匡信科技技术框架开发规范（JAVA）

|  |  |
| --- | --- |
| 版本 | 1.0.0 |
| 发布时间 | 2014/9/16 |
| 作者 | 张欣 |

# 目录

[匡信科技技术框架开发规范（JAVA） 1](#_Toc398908363)

[目录 2](#_Toc398908364)

[综述 3](#_Toc398908365)

[设计目标 3](#_Toc398908366)

[总体架构 3](#_Toc398908367)

[业务框架（albacore） 3](#_Toc398908368)

[服务总线（albus） 3](#_Toc398908369)

[业务层次框架 5](#_Toc398908370)

[业务单元定义 5](#_Toc398908371)

[业务上下文定义 6](#_Toc398908372)

[上下文入口 6](#_Toc398908373)

[业务单元层次定义 7](#_Toc398908374)

[数据持久化框架 8](#_Toc398908375)

[关系数据持久化支持 8](#_Toc398908376)

[数据源 8](#_Toc398908377)

[事务控制 8](#_Toc398908378)

[数据模型映射 8](#_Toc398908379)

[数据操作映射 8](#_Toc398908380)

[NoSQL持久化支持 8](#_Toc398908381)

[缓存支持 8](#_Toc398908382)

[测试框架 9](#_Toc398908383)

[工具类支持 10](#_Toc398908384)

[实例化工具类 10](#_Toc398908385)

[静态工具类 10](#_Toc398908386)

[日志工具支持 11](#_Toc398908387)

[配置管理 12](#_Toc398908388)

[业务服务框架 13](#_Toc398908389)

[接口定义 13](#_Toc398908390)

[接口发布 13](#_Toc398908391)

[接口调用 13](#_Toc398908392)

# 综述

## 设计目标

技术开发框架作为匡信科技开发团队日常业务和技术开发的基础架构，主要目标是

* 规范日常开发行为，为团队协同开发在代码表现、业务实现、部署运维等多个维度上定义统一、稳定的标准化结构和风格。
* 提高代码和实现质量，增强开发产出的可维护性，包括：
  + 代码可读性
  + 业务扩展和变更灵活性。
* 降低开发团队成员进入门槛，减少培训和学习成本，便于开发团队规模和组成适应业务和技术要求发展实现敏捷化调整。
* 外部技术引入和封装：
  + 降低外部技术引入为团队开发带来的难度和风险。
  + 统一同一类型技术依赖方式和接口，屏蔽不同技术选型要求的业务实现差异。
  + 增强外部技术依赖选型变更灵活性，保证选型变更不影响业务实现。

## 总体架构

完整框架当前包括两个部分：业务框架和服务总线

### 业务框架（albacore）

主要负责业务层次定义和规范、数据持久化建模和访问、测试自动化、常用工具类等。包括以下组件：

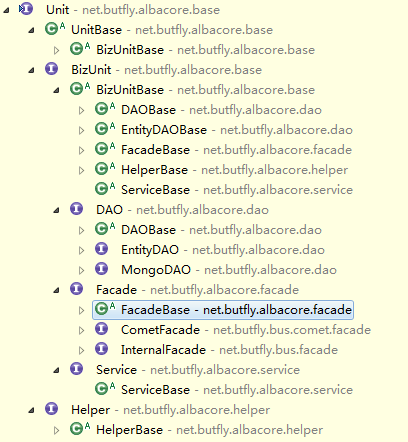
* albacore-core：框架核心组件，被所有其他框架依赖，定义了albacore底层实现和业务单元基本抽象。
* albacore-frame：负责业务单元层次化架构。
* albacore-orm：负责数据模型定义和数据持久化基本操作，同时提供数据持久化访问的缓存支持扩展接口。
* albacore-cache：负责缓存相关工具抽象、接口定义。
* albacore-test：负责自动化测试用例开发、运行。
* albacore-utils：定义了日常开发中使用的工具类，负责外部依赖在albacore框架中的集成和抽象。

### 服务总线（albus）

主要负责业务前端（展现和交互组件）和后端（业务逻辑实现组件）之间的数据通讯、协议抽象和适配、接口定义、接口发布、接口发现、访问监控、权限管理等功能。主要包括以下组件：

