已可用) 3. Cache：加入ignite的cache模块，用户可选redis或者ignite作为cache，推荐使用ignite。针对ignite做了SQL语言处理 4. Persist：模块化持久化接口。 5. 升级version：v0.0.1 🡪 v0.1.0   Python更新内容：   1. PyPservice部分，参照java版本的PService开发，目前可用功能：log/config/Qpid/pub/sub/bo   暂时不建议在python版本上开发  下一步开发工作：   1. JAVA版本：完善core层，开发User-core-service，例如ignite计算服务等。 2. Python版本：完善核心功能，重点解决ignite问题 | Hans |

# 一．环境搭建

第三方依赖

|  |  |  |
| --- | --- | --- |
| Name | Version | Model |
| JRE | 1.8+ |  |
| Redis | 3.0.5+ | Cache |
| Apache Qpid | 7.0.0+ | IPC |
| Orcal MySQL | 5.7+ | 任意DB |
| Apache Ignite | 2.4.0+ | Cache |
| Goolge Proto Buffer | 3.5.1+ | Entity |

注:1.目前仅支持Java版本客户端，python版本开发中。两者区别：

1. java可以选用ignite/redis作为cache，python只能redis
2. monitor只存在java版本，python版本无monitor概念
3. java客户端可以与python版本相互消息转发、python可以通过特定的java客户端（设想）查询ignite服务

