## 1、功能说明

zPaaS包提供的功能封装包括：

（1）统一配置中心（基于Zookeeper）

（2）统一Session管理（基于Redis）

（3）统一缓存管理（基于Redis）

（4）统一普通文件管理（基于MongoDB、GridFS）

（5）统一机密文件管理（基于MongoDB、GridFS）

（6）统一交易日志管理（基于MongoDB）

（7）统一系统日志管理（基于Log4j、Kafka和MongoDB）

（8）消息生产和消费框架封装（基于Kafka）

（9）统一签名和签名验证管理（基于统一机密文件管理）

（10）规则引擎服务（基于Drools）

（11）分布式资源调度器（基于Zookeeper）

（12）本地缓存服务

（13）对称加密工具

（14）各种工具类

## 2、统一配置中心（基于Zookeeper）

基于Zookeeper提供统一配置和实时推送功能框架，平台提供的所有功能都集成到了统一配置中心中，是整个zPaaS平台最核心的组件。

（1）配置管理中心类ConfigurationCenter

（2）spring配置文件举例

<bean id=*"cc"* class=*"com.zpaas.ConfigurationCenter"* init-method=*"init"*>

<constructor-arg value=*"10.211.55.5:2181,10.211.55.6:2181,10.211.55.7:2181"*/>

<constructor-arg value=*"P"*/>

<property name=*"auth"* value=*"7e6161fe018bd4054414cbc9999fee89f9110662f8981993"*/>

</bean>

（3）集成举例

（4）部署说明

至少3个Zookeeper实例（建议大于3的奇数个实例），组成Zookeeper集群

## 3、统一Session管理（基于Redis）

基于Redis提供统一Session管理，将原httpsession中的内容统一存放到redis集群中，使web应用成为无状态的服务实例，提供更好的高可用应用，减少应用实例故障对用户的影响，同时能够使负载均衡设备采用无状态分发的方式，提高性能。

平台提供的统一Session管理方案对应用开发没有侵入。

（1）Session管理核心类SessionManager

（2）Filter类CacheSessionFilter

（3）Web.xml配置说明

<filter>

<display-name>CacheSessionFilter</display-name>

<filter-name>CacheSessionFilter</filter-name>

<filter-class>com.zpaas.session.CacheSessionFilter</filter-class>

<init-param>

<param-name>ignore\_suffix</param-name>

<param-value>.png,.jpg,.jpeg,.gif,.css,.js,.html,.htm</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CacheSessionFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

（4）Spring配置文件

<bean id=*"sessionManager"* class=*"com.zpaas.session.SessionManager"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/com/zpaas/session/sessionSVC"*/>

</bean>

（5）部署说明

根据Redis应用服务器的性能，可以采用主从模式或集群模式

（6）配置说明

*/com/zpaas/session/sessionSVC* {'host':'10.211.55.5','port':'6379','timeOut':'20000','maxActive':'100','maxIdle':'100','maxWait':'1000','testOnBorrow':'true','testOnReturn':'true','dbIndex':'0','expirationUpdateInterval':'300','maxInactiveInterval':'1800'}

## 4、统一缓存管理（基于Redis）

基于Redis提供统一的调整缓存服务，可用于存放一些静态数据或写少读多的数据

（1）核心服务接口类RemoteCacheSVC

（2）核心服务实现类RemoteCacheSVCImpl

（3）Spring配置文件

<bean id=*"remoteCacheSVC"* class=*"com.zpaas.cache.remote.RemoteCacheSVCImpl"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/zpaas/cache/remoteCacheSVC"*/>

</bean>

（4）服务方法列表

public void addItemToList(String key, Object object);

public List getItemFromList(String key);

public void addItem(String key, Object object);

public void addItem(String key, Object object, int seconds);

public String flushDB();

public Object getItem(String key);

public void delItem(String key);

public long getIncrement(String key);

public void setHashMap(String key, HashMap<String, String> map);

public Map<String, String> getHashMap(String key);

public void addSet(String key, Set<String> set);

public Set<String> getSet(String key);

（5）使用说明

RemoteCacheSVC remoteCache = (RemoteCacheSVC)PaasContextHolder.*getContext*().getBean("remoteCacheSVC");

remoteCache.addItem("testKey", "testValue");

System.***out***.println(remoteCache.getItem("testKey"));

remoteCache.delItem("testKey");

