Restful服务框架(Athena)的架构设计

作者：李艳鹏

日期: 2015.05.08

[Restful服务框架(Athena)的架构设计 1](#_Toc419487563)

[当前REST服务存在的问题 2](#_Toc419487564)

[需求整理 2](#_Toc419487565)

[概要设计 3](#_Toc419487566)

[需求分层 3](#_Toc419487567)

[分层目标输出 4](#_Toc419487568)

[详细设计 5](#_Toc419487569)

[集成层架构设计 5](#_Toc419487570)

[容器层架构设计 8](#_Toc419487571)

[Webapp层架构设计 9](#_Toc419487572)

[REST层，服务层和数据层 12](#_Toc419487573)

[性能需求 12](#_Toc419487574)

# 当前REST服务存在的问题

1. 各业务的服务提供方式有所不同，有的使用tomcat有的使用jetty，上线脚本和上线流程也有不同。
2. 属性管理耦合在项目的源代码中，有的项目会在打包脚本里有冗余的属性配置，打包时合并，这导致可能开发环境可以存取线上的资源。
3. 有的业务模块绑定到tomcat或者jetty，不能很容易的在tomcat和jetty之间切换。
4. 各个模块有自己的异常代码处理器，各个业务之间导致使用的http code不一致。也就是没有一个统一的REST规范。
5. 有些项目使用Spring Web MVC，有些项目使用其他Web MVC框架，例如：Apache CXF，没有一个统一的Web MVC框架或者REST框架，给开发框架级功能造成了麻烦，例如：监控，降级，日志等。
6. 内嵌tomcat和jetty启动和Web MVC框架提供的功能耦合在业务项目中，层次不清晰，导致业务的开发人员不但要负责业务开发还有关心框架功能的开发。

# 需求整理

|  |  |  |
| --- | --- | --- |
| **功能目标** | **功能** | **非功能质量** |
| 发布人员 | 1. 能够较容易的与CI打包集成，需要和CI的一键打包无缝集成。 2. 能够较容易的与DCMD发布集成。 3. 管理各个项目连调，集成和线上属性，并且打包时能够容易的选择各个环境的属性进行替换。 4. 提供一个统一的standalone启动脚本，用于启动，停止和检查restful服务。 5. 进行统一的GC优化，GC日志的设置，线程池的设置，并且上线打包可以针对特殊业务进行覆盖。 | 可维护性 易用性 安全性 |
| 框架级需求 | 1. 能够切换内嵌的http容器，例如tomcat和jetty, 最好封装一个插拔的框架，便于切换到其他容器，例如netty 2. 可重用容器配置server.xml或者相对应配置,以及tomcat-users容器的专用配置文件 3. 可重用web-content的内容，包括web.xml, index.jsp, 503.jsp,200.jsp等,并且提供机制让业务项目可覆盖web.xml。 4. 提供属性替换功能，打包时可替换为连调，集成和线上环境。 5. 提供框架级别的可插拔的功能，自动降级，请求日志的归类和阶段化，性能日志的推送 6. 支持http和https协议的转换 7. 与注册中心，监控中心集成 8. 指定返回对象的具体字段，指定分页范围，指定返回格式，指定版本号 9. 异常代码翻译，返回HTTP标准错误代码，异常处理需要实现可插拔的AOP处理器 10. 管理默认的dev属性 11. 提供服务启动后加载完成的钩子，配合发布后校验 | 高性能 可扩展性 可伸缩 安全性 可插拔性 易用性 可用性 |
| 开发者 | 1. 管理默认的dev属性 2. 方便的单元测试 3. 能够简单的在dev环境运行服务进行简单的集成测试 4. 参数校验和参数转换 5. 针对业务进行服务的组合和拆分，并且提供REST的导出 6. 提供原子服务层，可提供给REST层进行JVM调用也可提供给RPC进行服务导出 7. 支持分库分表，支持灵活的分库分表策略的插拔，hash，按时间顺序或者两个的结合，实现方式可选择路由，业务层实现，ORM层实现或者JDBC层实现，SQL路由和结果合并 | 正确性 可扩展 易用性 |