2.目前ignite仅应用在cache中，之后将开发连接spark等数据处理模块

# 二．FSTP架构

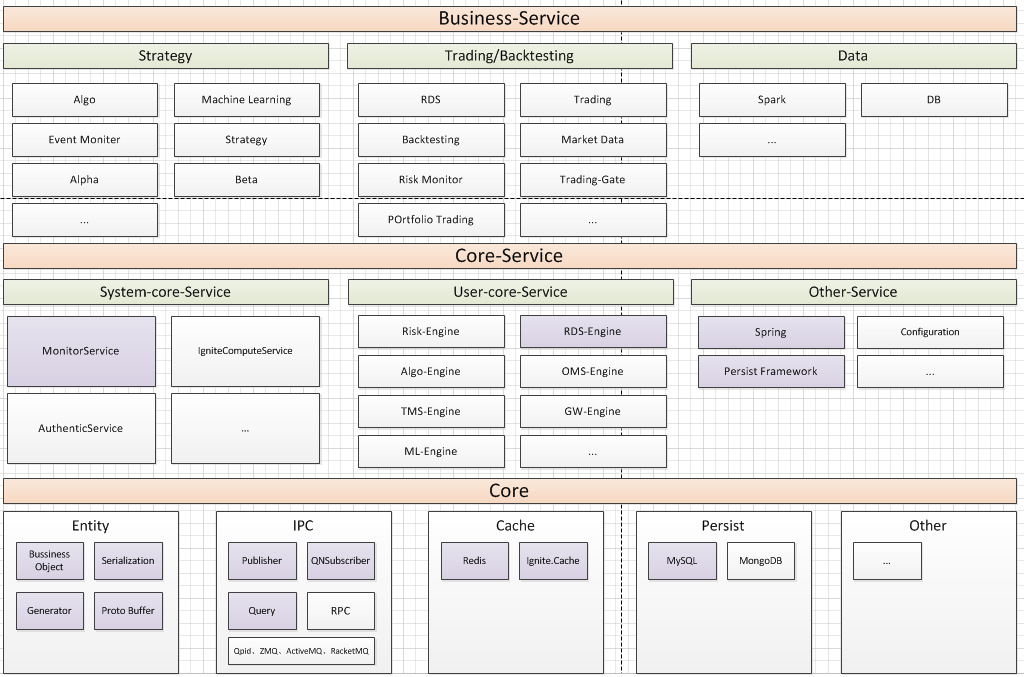


图1 FSTP总体架构（v 0.0.1）

## Core层

Core层作为FSTP的最底层，可以分为四个模块。

### 1）Entity

Entity模块是整个FSTP的数据层的传输实体。包括BO，OTW，PRO三种不同实体，每个bo都有对应的OTW/PRO。

BO是实体的定义类，generator会根据BO创建出对应的OTW,PRO类。同时BO类会用于ignite版本cache的存储，以及持久化。

OTW是用户操作的类，一般情况下，建议用户只操作OTW。

PRO是经过proto buffer生成的用于序列化的类，用户不可以直接对其操作。

该模块对应的jar是fstp-core-bo

### 2）IPC

IPC模块是FSTP中的通讯实体。包括中间件和具体的通信方式。

目前中间件采用的Apache的Qpid，python版本和Java版本是可以共用同一个Qpid Broker的，因为传输实体均为对应bo的proto byte数据，两者可以互相解析。对应的jar是fstp-core

通信方式目前提供三种：

1. Publish/subscribe

发送/订阅方式，最基本的通信方式，见示例。

1. Query/Respond

请求/应答模式，每个service均可以作为请求方或者应答方，具体参数可见示例。

1. QNS

请求并订阅方式。当一个Service发起QNS时，它将先做一次query，查询到历史数据，在做一次订阅，见示例。

### 3）Cache

Cache模块是FSTP中，沟通上层服务和底层不可见服务的关键，目前有两种Cache服务ignit和redis。每个服务只可以选择其一，可以配置。目前Ignite只支持Java版本的FSTP，如果系统中存在python客户端，建议使用redis。之后有计划开发一个core-service服务，用于同步redis和ignite的数据，以便python用户也可以使用ignite。对应的jar是fstp-core

### 4）Persist

Persist模块是存储模块。建议用户使用spring-data-jpa存储BO类。客户端可以使用任意DB产品。默认使用mysql。

## Core-Service层

Core-service层是系统服务层。JAR对应为fstp-core-service目前已经存在或计划加入的系统服务包括三大类：

### System-core-service

用于FSTP整个平台的管理，服务与所有service。

|  |  |  |
| --- | --- | --- |
| ServiceName | 是否必须 | 功能 |
| MonitorService | 至少存在一个 | 1. 与其他服务的心跳通信，检测其他服务的状态。 2. QNS解析服务 |
| AuthService | 当且只能一个 | 1）控制所有服务的认证注册（未开发） |
| IgniteComputeService | 否 | 用于支持上层ignite数据运算（未开发） |

### User-core-service

仅针对某些上层服务的特殊需求，这一模块不一定是一个FSTP Service，可以只是一个服务的接口，用于上层实体Service的方法声明。例如RDS-enginer，服务与上层的RDS模块。在RDS-enginer定义了一系列RDS的通用机制。上层的RDService只是下层的具体实现。这一块的JAR不一定在fstp-core-service中，可以根据需要定义在用户的jar中。

### Other-service

包括FstpConfiguration，Spring封装等。

## Business-Service层

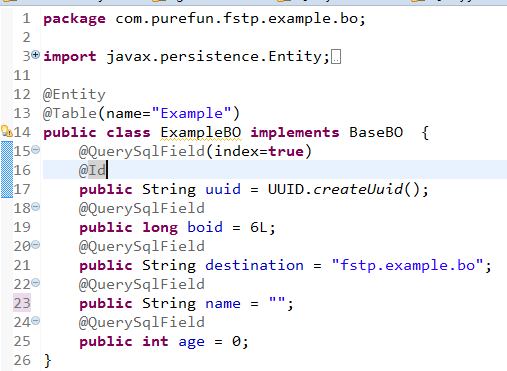
Business-Service层包括所有用户的service，计划按功能分为三个大部分，目前都没有实现。

|  |  |
| --- | --- |
| ModelName | 功能 |
| Strategy | 包括一系列用于辅助分析、事件爬虫、策略定义等一系列非交易服务 |
| Trading/backetesting | 交易和回测模块。包括交易、行情、算法交易、组合交易、策略交易等一系列服务 |
| Data | 大数据的存储和处理 |