（6）部署说明

根据Redis应用服务器的性能，可以采用主从模式或集群模式

（7）配置说明

*/zpaas/cache/remoteCacheSVC* {'host':'10.211.55.5','port':'6379','timeOut':'20000','maxActive':'100','maxIdle':'100','maxWait':'1000','testOnBorrow':'true','testOnReturn':'true','dbIndex':'1'}

## 5、统一普通文件管理（基于MongoDB、GridFS）

基于MongoDB、GridFS提供统一的文件管理服务，如附件、图片等

（1）核心文件管理服务接口类FileManagerSVC

（2）核心文件管理服务实现类MongoFileManagerSVCImpl

（3）服务方法列表

public String saveFile(String fileName, String fileType);

public String saveFile(byte[] byteFile, String fileName, String fileType);

public byte[] readFile(String fileId);

public void readFile(String fileId, String localFileName);

public void deleteFile(String fileId);

public byte[] readFileByName(String fileName);

public void readFileByName(String fileName, String localFileName);

public void deleteFileByName(String fileName);

（4）Spring配置文件

<bean id=*"fileSVC"* class=*"com.zpaas.file.MongoFileManagerSVCImpl"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/zpaas/file/fileSVC"*/>

</bean>

（5）使用说明

FileManagerSVC fileSVC = (FileManagerSVC)PaasContextHolder.*getContext*().getBean("fileSVC");

String fileId = fileSVC.saveFile("/Users/ZhangJunyong/Downloads/bank.gif", "gif");

fileSVC.readFile(fileId, "/Users/ZhangJunyong/Downloads/bank\_bak.gif");

fileSVC.deleteFile(fileId);

（6）部署说明

根据服务压力可以采用复制组或集群的模式

（7）配置说明

*/zpaas/file/fileSVC* {fileServer:[{ip:'10.211.55.5',port:'27017'},{ip:'10.211.55.6',port:'27017'},{ip:'10.211.55.7',port:'27017'}],fileRepo:'zpaasFileDB',username:'zpaasFileDB',password:'32eafe07c8d1a4db'}

## 6、统一机密文件管理（基于MongoDB、GridFS）

与统一普通文件管理基本相同，只是配置有区别。统一机密文件主要用于存储密钥、私钥、公钥、电子章等需要加密存储的机密文件。

（1）Spring配置文件

<bean id=*"secureFileSVC"* class=*"com.zpaas.file.MongoFileManagerSVCImpl"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/zpaas/file/secureFileSVC"*/>

</bean>

（2）配置文件

*/zpaas/file/secureFileSVC* {fileServer:[{ip:'10.211.55.5',port:'27017'},{ip:'10.211.55.6',port:'27017'},{ip:'10.211.55.7',port:'27017'}],fileRepo:'zpaasSecureFileDB',username:'zpaasSecureFileDB',password:'d10bb6123a72de91072842041c49dca8'}

## 7、统一交易日志管理（基于MongoDB）

基于MongoDB提供统一的交易日志管理，主要用于存放交易报文、接口报文等重要的业务日志

（1）核心服务接口类LogSVC

（2）核心服务实现类MongoLogSVCImpl

（3）服务方法列表

public void write(String log);

public void write(JSONObject logJson);

public void write(Map logMap);

（4）Spring配置文件

<bean id=*"tradeLogSVC"* class=*"com.zpaas.log.MongoLogSVCImpl"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/zpaas/log/tradeLogSVC"*/>

</bean>

（5）使用说明

LogSVC tradeLogSVC = (LogSVC) PaasContextHolder.*getContext*().getBean("tradeLogSVC");

tradeLogSVC.write("{\"info\":\"trade\_log\_test\"}");

（6）部署说明

根据服务压力可以采用复制组或集群的模式

（7）配置说明

*/zpaas/log/tradeLogSVC* {logServer:[{ip:'10.211.55.5',port:'27017'},{ip:'10.211.55.6',port:'27017'},{ip:'10.211.55.7',port:'27017'}],logRepo:'zpaasTradeLogDB',logPath:'zpaasTradeLogCollection',username:'zpaasTradeLogDB',password:'32eafe07c8d1a4db'}