# 概要设计

## 需求分层

|  |  |  |  |
| --- | --- | --- | --- |
| **层次** | **功能目标** | **功能** | **非功能质量** |
| 集成层 | 发布人员 | 1. 能够较容易的与CI打包集成 2. 能够较容易的与DCMD发布集成 3. 管理下层连调，集成和线上属性，并且打包时使用maven-filter替换,实现属性替换功能，打包时可替换为连调，集成和线上环境 4. 提供一个统一的standalone启动脚本，用于启动，停止和检查restful服务 5. 进行统一的GC优化，GC日志的设置，线程池的设置，并且上线打包可以针对特殊业务进行覆盖。 | 可维护性 易用性 安全性 |
| 容器层 | 业务开发者 | 1. 能够切换内嵌的http容器，例如tomcat和jetty, 最好封装一个插拔的框架，便于切换到其他容器，例如netty 2. 可重用容器配置server.xml或者相对应配置,以及tomcat-users容器的专用配置文件 3. 支持http和https协议的转换 | 可插拔性 易用性 可用性 |
| Webapp层 | 业务开发者 | 1. 可重用web-content的内容，包括web.xml, index.jsp, 503.jsp,200.jsp等,并且提供机制让业务项目可覆盖web.xml。 2. 提供框架级别的可插拔的功能，自动降级，请求日志的归类和阶段化，性能日志的推送 3. 与注册中心，监控中心集成 4. 指定返回对象的具体字段，指定分页范围，指定返回格式，指定版本号 5. 异常代码翻译，返回HTTP标准错误代码，异常处理需要实现可插拔的AOP处理器 6. 提供服务启动后加载完成的事件，配合发布后校验 | 高性能 可扩展性 可伸缩 安全性 可插拔性 易用性 可用性 |
| REST层 | 业务开发者 | 1. 参数校验和参数转换 2. 针对业务进行服务的组合和拆分 3. 管理默认的dev属性 | 正确性 可扩展 易用性 |
| 服务层 | 业务开发者 | 1. 提供原子服务的实现 2. 可提供给REST层进行JVM调用也可提供给RPC进行服务导出 3. 管理默认的dev属性 | 正确性 高性能 |
| 数据层 | 业务开发者 | 1. 支持分库分表 2. 支持灵活的分库分表策略的插拔 3. Sql路由和结果集合并？ 4. 业务层实现？代理实现？OR-Mapping层实现？还是JDBC层实现？ 5. 管理默认的dev属性 | 正确性 高性能 可伸缩 |

## 分层目标输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **层次** | **负责人** | **调试和测试** | **构建工具和输出** | **框架** |
| 集成层 | 框架开发者 CI开发者 |  | 1.输出包为tar.gz格式，内容包括，DCMD包模板，各个项目启动属性，打包脚本，并且发布到maven私服  2. CI从Maven私服下载tar.gz包，解压后调用打包脚本直接输出相应项目和相应环境的DCMD包 | Shell脚本 Maven |
| 容器层 | 框架开发者 |  | 1. 使用Maven打自启动Jar包，上传到Maven私服 2. 默认加载类路径下的web.xml 3.jar包可自启动，不需要shell脚本 | 嵌入式Tomcat Jetty |
| Webapp层 | 框架开发者 |  | 1. 使用Maven打自启动Jar包，上传到Maven私服 2. 提供通用web.xml，默认加载下层类路径下的athena-rest-container.xml,在此Spring环境中加载下层的athena-rest-context.xml 3.jar包可自启动，不需要shell脚本 | Spring Web MVC AOP |
| REST层 | 业务开发者 | 1. 可Maven依赖容器层，使用容器层启动内嵌容器进行测试和调试，容器曾通过jar包发布到maven库上 2. 单元测试：Junit/TestNG Mockito/EasyMock/Jmock | 1. 使用Maven打Jar包 2. 依赖Maven私服上容易jar包 3. 打包输入tar.gz格式，内容包含所有依赖的jar包 4. 提供Spring容器单入口athena-rest-context.xml，可供单元测试使用，开发测试，RPC导出，REST服务导出 | Spring Web MVC Apache CTX Jersey |
| 服务层 | 业务开发者 | 单元测试：Junit/TestNG Mockito/EasyMock/Jmock | 1.使用Maven打Jar包 2.由REST层依赖，Spring环境自由配置 | Spring Context |
| 数据层 | 业务开发者 | 1. 单元测试：Junit/TestNG Mockito/EasyMock/Jmock 2. 使用内存数据库做单元测试 | 1.使用Maven打Jar包 2.由REST层依赖，Spring环境自由配置 | Spring Context Coba？ |

