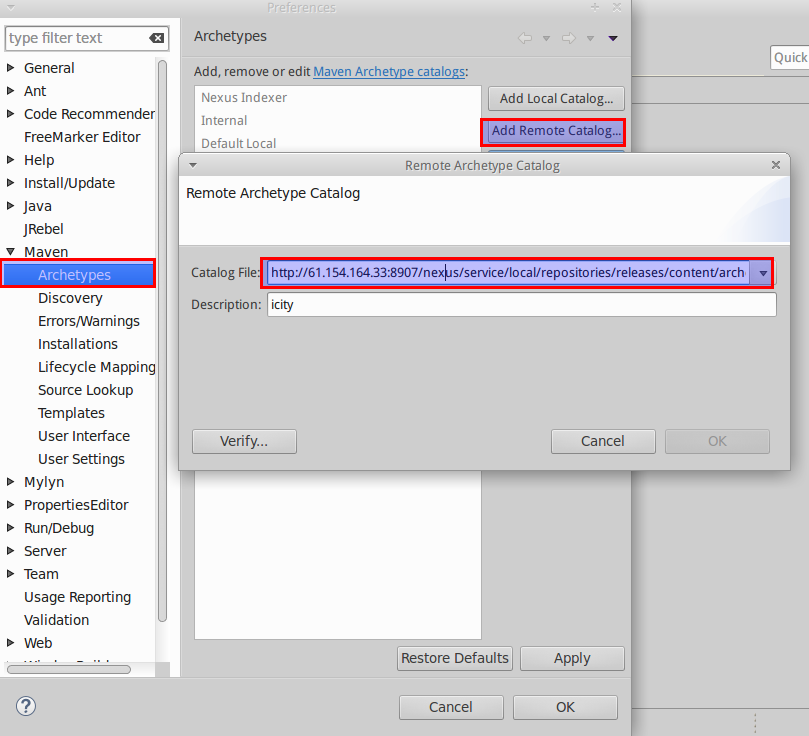
**当前框架版本: 1.2.0-beta**

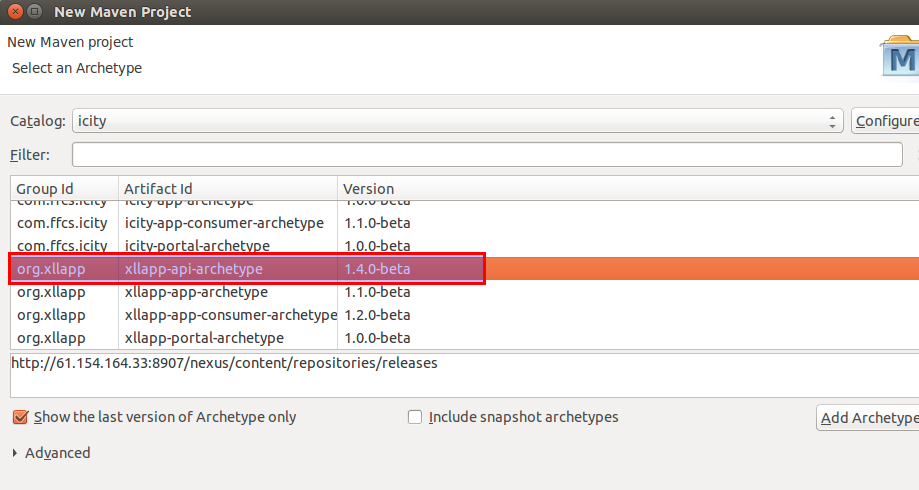
# 创建接口工程

在Eclipse中添加如下Archetype catalog：

http://61.154.164.33:8907/nexus/service/local/repositories/releases/content/archetype-catalog.xml



创建工程时选择如下Archetype：



# 接口工程介绍



# 开发接口

## 无缓存的接口

1,继承org.xllapp.api.core.BaseController类并实现其中的方法。

示例：

public class DemoController extends BaseController {

@Override

public Object resolveAndVerifyArgument(HttpServletRequest request)

throws Exception {

//解析参数

}

@Override

public String handleRequest(Object argument, HttpServletRequest request) throws Exception {

//处理请求

}

}

详见:工程demo包中的DemoController类.