## 8、统一系统日志管理（基于Log4j、Kafka和MongoDB）

统一系统日志管理的组件如下：

统一系统日志管理通过扩展Log4j的Appender来实现写日志，对应用开发没有侵入；

通过Kafka的异步消息来减少对应用性能的影响；

在服务端需要部署一个系统日志消息消费程序，用于消费所有的系统日志消息并插入到MongoDB中；

同时采用MongoDB来统一存储系统日志，做到对系统日志的统一管理。

如果遵循一定的开发规范，可使用统一的threadId将整个执行堆栈串联起来，重现当时的执行堆栈。

（1）核心服务接口类LogSVC

（2）核心服务实现类MongoLogSVCImpl、MessageLogAppender、KafkaLogMessageListener

（3）服务方法列表

public void write(String log);

public void write(JSONObject logJson);

public void write(Map logMap);

（4）Spring配置文件

<log4j:configuration xmlns:log4j=*"http://jakarta.apache.org/log4j/"*>

<appender name=*"STDOUT"* class=*"org.apache.log4j.ConsoleAppender"*>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"%d %-5p %c.%M:%L - %m%n"*/>

</layout>

</appender>

<appender name=*"RollingFile"* class=*"org.apache.log4j.RollingFileAppender"*>

<param name=*"MaxFileSize"* value=*"100MB"*/>

<param name=*"File"* value=*"zpaas\_logMessageServer.log"*/>

<param name=*"Append"* value=*"true"* />

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"%d %-5p %c.%M:%L - %m%n"*/>

</layout>

</appender>

<!-- -->

<appender name=*"MSGOUT"* class=*"com.zpaas.log.MessageLogAppender"*>

<param name=*"logTopic"* value=*"paas\_log\_mongo\_topic"*/>

<param name=*"appName"* value=*"zpaas\_template"*/>

<param name=*"confPath"* value=*"/zpaas/message/logMessageSender"*/>

<param name=*"ccAddr"* value=*"192.168.1.91:2181,192.168.1.92:2181,192.168.1.93:2181"*/>

<param name=*"runMod"* value=*"P"*/>

<param name=*"auth"* value=*"dc7727d07afc803b6fdbb06914fb52e014d6256e5e8711cf"*/>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"%d %-5p %c.%M:%L - %m%n"*/>

</layout>

<filter class=*"org.apache.log4j.varia.LevelRangeFilter"*>

<param name=*"levelMin"* value=*"INFO"* />

<param name=*"AcceptOnMatch"* value=*"true"* />

</filter>

</appender>

<!-- specify the logging level for loggers from other libraries -->

<logger name=*"com.alibaba"*>

<level value=*"error"* />

</logger>

<!-- 注意：以下配置需要保留，否则在使用MessageLogAppender时会造成死循环 -->

<!-- pay attention: following configuration should be retained for MessageLogAppender -->

<logger name=*"com.zpaas.log.KafkaLogMessageListener"* additivity=*"false"*>

<level value=*"DEBUG"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<logger name=*"com.zpaas.message.MessageSender"* additivity=*"false"*>

<level value=*"DEBUG"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<logger name=*"com.zpaas.ConfigurationCenter"* additivity=*"false"*>

<level value=*"DEBUG"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<logger name=*"com.zpaas.log.MongoLogSVCImpl"* additivity=*"false"*>

<level value=*"DEBUG"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<logger name=*"org.springframework"* additivity=*"false"*>

<level value=*"WARN"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<logger name=*"kafka"* additivity=*"false"*>

<level value=*"WARN"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<logger name=*"org.apache.zookeeper"* additivity=*"false"*>

<level value=*"WARN"* />

<appender-ref ref=*"STDOUT"* />

</logger>

<!-- end of pay attention-->

<!-- for all other loggers log only info and above log messages -->

<root>

<priority value=*"debug"*/>

<appender-ref ref=*"STDOUT"* />

<appender-ref ref=*"MSGOUT"* />

</root>

</log4j:configuration>

（7）部署说明

Kafka采用集群模式

MongoDB根据服务压力可以采用复制组或集群的模式

消息消费端根据Kafka主题的分区情况可用部署多个实例的方式

（8）配置说明

*/com/zpaas/log/sysLogSVC* {logServer:[{ip:'10.211.55.5',port:'27017'},{ip:'10.211.55.6',port:'27017'},{ip:'10.211.55.7',port:'27017'}],logRepo:'zpaasLogDB',logPath:'zpaasLogCollection',username:'zpaasLogDB',password:'32eafe07c8d1a4db'}

*/zpaas/message/logMessageSender* {metadata.broker.list:'10.211.55.5:9092,10.211.55.6:9092,10.211.55.7:9092',serializer.class:'com.zpaas.message.MessageEncoder',key.serializer.class:'kafka.serializer.StringEncoder',partitioner.class:'com.zpaas.message.MessageIdPartitioner',request.required.acks:0,producer.type:'async'}