# 详细设计

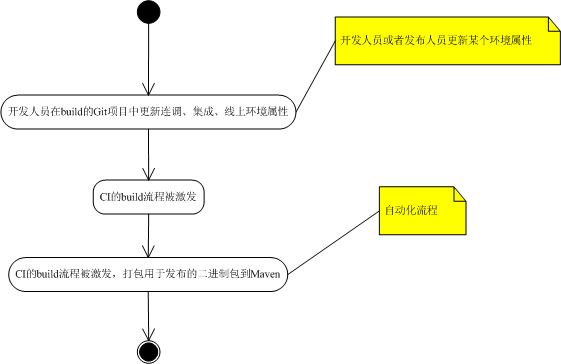
## 集成层架构设计

1. 环境属性管理

开发环境的属性保存在业务项目中，业务项目只保存开发属性，以免开发人员误用其他环境资源，尤其线上资源。

连调，集成，产品环境的属性统一托管在本项目中，并且在GIT中维护更新记录，每次属性被更新后，激活CI流程将本项目打包后发布到Maven私服。

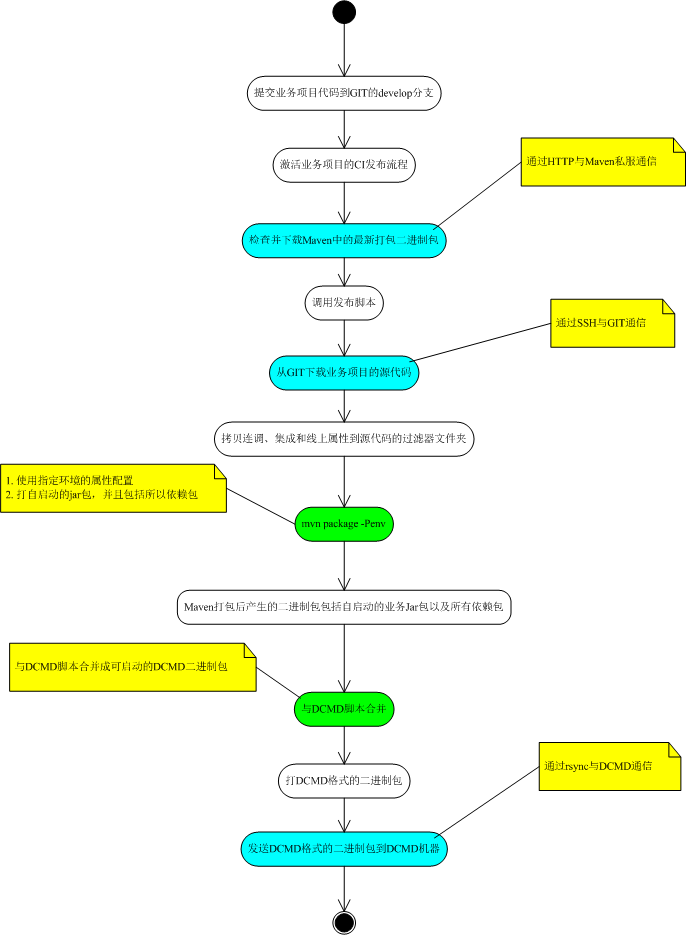
CI流程每次打包前检查Maven服务器，查看是否有更新的集成包，如果有则使用最近的包进行构建。



1. DCMD模板

维护DCMD启动停止脚本的模板以及业务项目的启动和停止脚本。

1. 发布构建流程



1. GC，GC日志，线程池设置

GC，GC日志，线程池在服务启动脚本server.sh中进行设置和优化，优化是针对通用程序的，如果某些应用想更改这些设置，需要在上线打包的属性中进行定制化配置。

#minThreads=1

#maxThreads=100

#GC\_OPTS=" -server -Xms512m -Xmx1g -Xmn128m -XX:PermSize=128m -Xss256k -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70"

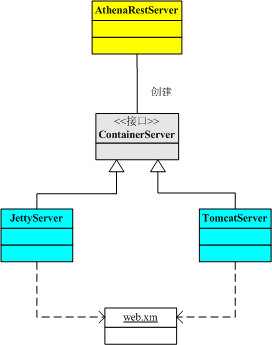
#GC\_LOG\_OPTS=" -verbose:gc -XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -XX:+PrintGCDetails -Xloggc:./logs/gc1.log"

#CUST\_OPT=" -XX:LargePageSizeInBytes=128m"

## 容器层架构设计

1. 内嵌HTTP服务器的互相转换

类图如下：

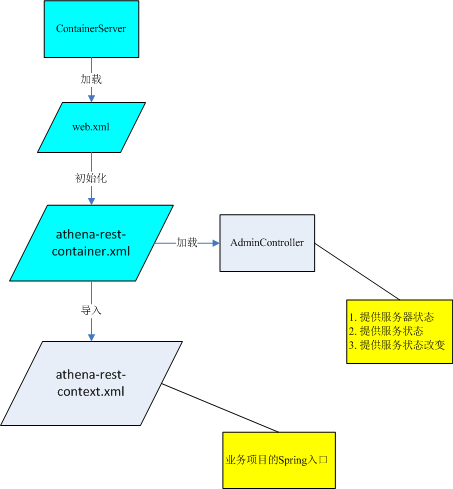


1. HTTP和HTTPs的转换

通过打包配置可指定HTTP和HTTPs的发布方式，实现通过容器层加载不同的容器连接器来实现。

1. Spring环境的加载以及加载层次

业务层需要在类路径提供spring/athena-rest-context.xml作为Spring环境的单一入口点，REST层下的服务层和数据层的Spring环境不做任何限制，可自由加载。