* albus-core：服务总线核心实现，定义了基本服务总线功能，包括：
  + 接口定义
  + 接口发布
  + 配置管理
  + 过滤机制
  + 路由机制
  + 基于BeanFactory的默认业务单元管理机制
  + 基于进程内访问的默认接口调用机制
  + 异步访问机制
* albus-plus：服务总线扩展实现，包括：
  + 基于Spring的业务单元管理机制
  + 基于Servlet的远程接口调用机制及相应容器，默认提供：
    - 基于嵌入化Jetty的接口部署容器。
    - 基于JSON协议的接口调用机制。
* albus-hessian：服务总线的hessian协议扩展实现。

# 业务层次框架

在albacore框架中，业务逻辑实现以**业务单元**形式实现和管理。业务单元在一个完整的业务逻辑运行上下文中一般作为单例存在，并由业务逻辑上下文提供业务单元之间的依赖、访问以及创建、维护、销毁等生命周期管理。

符合albacore规范的业务逻辑实现，应由Spring负责管理业务单元层次、依赖和生命周期。

## 业务单元层次

一个完整的业务单元声明继承树大致如右图：

业务单元被由顶向下的定义为4个层次：

1. 门面层（Facade）：负责业务逻辑入口开放。

每一个Facade单元包括业务子系统内一组相关的业务逻辑实现，负责管理其对外（对其他业务子系统）的暴露和被引用。

每一个Facade单元实现由其暴露的业务逻辑接口对实际业务逻辑实现的直接调用组成，不应在Facade实现中包含真实业务逻辑，亦即，不应在Facade实现中包含多次实际业务逻辑实现调用或业务流程（条件、判断、分支、循环等）。

Facade单元被定义为服务总线框架（albus）的服务接口发布工具，亦即，一切由服务总线框架定义的服务接口均应作为在某个Facade中定义的接口暴露。

声明和命名规范：

* 每个Facade单元由一个接口声明和（多个）接口实现组成。
* 接口声明应定义为一个继承了Facade通用接口的声明接口。
* 接口声明命名以Facade为后缀，如<模块名称>Facade。
* 接口实现应包含其中实现的业务接口的接口声明接口并继承FacadeBase通用基类。
* 接口实现命名以接口声明前缀为前缀，接口实现类型为中缀（可选），以FacadeImpl为后缀，如<模块名称><实现方式>FacadeImpl。
* 接口声明、接口实现、接口输入和输出参数均应被声明在业务子系统名字空间对应的包（package）根的直接子包facade下（可以进一步定义子包以划分模块或类型），如：<组织>.<产品>.<组件>.<子系统>.facade。

1. 服务层（Service）

每一个Service单元包括业务子系统内一组相关的业务逻辑具体实现，负责具体实现其业务的真实逻辑和流程。

声明和命名规范：

* 每个Service单元由一个接口声明和（多个）接口实现组成。
* 接口声明应定义为一个继承了Service通用接口的声明接口。
* 接口声明命名以Service为后缀，如<模块名称>Service。
* 接口实现应包含其中实现的业务接口的接口声明接口并继承ServiceBase通用基类。
* 接口实现命名以接口声明前缀为前缀，接口实现类型为中缀（可选），ServiceImpl为后缀，如<模块名称><实现方式>ServiceImpl。
* 接口声明、接口输入和输出参数均应被声明在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的service子包下（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.service。
* 接口实现应被声明在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的service子包下impl子包中（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.impl.service。

1. 数据操作层（DAO）

每一个DAO单元定义了一组持久化数据操作相关元实现，通过调用DAO基类实现中定义的基本数据操作实现复杂的数据操作功能。

声明和命名规范：

* 每个DAO单元由一个接口声明和（多个）接口实现组成。
* 接口声明应定义为一个继承了DAO通用接口的声明接口。
* 接口声明命名以DAO为后缀，如<模块名称>DAO。
* 接口实现应包含其中实现的业务接口的接口声明接口并继承DAOBase通用基类。
* 接口实现命名以接口声明前缀为前缀，接口实现类型为中缀（可选），以DAOImpl为后缀，如<模块名称><实现方式>DAOImpl。
* 接口声明、接口实现、接口输入和输出参数均应被声明在业务子系统名字空间对应的包（package）根的直接子包façade下（可以进一步定义子包以划分模块）。
* 接口声明、接口输入和输出参数均应被声明在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的dao子包下（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.dao。
* 接口实现应被声明在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的dao子包下impl子包中（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.impl.dao。

