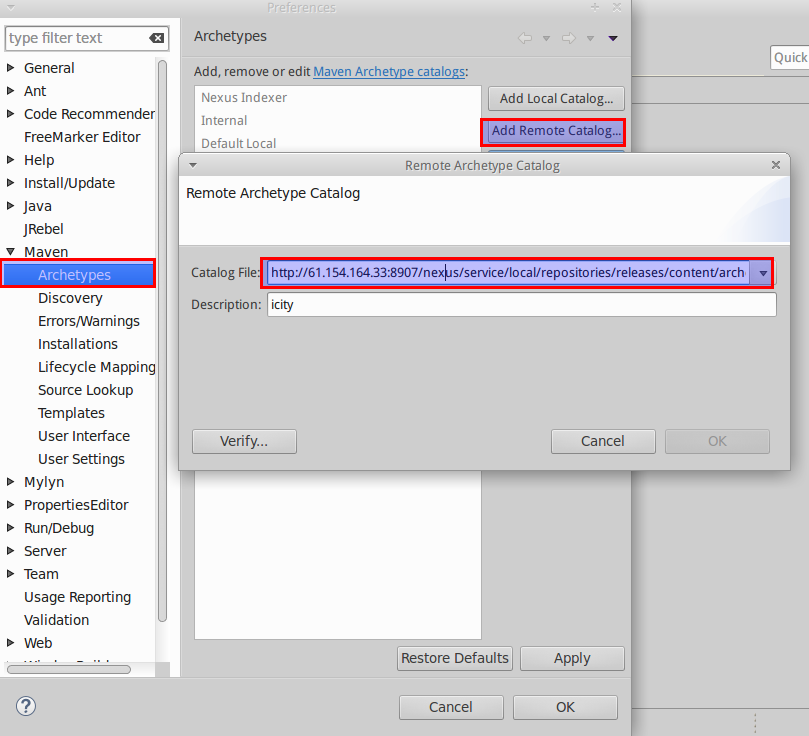
**当前框架版本: 1.3.0-beta**

**最后跟新时间：2014-09-22**

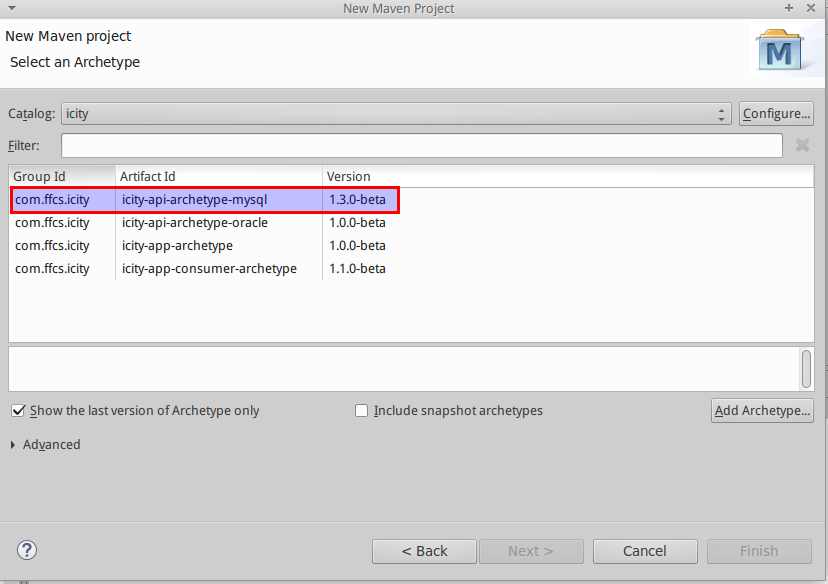
# 创建接口工程

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com.ffcs.icity.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="com.ffcs.icity.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com.ffcs.icity.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="com.ffcs.icity.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

com.ffcs.icity.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。

## 二期规范的接口

### 无缓存接口

1,继承com.ffcs.icity.api.core.JSONController类并实现其中的方法。

JSONController实现功能说明：

1,JSON格式请求参数的解析和响应内容转换成JSON；

2,验证二期规范中系统参数;

3,验证数字签名;