上图中web.xml默认是由容器提供的，如果业务项目有定制化的更改，我们提供业务项目进行覆盖的机会，可以通过把定制化的web.xml放入类路径的如下路径：

/webapp/WEB-INF/web-cust.xml

1. 打包方式

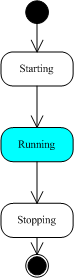
容器层是REST框架的一部分，属于公用组件，通过Maven私服来发布Jar包，此Jar包被Webapp层的Jar包依赖。

## Webapp层架构设计

1. Webapp层的管理模块

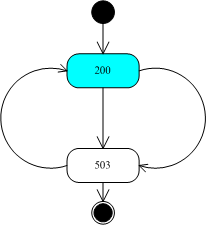
容器默认加载一个系统的Spring控制器类，在此控制器类中导出服务器状态，服务状态，以及提供服务状态上下线切换等系统管理服务。

服务器状态包括STARTING, RUNNING, STOPPING。状态机图如下：



服务器状态服务能够帮助CI流程进行集成环境测试的服务验证，CI流程发布服务后，可以根据此接口查询何时服务器已经初始化完毕，并且开始自动的服务验证。

服务状态包括200和503两种状态，这个状态是人为设置的，也就是管理员可以动态选择是否将当前机器上线或者下线。流程图如下：



提供查询200和503两种服务状态的接口是为了帮助7层代理上线服务机器，可用于无停机版本升级，服务池扩容和缩容等。

1. REST规范

* 取对象的字段

可以通过使用@RequestMapping的正则匹配来实现，URL格式定义如下：

/{id}:(id,first-name,last-name,industry)

* 取级联对象

可以通过使用@RequestMapping的正则匹配来实现，URL格式定义如下：

/{id}:[child1,child2]

* 指定返回格式

可以通过使用@RequestMapping的正则匹配来实现，并且需要使用Jaskson的json/xml默认转换器，URL格式定义如下：

@RequestMapping(value = "/pets/{petId}.json", method = RequestMethod.GET, produces="application/json")

@RequestMapping(value = "/pets/{petId}.xml", method = RequestMethod.GET, produces="application/xml")

* 指定分页范围

这个使用标准的URL参数映射，URL格式定义如下：

/dogs?limit=25&offset=50

分页逻辑属于业务特有的，不做任何通用的设计。

* 指定版本号

直接在URL上实现，默认版本为1，由7层将默认的转发至版本1。

1. 异常代码处理器

* DefaultHandlerExceptionResolver是默认注册的，能够处理Spring框架产生的各种异常，这些异常大多属于客户端异常，因此，默认返回HTTP 400错误码。
* 使用SimpleMappingExceptionResolver拦截所有的Exception，避免异常上抛导致容器线程挂掉。并且使用此异常解析器来处理常见的异常，例如：IO异常，非法参数异常，数据库异常等等。

Exception, RuntimeException以及没有处理的任何Throwable在这个层次都会被处理，并且返回相应的HTTP 500错误码。

* 业务层使用@ExceptionHandler来处理业务异常的返回，并且可以通过@ResponseStatus来注解异常，指定异常返回的http代码。
* 配置servlet容器的默认错误页面，返回通用错误的JSON串，并且返回相应的HTTP 500错误码。

1. Webapp层打包方式

Webapp层是REST框架的一部分，属于公用组件，通过Maven私服来发布Jar包，业务项目可依赖此Jar包，本Jar包是自启动的，业务项目可通过其启动容器做开发者测试，本Jar包依赖容器曾Jar包。

## REST层，服务层和数据层

1. REST层是服务导出层，服务的入口则是athena-rest-container.xml，服务通过Spring的Web MVC的Controller实现的。这个层次除了依赖Spring Web MVC，不依赖与任何框架，可使用Junit进行单元测试，也可依赖容器Jar包在Eclipse中进行调试。
2. 服务层提供原子操作，可供REST和RPC调用，REST层对外提供REST服务，服务层依赖底层的数据层。
3. 底层的数据层是与底层资源，例如：数据库，缓存，消息队列等，的集成层。

# 性能需求

由于REST服务框架运行在Web容器下，线程的调度由容器管理，需要调节Web容器的线程相关参数，资源的访问则由底层的服务层和数据层管理，不需要太多的性能设计，但是需要满足下面的性能指标。

|  |  |
| --- | --- |
| 样例服务接口 | QPS |
| 开启日志的Echo服务 | 8000+ |
| 关闭日志的Echo服务 | 20000+ |
| 数据库更新的服务 | 700+ |
| 数据库查询的服务 | 1000+ |