2,在applicationContext-api.xml添加配置：

<bean id="/demo/add" class="org.xllapp.api.demo.controller.DemoController"/>

3,接口的访问URL：http://host:port/工程上下文名/demo/add

## 有缓存的接口

1,继承org.xllapp.api.core.CachedController类并实现其中的方法。

示例：

public class DemoCachedController extends CachedController {

@Override

public Object resolveAndVerifyArgument(final HttpServletRequest request) throws Exception {

//解析参数

}

@Override

public String handleRequest(Object argument, HttpServletRequest request) throws Exception {

//处理请求

}

@Override

public String getCacheKey(Object argument, HttpServletRequest request) {

//生成请求对应的key

}

@Override

public int getCacheExpiry(Object argument, HttpServletRequest request) {

//缓存过期时间

}

}

详见:工程demo包中的DemoCachedController类.

2,在applicationContext-api.xml添加配置：

<bean id="/demo/search" parent="abstractCachedController"

class="org.xllapp.api.demo.controller.DemoCachedController" />

3，接口的访问URL：http://host:port/应用文名/demo/search

## 触发事件

可在完成请求处理后，触发一个事件。具体的事件触发逻辑由

org.xllapp.api.core.event.EventFirer接口的实现类提供。

详见：工程demo包中的DemoFireEventController类。

注：1.2.0-beta版本开始提供此功能。

# 缓存配置

## 启用缓存

在applicationContext-support.xml文件中添加如下配置:

<bean id="cacheProvider" class="org.xllapp.cache.XMMemcCacheProvider">

<constructor-arg index="0">

<bean class="org.xllapp.cache.XMMemcCacheProvider.Config">

<property name="servers" value="${memcached.servers}" />

</bean>

</constructor-arg>

</bean>

<bean id="cacheTemplate" class="org.xllapp.cache.CacheTemplate">

<constructor-arg index="0" ref="cacheProvider" />

</bean>

<bean id="abstractCachedController" abstract="true">

<property name="cacheTemplate" ref="cacheTemplate" />

</bean>

注：默认提供基于memcached的缓存实现。

## 使用其他缓存实现

1,实现org.xllapp.cache.CacheProvider接口。

2,修改id为cacheProvider的bean的class为此实现类即可。

## 缓存控制

### 全局启用/禁用缓存

添加jvm启动参数icity.disable.cache=true|false。

### 单个接口启用/禁用缓存

通过CachedController类的setDisableCache()方法控制。

### 删除旧缓存

通过在请求中添加icityDeleteCache=true参数，来清空此请求的缓存。

# 消息队列配置

在applicationContext-support.xml文件中添加如下配置:

<bean id="jmsProducer" class="org.xllapp.jms.JMSProducer">

<constructor-arg index="0" value="${jms.brokerUrl}" />

<constructor-arg index="1" value="${jms.userName}" />

<constructor-arg index="2" value="${jms.password}" />

</bean>

注：1.2.0-beta版本开始提供此功能。

# 日志配置

## 异常采集

框架提供将异常信息采集汇总展示并告警功能。启用此功能，需要在application.\*.properties配置文件中添加如下配置（去除这些配置即可禁用此功能）：

#日志所属模块名

log.moduleName=icity-api-client-demo

#用于记录错误日志的ActiveMQ Broker的URL

log.jms.brokerUrl=tcp://localhost:60000

#用于记录错误日志的ActiveMQ Broker的用户名

log.jms.userName=system

#用于记录错误日志的ActiveMQ Broker的密码

log.jms.password=manager

## 日志输出控制

日志配置项(application.\*.properties文件中)说明:

#日志输出级别

log.logger.root.level=DEBUG

#日志输出格式

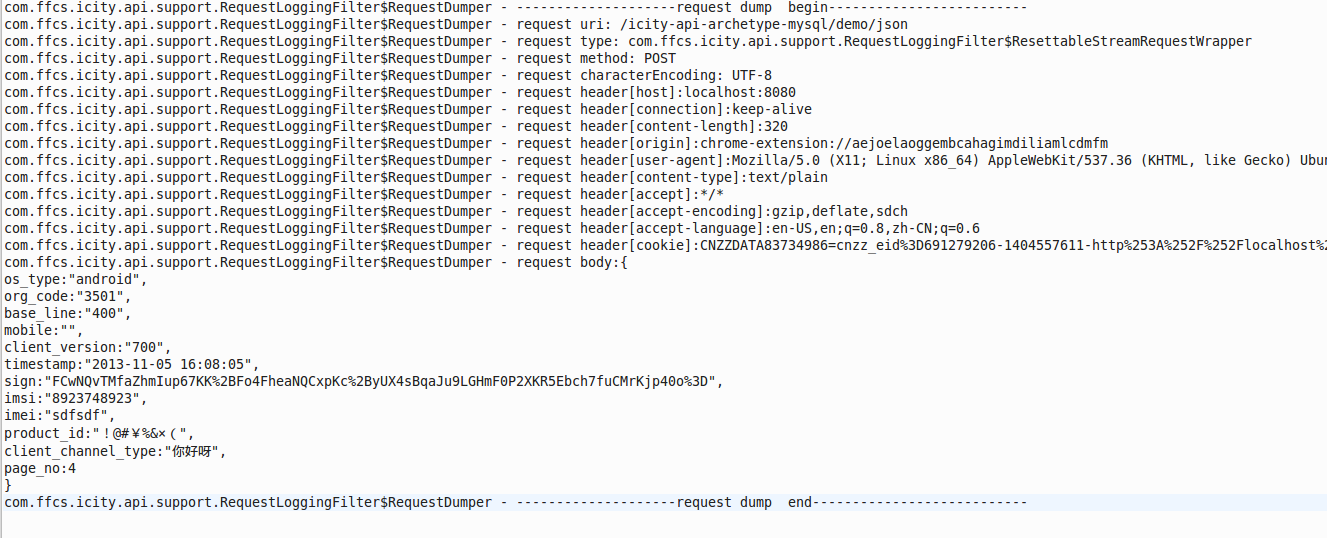
log.encoderPattern=%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] IP[%X{IP}] REQUEST\_ID[%X{REQUEST\_ID}] %-5level %logger{80} - %msg%n

#是否dump请求

log.isDumpRequest=true

## 请求Dump功能

启用此功能可以在日志中打印出请求的详细信息，包括请求头,请求体和请求参数等。



注：1.2.0-beta版本开始提供此功能。

## 如何看懂接口日志

每个接口日志条目额外包含有请求发起方的IP和请求标示（REQUEST\_ID）信息。通过IP可以过滤出此IP的所有接口日志；通过请求标示（REQUEST\_ID）可以过滤出同一个请求的所有日志（包括底层框架输出的日志）。

示例：

2014-08-29 17:17:54.470 [http-8080-1] IP[127.0.0.1] REQUEST\_ID[20140829171754-58] DEBUG org.xllapp.api.demo.controller.DemoFireEventController - response:{"timestamp":"2014-08-29 17:17:54","result\_code":"0","result\_desc":"请求成功"}

日志说明：

**IP[127.0.0.1]** - 表明此日志为IP为127.0.0.1的调用方发送的请求的日志**REQUEST\_ID[20140829171754-58]** - 表明此日志为标示为20140829171754-58的请求处理过程中输出的日志.可以使用20140829171754-58这个标示过滤出此请求处理过程中输出的全部日志。

# 配置

## 分环境配置

基于spring的profilie实现的分环境配置机制，当前支持开发,测试和生产这3种环境，分别对应application.development.properties，application.test.properties和application.properties这3个配置文件，只要将不同环境的配置写入对应的配置文件即可。可以在applicationContext-core.xml添加新的环境配置。

默认启用的是生产环境的配置。要启用特定的环境配置，可以通过添加jvm启动参数spring.profiles.active来实现（spring.profiles.active参数值：development表示生成环境；test表示测试环境。取值的规则是配置文件中间部分:application.<spring.profiles.active参数的值>.properties）。

## 运行时修改配置

框架提供在运行时临时修改配置的功能(注:这些修改在重启后会丢失)。此功能主要用于在不重启应用的情况下进行应用控制,如生成环境排错和临时启用请求dump功能等。

注：1.2.0-beta版本开始提供此功能。

### 添加自定义属性

使用此功能，需要将配置属性放在到org.xllapp.api.support.ApplicationConfig类中（只支持的基础类型和String类型的配置属性），然后通过访问/config页面进行修改。同时程序中需要通过ApplicationConfig使用此属性。

示例：

public class ApplicationConfig extends BaseApplicationConfig {

@FieldDescription("是否展示")

private boolean isShow=false;

public boolean isShow() {

return isShow;

}

}

访问http://host:port/应用文名/config页面进行修改。



### 修改框架的属性

在config页面中除了可以修改自定义的属性，还可以修改框架的属性。当前框架定义的可以在运行时修改的属性只有isDumpRequest，其控制是否启用dump请求功能。

### 修改properties文件中的属性

将properties文件中的属性注入到ApplicationConfig类中，程序中通过ApplicationConfig类访问这些属性。

示例：

properties文件：

#日志所属模块名

log.moduleName=xllapp-api-client-demo

ApplicationConfig类：

public class ApplicationConfig extends BaseApplicationConfig {

@Value("${log.moduleName}")

@FieldDescription("日志所属模块名")

private String moduleName;

public String getModuleName() {

return this.moduleName;

}

}