# 三．JAVA版本示例

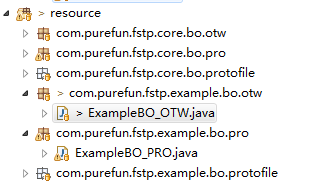
## （一）Pub/Sub服务

1. 创建bo

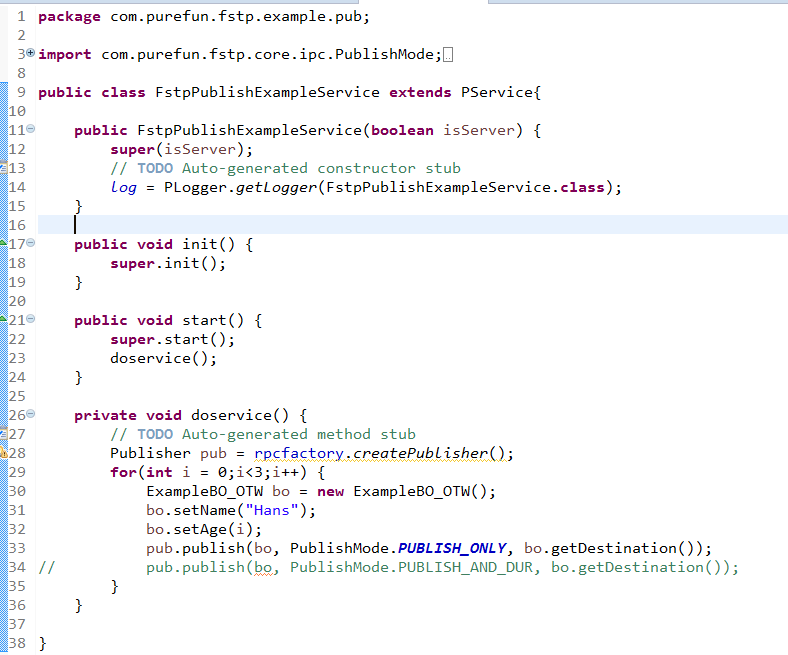


1. 生成对应OTW及PRO类，程序在fstp-core-bo的com.purefun.fstp.example.bo中。

GenerateJavaBOFile2.java。生成目录在resource中



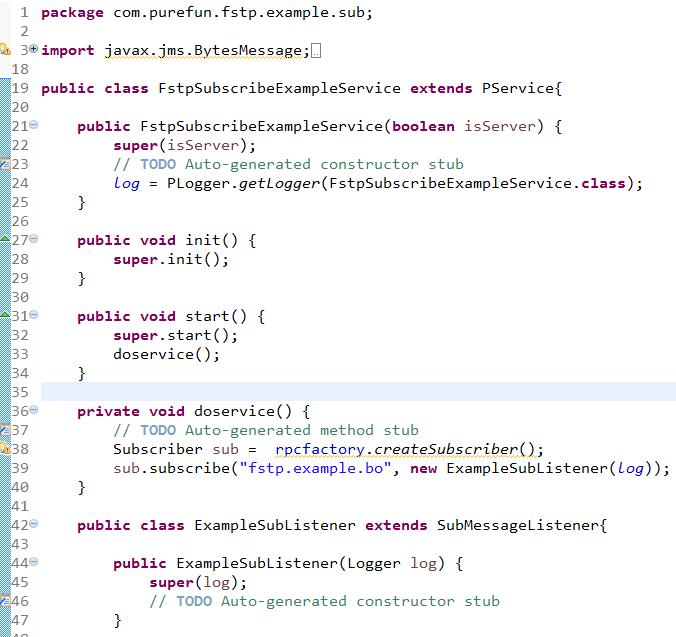
1. 将fstp-core-bo打包（以后会分开，用户的bo不会放在此处）package install -Dmaven.test.skip=true。检查是否已经生成了新的fstp-core-bo.jar
2. 创建FstpPublishExampleService服务。采用publish mode是只发送，没有保存