注：1.3.0-beta版本开始提供此功能。

示例：

假设接口请求如下：

{

"product\_id": "icity",

"client\_verion": "200",

"org\_code": "3501",

"client\_channel\_type": "icity-ver",

"os\_type": "android",

"imsi": "123",

"imei": "abc",

"mobile": "18900000000",

"timestamp": "2022-09-16 17:08:00",

"sign": "9qS0v1ImAICQbT8qFx6TAMZnLrOXPQBK6L%2FL%2Fhg4KuC%2BRw3m9RQ1ilTvzx7ovkzyyPFWHva0H%2BQ%3D"

}

Sign规则：

URLEncoding(BASE64(3DES(timestamp+$+MD5(imsi+$+imei+$+app\_key+$+timestamp))))

响应如下：

{

"result\_code": "0",

"result\_desc": "request successful",

"timestamp": "2014-09-18 10:40:27",

"data": {

"user\_name": "abc",

}

}

接口代码：

public class DemoJSONController extends JSONController{

/\*\*

\* 验证业务参数

\*/

@Override

protected void verifyBusiArgument(Map<String, Object> requestArgument) throws InvalidRequestArgumentException {

//验证电话号码是否为空

RequestArgumentAssert.assertNotBlankString(requestArgument, "mobile");

}

/\*\*

\* 返回生成签名中MD5部分需要的请求参数值.注意返回数组中元素的顺序与签名规则中的一致,无需包含app\_key和timestamp.详见JSONController.getSignItems()方法的注释.

\*/

@Override

protected String[] getSignItems(Map<String, Object> requestArgument) {

return new String[]{(String)requestArgument.get("imsi"),(String)requestArgument.get("imei")};

}

/\*\*

\* 处理请求

\*/

@Override

protected Object handleRequest(Map<String, Object> requestArgument) throws ApiException {

/\*//也可以使用map返回数据

Map<String,String> data=new HashMap<String, String>();

data.put("user\_name","abc");

return data;\*/

/\*\*

\* 将对象转换成JSON时会自动将驼峰方式命名的属性名转换成小写加下划线方式

\*/

return new DemoResponseData("abc");

}

public static class DemoResponseData{

private String userName;

public DemoResponseData(String userName) {

super();

this.userName = userName;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

}

}

详见:工程demo包中的DemoJSONController类.

### 有缓存接口

1,继承com.ffcs.icity.api.core.CachedJSONController类并实现其中的方法。

CachedJSONController实现功能说明：在继承JSONController的功能的基础上提供缓存功能.

注：1.3.0-beta版本开始提供此功能。

示例：

假设请求和响应和本节中的无缓存接口一样.

public class DemoCachedJSONController extends CachedJSONController {

/\*\*

\* 验证业务参数

\*/

@Override

protected void verifyBusiArgument(Map<String, Object> requestArgument) throws InvalidRequestArgumentException {

//验证电话号码是否为空 RequestArgumentAssert.assertNotBlankString(requestArgument, "mobile");

}

/\*\*

\* 返回生成签名中MD5部分需要的请求参数值.注意返回数组中元素的顺序与签名规则中的一致,无需包含app\_key和timestamp.详见JSONController.getSignItems()方法的注释.

\*/

@Override

protected String[] getSignItems(Map<String, Object> requestArgument) {

return new String[]{(String)requestArgument.get("imsi"),(String)requestArgument.get("imei")};

}

@Override

protected Object handleRequest(Map<String, Object> requestArgument) throws ApiException {

Map<String,String> data=new HashMap<String, String>();

data.put("user\_name","abc");

return data;

}

/\*\*

\* 处理请求

\*/

@Override

protected String getCacheKey(Map<String, Object> requestArgument) {

return (String)requestArgument.get("imsi")+"."+requestArgument.get("imei");

}

@Override

protected int getCacheExpiry(Map<String, Object> requestArgument) {

return 120;

}

}

详见:工程demo包中的DemoCachedJSONController类.

### 请求签名验证

1, 对于父类为JSONController和CachedJSONController类的接口类,实现getSignItems()方法返回需要的数据即可。具体详见JSONController. getSignItems()的方法注释.

protected String[] getSignItems(Map<String, Object> requestArgument)方法说明：

方法说明：返回生成请求签名中MD5部分需要的请求参数值.