/com/zpaas/message/messageConsumer {zookeeper.connect:'10.211.55.5:2181,10.211.55.6:2181,10.211.55.7:2181/kafka',msg.processor.num:'6'}

（9）堆栈重现代码规范

a、在web应用的filter中添加以下代码

String clientAddr = CommonUtil.getClientAddr(request);

ThreadId.setThreadIdWithIP(clientAddr);

b、在涉及跨JVM调用的调用方添加以下代码

//构造请求服务报文

PacketVerifyReq req = new PacketVerifyReq();

req.setPayPacket(payPacket.getRequestPacket());

req.setPartnerDomain(CommonTools.getDomainNameFromReferer(request));

req.setClientIp(CommonTools.getClientAddr(request));

req.setThreadId(ThreadId.getThreadId());

//校验请求报文

PacketVerifyRes res = paySV.verifyPayPacket(req);

c、在服务的入口方法

ThreadUtils.setThreadId(packetVerifyReq);

d、在所有的方法调用起始

log.info("Enter method verifyPayPacket, param is:" + packetVerifyReq.toString());

## 9、消息生产和消费框架封装（基于Kafka）

使用Kafka作为平台的消息中枢，提供统一的消息服务

（1）核心类MessageSender、MessageListener

（2）Spring配置文件

<bean id="messageSender" class="com.zpaas.message.MessageSender" init-method="init">

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

<property name="confPath" value="/com/zpaas/message/messageSender"/>

</bean>

（3）消息端Spring配置文件

<bean id="logMessageListener" class="com.zpaas.log.KafkaLogMessageListener">

<property name="logTopic" value="paas\_log\_mongo\_topic"/>

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

</bean>

<bean id="messageConsumer" class="com.zpaas.message.MessageConsumer" init-method="init">

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

<property name="listener" ref="logMessageListener"/>

<property name="confPath" value="/com/zpaas/message/messageConsumer"/>

<property name="groupId" value="kafka\_log\_message\_group"/>

<property name="monitorTopicList">

<list>

<value>paas\_log\_mongo\_topic</value>

</list>

</property>

</bean>

（4）生产端使用说明

MessageSender sender = (MessageSender)PaasContextHolder.getContext().getBean("messageSender");

sender.sendMessage("test", "message-topic");

（5）消费端使用说明

public void receiveMessage(Message message, MessageStatus status) {

if(logTopic.equals(message.getTopic())) {

if(log.isDebugEnabled()) {

log.debug("get log message: " + message.getMsg());

}

logWriter.write(JSONObject.fromObject(message.getMsg()));

}

}

（6）部署说明

Kafka采用集群模式

（7）配置说明

/com/zpaas/message/messageSender {metadata.broker.list:'10.211.55.5:9092,10.211.55.6:9092,10.211.55.7:9092',serializer.class:'com.zpaas.message.MessageEncoder',key.serializer.class:'kafka.serializer.StringEncoder',partitioner.class:'com.zpaas.message.MessageIdPartitioner',request.required.acks:1,producer.type:'sync'}

/com/zpaas/message/messageConsumer {zookeeper.connect:'10.211.55.5:2181,10.211.55.6:2181,10.211.55.7:2181/kafka',msg.processor.num:'6'}

## 10、签名和签名验证管理（基于统一机密文件管理）

提供签名和签名验证服务

（1）核心类SignatureSVC

（2）服务方法清单

public String sign(String plainText)；

public String sign(String plainText, String charset)；

public boolean verifySignature(String sign, String plainText)；

public boolean verifySignatureWithCharset(String sign, String plainText, String charset)；

public boolean verifySignature(String sign, String plainText, String cerFile)；

public boolean verifySignatureWithCharset(String sign, String plainText, String cerFile, String charset)；

（3）Spring配置文件

<bean id="signature" class="com.zpaas.security.signature.SignatureUtil" init-method="init">

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

<property name="confPath" value="/com/zpaas/secure/signatureSVC"/>

<property name="secureFileRepo" ref="secureFileSVC"/>

</bean>

（4）使用说明

SignatureSVC util = (SignatureSVC)PaasContextHolder.getContext().getBean("signature");

String sign = util.sign(strSendData,"GBK");

System.out.println(sign);

boolean ret = util.verifySignatureWithCharset(sign, strSendData, "GBK");