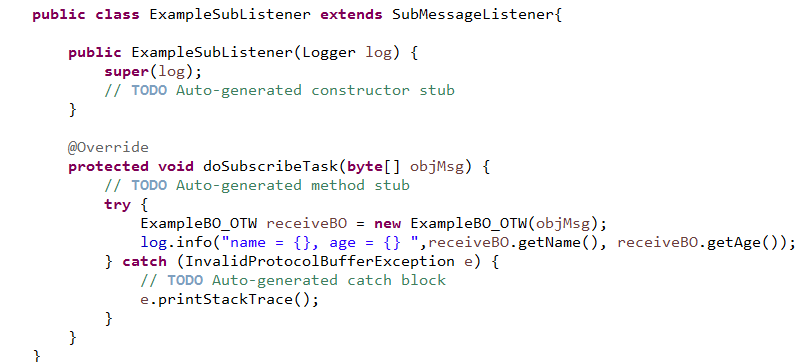


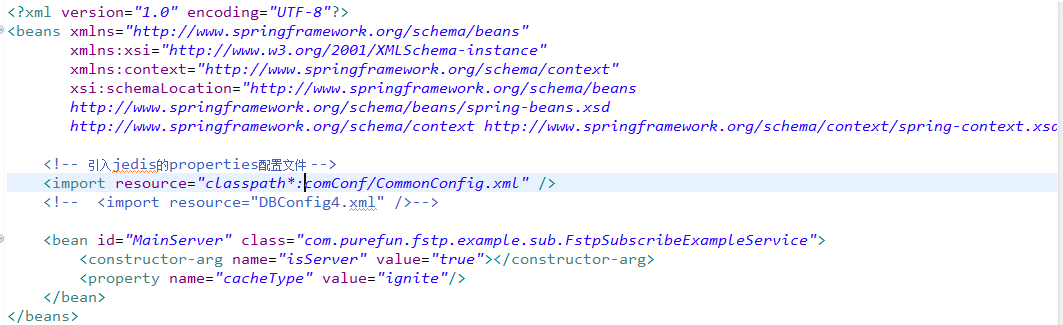
1. 配置config.xml



1. 创建FstpSubscribeExampleService服务，配置自己的config.xml

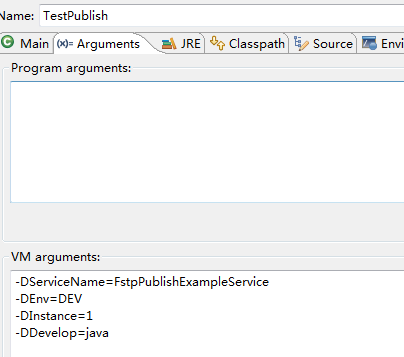
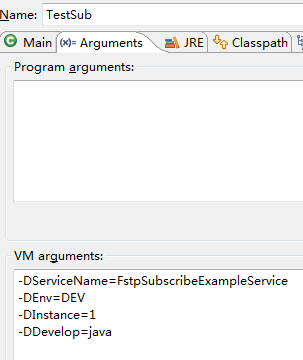






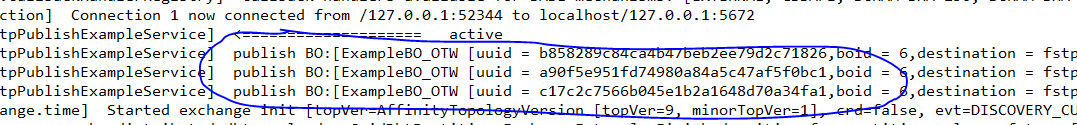
1. 设置两个service的启动参数，注意先启动sub，在启动pub，同时保证系统中有一个Monitor已经正常工作。