注意：

1,返回的数组中不能包含app\_key和timestamp;

2,验证时会自动在此方法返回的数组后面依次添加app\_key和timestamp(既在数字签名规则的MD5部分必须是这样的:MD5(item1+$+...+$+itemN+$+app\_key+$+timestamp)

3,返回数组中元素的顺序与签名规则中的一致;

示例：

如果sign规则为：URLEncoding(BASE64(3DES(timestamp+$+MD5(imsi+$+imei+$+app\_key+$+timestamp))))

那么此方法返回的数组为：

new String[]{(String)requestArgument.get("imsi"),(String)requestArgument.get("imei")}

2,对于其他的接口类,可以使用RequestUtils类的verifySign()方法验证数组签名。具体详见方法注释.

public static boolean verifySign(String sign, String timestampInRequest, String[] signItems, String des3Key, String appKey, int expiry)方法说明：

方法说明: 验证请求的数字签名.

参数说明:

sign:待校验的数字签名串

timestampInRequest:请求中的时间戳,格式：yyyy-MM-dd HH:mm:ss

signItems:生成请求签名中MD5部分需要的内容

注意：

1,数组中不能包含app\_key和timestamp;

2,此方法会在验证时自动在此参数的数组后面依次添加app\_key和timestamp(既在数字签名规则的MD5部分必须是这样的:MD5(item1+$+...+$+itemN+$+app\_key+$+timestamp)

3,返回数组中元素的顺序与签名规则中的一致;

示例：

如果sign规则为：URLEncoding(BASE64(3DES(timestamp+$+MD5(imsi+$+imei+$+app\_key+$+timestamp)))),

假设imsi为123，imei为abc,

那么此参数为：new String[]{"123","abc")}.

des3Key:3DES加密的Key

appKey:客户端的appKey

expiry:请求过期时间,单位:秒

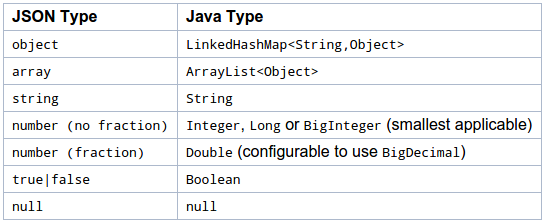
注：1.3.0-beta版本开始提供此功能。

### 如何获取HttpServletRequest对象和HttpServletRequest对象

通过com.ffcs.icity.api.support包下面的RequestContextHolder类的静态方法 getRequest()和getResponse()获取请求和响应对象。

### 请求参数转换成map的规则

在父类为JSONController和CachedJSONController的子类中，已将请求中的JSON参数转换成map，转换规则如下：



示例:

Json参数:

{

"string": "test",

"array": [

"array\_item1",

"array\_item2"

],

"map": {

"obj\_field2": "abc",

"obj\_field1": "123"

},

"bool": true,

"i": 10

}

由转换后的map中获取参数：

List<String> array=(List<String>)map.get("array");

Map<String,Object> map1=(Map<String,Object>)map.get("map");

String string=(String)map.get("string");

Boolean bool=(Boolean)map.get("bool");

Integer i=(Integer)map.get("i");

### 响应对象转换成json的规则

对于protected Object handleRequest(Map<String, Object> requestArgument) throws ApiException 方法返回的对象的转换规则：

**Map：**

Map<String,String> data=new HashMap<String, String>();

data.put("user\_name","abc");

data.put("createTime","2014-09-11 12:00:00");

转换后的json:

{

"data": {

"createTime": "2014-09-11 12:00:00",

"user\_name": "abc"

},

"result\_code": "0",

"result\_desc": "request successful",

"timestamp": "2014-09-19 11:42:39"

}

**Bean:**

public class Bean{

private String user\_name="test";

private String createTime="2014-09-11 12:00:00";

public String getUser\_name() {

return user\_name;

}

public void setUser\_name(String user\_name) {

this.user\_name = user\_name;

}

public String getCreateTime() {

return createTime;

}

public void setCreateTime(String createTime) {

this.createTime = createTime;

}

}

转换后的json:

{

"data": {

"user\_name": "test",

"create\_time": "2014-09-11 12:00:00"

},

"result\_code": "0",

"result\_desc": "request successful",

"timestamp": "2014-09-19 11:44:25"

}

**List/数组：**

public class Bean{

private String user\_name="test";

private String createTime="2014-09-11 12:00:00";

……..