每个DAO接口中定义的方法会被albacore框架声明为一个数据库事务。事务类型由方法名前缀决定：

* select/load前缀方法会被定义为只读事务。
* update/delete/insert/toggle前缀方法会被定义为读写事务。

1. 工具层（Helper）

每一个Helper单元定义了与持久化数据无关（无需事务支持）的相关业务逻辑元实现，一般Helper业务单元用于对外部功能的依赖和引入，使用Helper接口声明抽象外部功能为业务逻辑提供的功能依赖，使用多个Helper实现封装不同外部功能实现和接口差异，以提供灵活的外部功能实现切换。

声明和命名规范：

* 每个Helper单元由一个接口声明和（多个）接口实现组成。
* 接口声明应定义为一个继承了Helper通用接口的声明接口。
* 接口声明命名以Helper为后缀，如<模块名称> Helper。
* 接口实现应包含其中实现的业务接口的接口声明接口并继承HelperBase通用基类。
* 接口实现命名以接口声明前缀为前缀，接口实现类型为中缀（可选），以HelperImpl为后缀，如<模块名称><实现方式> HelperImpl。
* 接口声明、接口输入和输出参数均应被声明在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的helper子包下（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.helper。
* 接口实现应被声明在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的helper子包下impl子包中（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.impl.helper。

## 上下文入口

albacore框架由Spring上下文管理完整的业务上下文，包括业务单元层次、依赖、配置和生命周期。

在Spring上下文中，统一使用beans.xml作为上下文入口配置。业务容器（例如：服务总线框架）以此为入口载入和使用业务上下文。

beans.xml应被定义在业务子系统名字空间对应的包（package）根目录，如：<组织>.<产品>.<组件>.<子系统>。

beans.xml中应该包含多个具体业务单元配置文件的引用（import）。

各具体业务单元配置文件应该对应业务子系统中的多个业务模块，以beans-为前缀，包含其对应的业务模块实现所需所有业务单元配置，例如：beans-<业务模块名称>.xml。

albacore框架提供大量默认业务单元配置，例如，负责关系数据库事务控制的配置文件beans-tx.xml（位于albacore-orm子项目中）。

albacore框架中每个子项目均有一个子项目入口配置，以beans-<子项目名称>.xml命名，可供框架用户直接引用以获得子项目提供的通用功能。也可以独立引用各子项目模块配置文件或自行定义各子项目实现的业务单元。

## 上下文定义

所有业务单元在Spring中作为bean定义时，均应继承对应的基类bean定义以获得统一定义的规范化功能：

* Facade单元bean应继承facade bean，定义如下：

<bean abstract=*"true"* id=*"facade"* ... />

* Service单元bean应继承service bean，定义如下：

<bean abstract=*"true"* id=*"service"* ... />

* DAO单元bean应继承dao bean，定义如下：

<bean abstract=*"true"* id=*"dao"* ... />

* Helper单元bean应继承helper bean，定义如下：

<bean abstract=*"true"* id=*"helper"* ... />

以上基类bean除dao由albacore-orm组件定义外，其他均由albacore-frame组件定义。

# 数据持久化框架

## 关系数据持久化支持

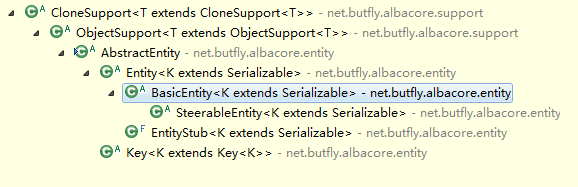
数据在关系数据库中的设计、建模、持久化、事务管理等功能，由albacore-orm项目提供。

albacore框架使用jndi机制提供数据源访问，使用mybatis提供数据映射和操作映射，使用dao业务单元层次提供数据操作实现，使用spring提供的基于声明式事务控制机制提供事务控制。