Pub： Sub：

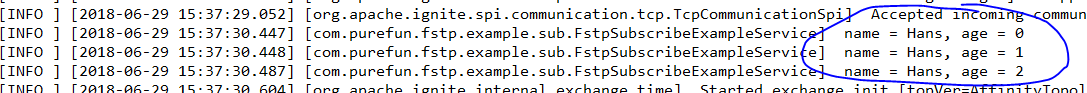
 

1. 运行结果：

Pub：

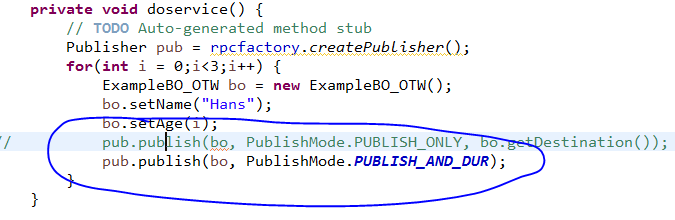


Sub：

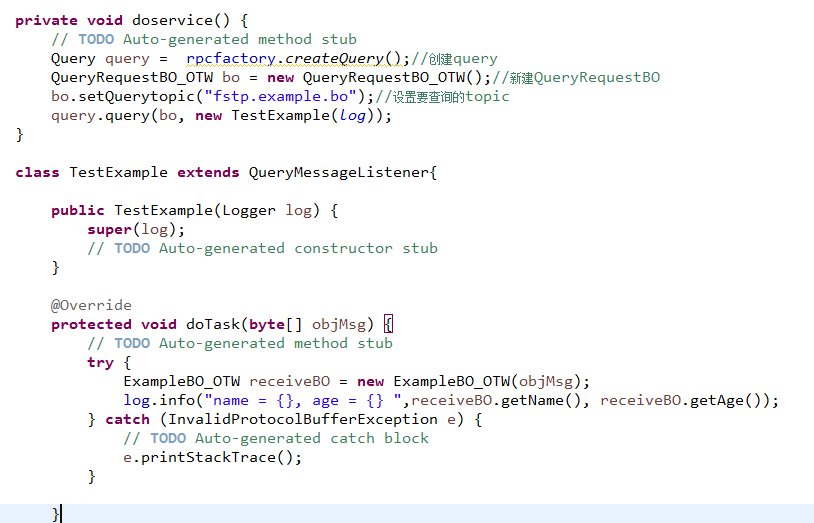


## （二）Query

1. 改造publishService，采用publish mode保存在cache中

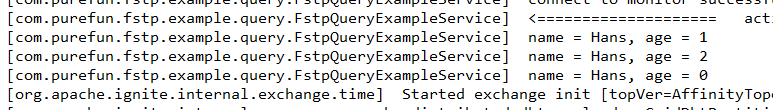


2）创建QueryService



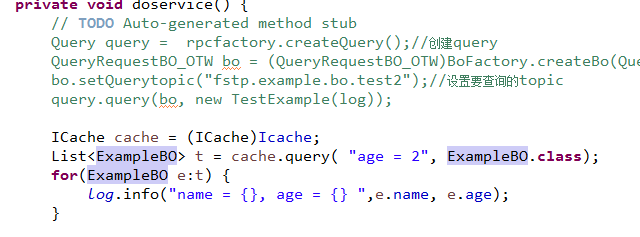


3）结果

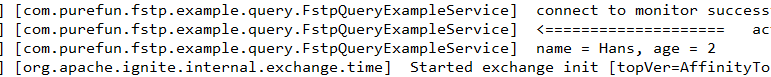


1. 查看自身服务的cache内数据。