}

List<Bean> list=new ArrayList<JSONHelperTest.Bean>();

list.add(bean);

list.add(bean);

或

Bean[] beans=new Bean[2];

beans[0]=bean;

beans[1]=bean;

转换后的json:

{

"data": [

{

"user\_name": "test",

"create\_time": "2014-09-11 12:00:00"

},

{

"user\_name": "test",

"create\_time": "2014-09-11 12:00:00"

}

],

"result\_code": "0",

"result\_desc": "request successful",

"timestamp": "2014-09-19 11:50:47"

}

说明：Bean对象（不管对象是直接返回的还是在返回的map,list和数组中）中驼峰格式命名的属性,转换时会被转换成小写加下划线方式(如例子中Bean对象的createTime属性转换成create\_time)。

### 请求参数验证

二期接口规范中定义的系统参数部分的验证框架已实现（包括非空验证，时间戳验证和数字签名验证等），继承自JSONController和CachedJSONControlle的子类只需要实现public void verifyBusiArgument(…) throws InvalidRequestArgumentException方法实现业务参数的验证即可。

# 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="com.ffcs.icity.cache.XMMemcCacheProvider">

<constructor-arg index="0">

<bean class="com.ffcs.icity.cache.XMMemcCacheProvider.Config">

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

</bean>

</constructor-arg>

</bean>

<bean id="cacheTemplate" class="com.ffcs.icity.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com.ffcs.icity.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="com.ffcs.icity.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

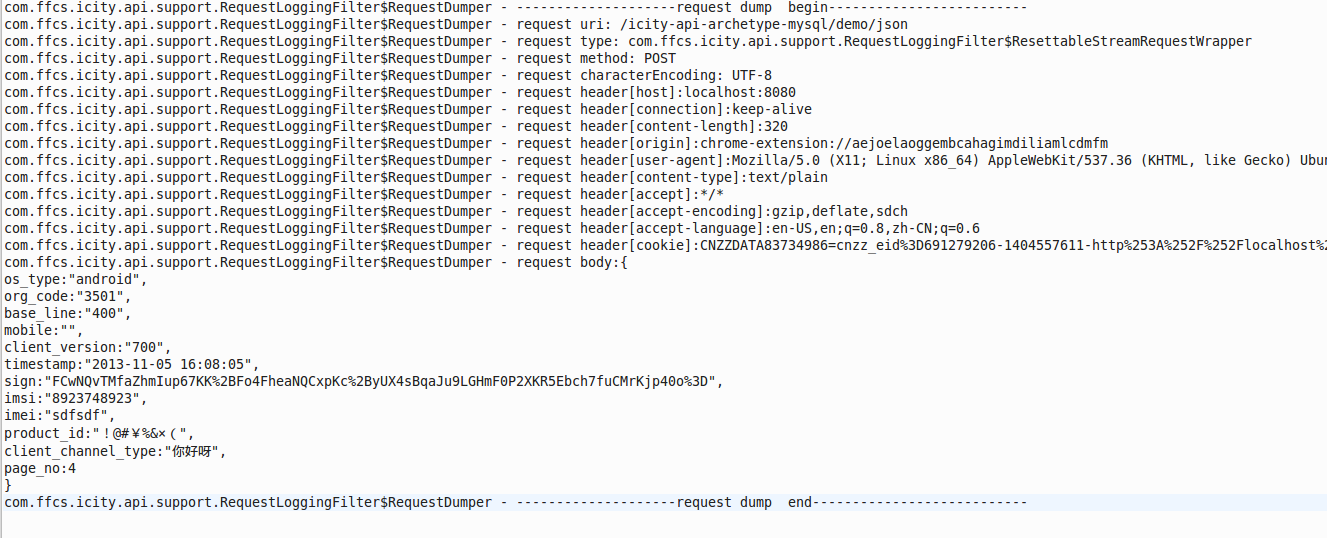
log.encoderPattern=%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] IP[%X{IP}] REQUEST\_ID[%X{REQUEST\_ID}] %-5level %logger{80} - %msg%n

#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 com.ffcs.icity.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profile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com.ffcs.icity.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。



在接口中使用此属性：

示例：

public class DemoController extends BaseController {

@Override

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

ApplicationConfig applicationConfig =

ApplicationConfigHolder.getApplicationConfig();

boolean isShow=applicationConfig.isShow();

....

}

}

### 修改框架的属性

在congig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，其时控制是否启用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=icity-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;

}

}