关系数据持久化支持主要包括以下几个方面：

### 数据模型映射

经业务数据建模设计后产生的业务数据模型，除被映射为关系数据库表结构以外，还应该被映射为以Java POJO对象为形式实现的数据实体对象（Entity）。

每个业务逻辑需要访问的关系数据库表结构（包括视图、存储过程返回结果集等其他关系化数据结构），均应定义相应的数据实体对象。

数据实体对象应继承AbstractEntity对象（或根据业务需求继承其某个子类）。当前实体类继承树如右图：

数据实体对象应根据关系数据库表或视图名命名，或根据数据业务命名，不带后缀。

数据实体对象应被定义在业务子系统名字空间对应的包（package）根下，对应业务模块所在子包中的mapper子包中（可以进一步定义子包以划分模块或类型），例如：<组织>.<产品>.<组件>.<子系统>.<模块>.mapper。

### 数据操作单元

数据操作单元定义了数据操作的具体业务逻辑和事务控制，详见“业务层次架构”中“数据操作层（DAO）”部分定义。

### 数据操作映射

albacore内置mybatis实现数据操作业务逻辑与数据操纵SQL语句的映射，在mybatis映射文件（Mapper XML）中以SQL ID标识多条SQL语句。所有映射文件均被mybatis入口配置文件（mybatis-config.xml）包含（import）。每个业务子系统的mybatis入口配置文件应被定义在业务子系统名字空间对应的包（package）根下，例如：<组织>/<产品>/<组件>/mybatis-config.xml。

#### 名字空间定义

映射文件中定义的多条SQL语句通过名字空间（namespace）组织，每个映射文件属于一个名字空间，多个映射文件可以属于同一个名字空间。

原则上，数据操作单元（DAO）类定义与mybatis映射文件名字空间（namespace）定义一一对应，每个映射文件名字空间中定义一组相关业务逻辑需要的数据持久操作。名字空间名与数据操作单元全名名一致，仅需去除DAOImpl后缀，例如：<组织>/<产品>/<组件>/<子系统>/<模块>/<数据操作单元名>.xml。

albacore在指定映射文件中定义了一些公用SQL语句（段），可供通用功能或特定业务功能应用，如分页语句等。该映射文件被定义在albacore-orm项目中net/butfly/albacore/orm/dbo/common-sql-<数据库实现>.xml文件中。该文件应被所有业务子系统实现的mybatis入口文件包含。

一个名字空间中，主要包括以下两类映射定义

#### 数据模型映射定义

数据模型映射定义了实体类对象到mybatis结果集的映射，mybatis结果集可以用于mybatis数据持久操作的输入参数或输出返回类型。

#### 数据操作映射定义

数据操作映射定义了实际完成数据持久操作的SQL语句。按照操作类型以指定动宾规范命名SQL语句的标识（SQL ID），包括以下几类：

* insert：以insert为前缀谓词，以被创建的实体（entity）类名为宾语，可以带具体操作类型后缀。
  + insert<Entity>List：批量插入语句
* delete：以delete为前缀谓词，以被删除的实体类名为宾语，可以带具体操作类型后缀。
* update：以update为前缀谓词，以被修改的实体类名为宾语，可以带具体操作类型后缀。
* select：以select为前缀谓词，以被查询（返回）的实体类名为宾语，可以带具体操作类型后缀。
  + select<Entity>ByCriteria：条件查询语句。
* count：以count为前缀谓词，以被计数的实体类名为宾语，可以带具体操作类型后缀。
  + count<Entity>ByCriteria：条件计数语句。

### 数据源配置

albacore通过数据源配置访问关系数据库。数据源通过jndi模式在容器中定义。由spring负责jndi数据源读取和引入。在albacore中定义了默认数据源配置文件net/butfly/bus/console/spring/beans-ds.xml，其中

自动读取jndi.ds配置项作为当前业务子系统容器中定义的数据源jndi名称。若业务子系统使用单一数据源，可以直接引入该beans文件以直接规范化使用数据源。

若业务子系统使用多数据源动态切换，需要自定义数据源配置文件并引入。

### 事务控制

albacore默认事务控制策略由配置文件net/butfly/albacore/orm/spring/beans-tx.xml定义，通过spring提供的基于切片的声明式事务控制机制提供事务控制。该文件中定义了以下事务策略：