查找的bo（此处为ExampleBO）应当是属于本服务管理的，即在config.xml有做配置，跨服务查询应使用上面的例子。推荐使用跨服务查询



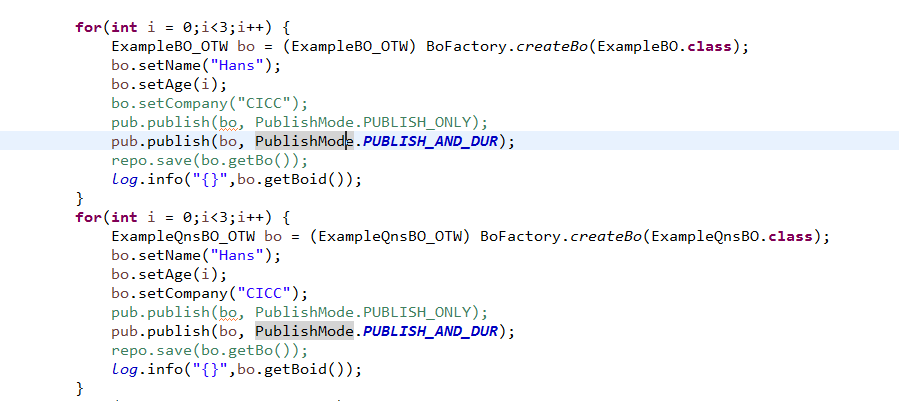
结果



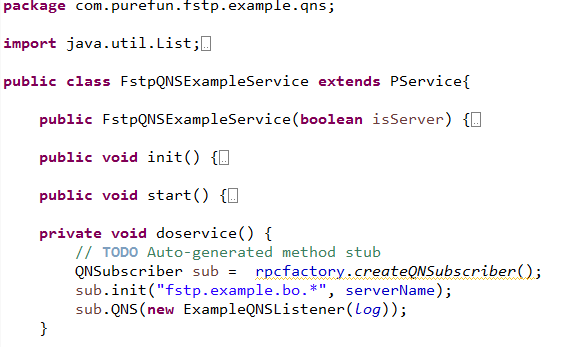
## （三）QNS

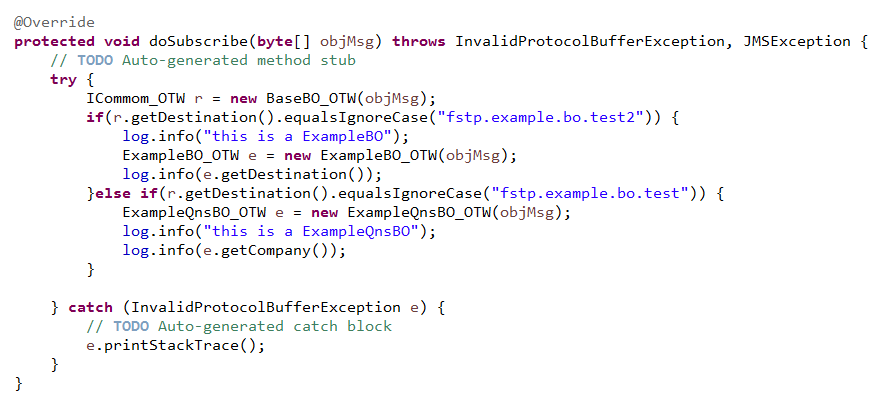
Qns服务，是指服务会首先做一次Query，将cache中的数据进行处理，之后再进行subscribe，处理后续分发出来的数据。QNS可以同时订阅多条topic，例如：fstp.example.bo.\*，经系统解析后，会同时订阅fstp.example.bo.test和fstp.example.bo.test2。用户在处理两个topic不同数据时，可以使用同一个listener，但用户需要自行判断来自于哪个topic。