* 事务控制对所有以DAOImpl为后缀的、由spring上下文管理的业务单元实例，带指定命名前缀的方法生效。
* 以select前缀命名的方法，被声明为只读事务，默认超时10秒。
* 以batch/insert/delete/update/toggle前缀命名的方法，被声明为读写事务。
  + 以batch前缀命名的读写事务方法，默认超时30秒（批处理事务）。
  + 以其他前缀命名的读写事务方法，默认超时5秒（单独写事务）。

## NoSQL持久化支持

待实现

## 缓存支持

待实现

# 测试框架

待编写

# 工具类支持

## 实例化工具类

实例化工具类以业务层次框架中的工具层（Helper）定义，由spring上下文实例化、维护、配置和管理，主要用于以下场景：

* 可能产生变更的工具实现或外部依赖引入。
* 需要配置调整的工具实现或外部依赖引入。

## 静态工具类

静态工具类以UtilsBase虚类的基类形式声明，不能被实例化，所有工具方法均应被声明为静态方法，主要用于变更稳定的简单工具方法实现。

# 日志工具支持

albacore框架通过slf4j支持基于log4j的高级日志功能。提供两级配置管理：

* albacore框架内置配置，位于albacore-core子项目中，提供简单的默认配置如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN"  "http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-files/log4j.dtd">  <log4j:configuration xmlns:log4j=*"http://jakarta.apache.org/log4j/"*  debug=*"false"*>  <appender name=*"\_albacore\_default"* class=*"org.apache.log4j.ConsoleAppender"*>  <layout class=*"net.butfly.albacore.logger.ANSIColorPatternLayout"*>  <param name=*"ConversionPattern"* value=*"%d{ABSOLUTE}[%5p][%c{1}][%t]: %m%n"* />  </layout>  </appender>  <root>  <level value=*"warn"* />  <appender-ref ref=*"\_albacore\_default"* />  </root>  </log4j:configuration> |

* 业务子系统日志配置，位于业务子系统项目根classpath中，由业务子系统自定义，可选，如果存在，覆盖albacore框架内置配置。

# 配置管理

## 配置机制

albacore框架配置主要由properties文件组成，由spring提供的PropertyPlaceholderConfigurer负责载入。一般由二级机制组成：

* 框架内置配置，位于albacore各子项目中指定子包内。
* 业务子系统自定义配置，位于业务子系统中，定义在根classpath中。

二级配置文件若都存在，均会被载入。若存在相同的配置选项，业务子系统自定义配置会覆盖框架内置配置。

## 已定义配置

当前定义了以下配置文件（按优先级从高到低排序）：

1. classpath:/albacore.properties（自定义配置）
2. 数据库配置（自定义配置）
   * + classpath:/mybatis.properties
     + classpath:/jdbc.properties
3. classpath:/net/butfly/albacore/utils/context/albacore-utils.properties（内置配置）
4. 数据库配置（内置配置）
   * + classpath:/net/butfly/albacore/orm/context/albacore-orm.properties
     + classpath:/net/butfly/albacore/orm/context/albacore-mybatis.properties
     + classpath:/net/butfly/albacore/orm/context/albacore-jdbc.properties
5. classpath:/net/butfly/albacore/frame/context/albacore-frame.properties（内置配置）
6. classpath:/net/butfly/albacore/core/context/albacore.properties（内置配置）

## 重要配置项

* albacore.project.name

业务子系统名（<子系统>）。

* albacore.project.package.root

业务子系统根包（一般为：<组织>.<产品>.<组件>.<子系统>）。

* albacore.datasource.jndi.name

单数据源子系统数据源JNDI资源名称

* ibatis.cacheEnabled=false

mybatis缓存支持

* ibatis.lazyLoadingEnabled=false
* mybatis延迟载入
* ibatis.aggressiveLazyLoading=false
* mybatis侵略性延迟载入
* ibatis.defaultExecutorType=BATCH
* mybatis默认执行方式
* ibatis.defaultStatementTimeout=120
* mybatis默认数据操作超时