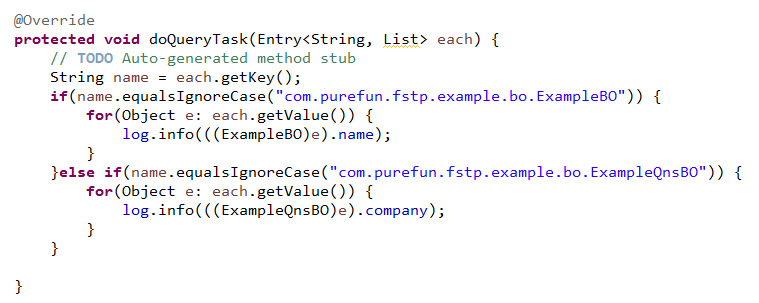
1）改造publishService，使它能发送两种数据ExampleBO和ExampleQnsBO，后者定义可以参照程序。



1. 创建QNS服务，他的listener继承QNSListener，该listener需要实现两个不同的方法doQueryTask用于处理从Query到的数据，doSubscribe用于处理sub后的数据。注意对不同topic收到的数据的处理，需要分别进行。

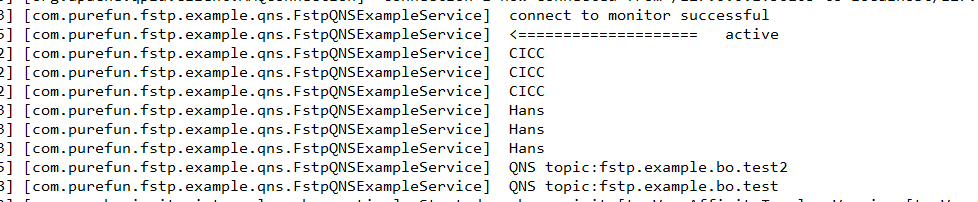




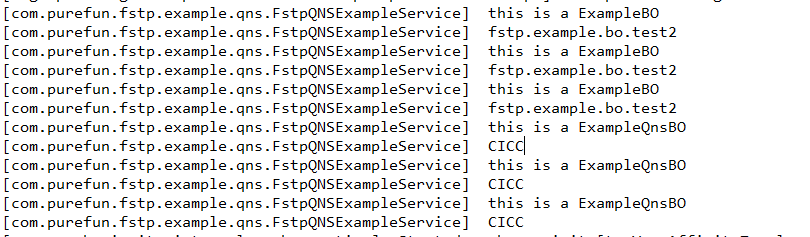


1. 验证过程，先起一个publish，再起QNS服务，验证query功能；再次起一个publish服务，验证sub功能

Query



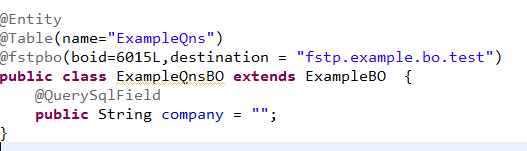
Subscribe

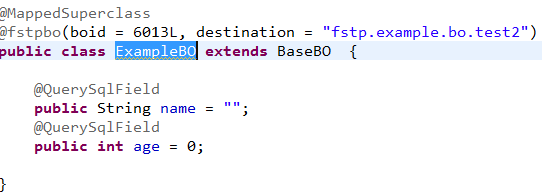


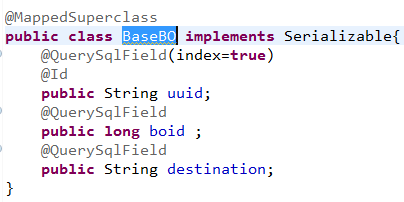
## 持久化到数据库

使用标准的spring-data-jpa解决方案，示例如下

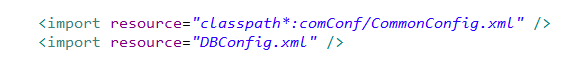
1. bo改造，将ExampleQnsBO持久化到数据库，存在继承关系，如下







1. config.xml，借用publish的config文件



1. DBConfig.xml





1. 改造publish服务



1. 在MySQL中创建对应表
2. 结果