# 服务总线框架

albus是一个专注远程服务接口调用过程的服务总线框架，在业务逻辑中位于前端（展现和交互）与后端（业务逻辑实现）之间，负责为远程业务接口调用提供协议封装、序列化/反序列化、业务单元管理、接口发布、监控等功能，从而为业务服务接口的实现者（业务后端）和调用者（业务前端）提供一个完全透明、可配置、可扩展的远程调用框架。

## 接口定义

在albus中，接口定义通过Java标注（Annotation）TX实现。其定义如下：

|  |
| --- |
| **public** **@interface** TX |

TX标注可以携带以下参数：

* value：默认参数，接口代码。
* version：可选参数，接口实现版本号，默认值为初始版本（1.0.0）。

TX标注应被声明在Façade接口的子接口中声明的方法签名上，以定义该方法签名成为一个可以被前端调用的业务服务接口。示例如下：

|  |
| --- |
| **public** **interface** CometFacade **extends** Façade {  @TX("TST\_CMT-000")  **public** CometEchoReponse echo(String echo);  } |

所有被定义为业务服务接口的方法，可以携带任意类型参数，并返回单个任意类型返回值，或无返回值。参数与返回值类型建议规范如下：

* 简单类型
  + 字符串类型
  + 标量类型（byte/short/int/long/char...）
  + 标量类型的封包类型（Byte/Short/Integer/Long/Char...）
* 以符合规范的任意类型为元素类型的数组类型（[]）
* 以字符串为键（key）类型、以符合规范的任意类型为值（value）类型的Map类型（Map<String, ?>）
* 以符合规范的任意类型字段的简单对象类型（POJO）

当定义POJO类型时，应继承net.butfly.bus.dto.Request或net.butfly.bus.dto.Response类型（根据类型供输入参数或返回结果）。

## 接口实现

albus中定义的业务服务接口由继承其声明接口的业务单元类实现。

所有实现了业务服务接口的业务单元实例由albus容器管理。所有容器实现均应继承AbstractInvoker父类。

albus提供了以下业务单元容器：

* BeanFactoryInvoker：业务单元实例生命周期由业务子系统自行管理，业务子系统创建的业务单元实例应被注册到BeanFactoryInvoker中。
* SpringInvoker：业务单元实例生命周期由Spring上下文管理，业务子系统提供spring入口配置文件（例如：beans.xml）。
* HessianInvoker：转发（路由）当前业务服务到另一个服务总线节点（通过Hessian协议）。

## 接口发布

接口发布由服务总线服务端配置完成，当前通过bus.xml提供本地配置方式：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <bus id=*"bus-console"*>  <filter title=*"async-filter"* class=*"net.butfly.bus.filter.AsyncFilter"*  enabled=*"false"* />  <filter title=*"logger-filter"* class=*"net.butfly.bus.filter.LoggerFilter"*  enabled=*"true"* />  <filter title=*"exception-filter"* class=*"net.butfly.bus.filter.ExceptionHandleFilter"*  enabled=*"true"* />  <router type=*"net.butfly.bus.policy.SimpleRouter"* />  <invoker id=*"albus-test-comet"* tx=*"TST\_CMT-\*"*  type=*"net.butfly.bus.invoker.BeanFactoryInvoker"* />  </bus> |

## 接口调用

接口调用配置由服务总线客户端配置完成，当前通过bus.xml提供本地配置方式：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <bus id=*""*>  <filter title=*"async-filter"* class=*"net.butfly.bus.filter.AsyncFilter"*  enabled=*"false"* />  <filter title=*"logger-handler"* class=*"net.butfly.bus.filter.LoggerFilter"*  enabled=*"true"* />  <filter title=*"exception-filter"* class=*"net.butfly.bus.filter.ExceptionHandleFilter"*  enabled=*"true"* />  <invoker type=*"net.butfly.bus.invoker.HessianInvoker"* id=*"albus-test-comet"*  tx=*"TST\_CMT-\*"*>  <path>http://127.0.0.1:9876/bus</path>  <typeTranslator>net.butfly.bus.hessian.serialize.EnumFactory  </typeTranslator>  <continuousSupported>true</continuousSupported>  </invoker>  </bus> |