（5）配置说明

/com/zpaas/file/secureFileSVC {fileServer:[{ip:'10.211.55.5',port:'27017'},{ip:'10.211.55.6',port:'27017'},{ip:'10.211.55.7',port:'27017'}],fileRepo:'zpaasSecureFileDB',username:'zpaasSecureFileDB',password:'d10bb6123a72de91072842041c49dca8'}

## 11、规则引擎服务（基于Drools）

提供规则引擎服务

（1）核心服务接口类：com.zpaas.rule.RuleSVC

（2）核心服务实现类：com.zpaas.rule.DroolsRuleSVCImpl

（3）服务方法清单：

public void executeRule(Object...objects);

public void executeRuleById(String ruleId, Object...objects);

（4）Spring配置文件示例

<bean id="ruleRepo" class="com.zpaas.file.MongoFileSVCImpl" init-method="init">

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

<property name="confPath" value="/com/zpaas/file/ruleSVC"/>

</bean>

<!-- -->

<bean id="zpaasRule" class="com.zpaas.rule.DroolsRuleSVCImpl" init-method="init">

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

<property name="ruleRepo" ref="ruleSVC"/>

<property name="confPath" value="/com/zpaas/rule/ruleSVC"/>

</bean>

（5）配置示例

create /com/zpaas/rule/ruleSVC {'ruleId':'53a8d4054728a640fbaf8eee'}

（6）规则文件示例

package com.zpaas.rule

import com.zpaas.front.vo.PayInfo

rule "zpaasRule"

no-loop true

when

payInfo : PayInfo();

then

if("402".equals(payInfo.getBankNo())) {//广西农信

if("3".equals(payInfo.getServiceType()) {

payInfo.setResult("-1");

}else {

payInfo.setResult("2");

}

}else {//好易联

payInfo.setResult("1");

}

end

（7）代码示例

RuleSVC ruleService = (RuleSVC)PaasContextHolder.getContext().getBean("zpaasRule");

try {

ruleService.executeRule(payInfo);

} catch (Exception e) {

e.printStackTrace();

log.error("Exception:" + e);

payInfo=null;

}

## 12、分布式资源调度器（基于zookeeper）

分布式资源调度器主要实现分布式资源的调度任务，方便应用程序实现高可用的分布式任务系统。

（1）核心类：com.zpaas.dispatcher.ResourceDispatcher

（2）扩展接口类：com.zpaas.dispatcher.ResourceExecutor

（3）扩展实现方法清单

public void modifyAllResources(JSONArray allResources);

public void prepareResourceAllocate(JSONArray resources);

public void commitResourceAllocate(JSONArray resources);

public boolean canReleaseResource(String resource);

（4）Spring配置文件

<bean id="resourceDispatcher" class="com.zpaas.dispatcher.ResourceDispatcher" init-method="init">

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

<property name="confPath" value="/com/zpaas/dispatcher/conf"/>

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

<property name="resourceId" value="testResource"/>

</bean>

<bean id="executor" class="com.zpaas.dispatcher.ResourceExecutorTest" >

（5）配置信息

create /com/zpaas/dispatcher/conf {zkServer:'133.0.43.195:2181,133.0.43.196:2181,133.0.43.196:2182',resources:['r0','r1','r2','r3','r4']}

（6）代码示例

public class ResourceExecutorTest implements ResourceExecutor {

private JSONArray resources = null;

@SuppressWarnings("unused")

private JSONArray allResources = null;

private Object lock = new Object();

@Override

public void modifyAllResources(JSONArray allResources) {

this.allResources = allResources;

}

@Override

public void prepareResourceAllocate(JSONArray resources) {

synchronized(lock) {

System.out.println("prepareResourceAllocate:" + resources);

try {

Thread.sleep(3\*1000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("prepareResourceAllocate: clear resources");

this.resources = null;

}

}

@Override

public void commitResourceAllocate(JSONArray resources) {

synchronized(lock) {

System.out.println("commitResourceAllocate:" + resources);

this.resources = resources;

}

}

@Override

public boolean canReleaseResource(String resource) {

System.out.println("canReleaseResource:" + resources);

if(resources != null && resources.contains(resource)) {

System.out.println("canReleaseResource: false");

return false;

}else {

System.out.println("canReleaseResource: true");

return true;

}

}

}

## 13、本地缓存

统一配置信息管理模块主要将配置信息在zookeeper中统一进行管理，基于zookeeper的订阅发布模式，能够将变动的配置信息实时推送到所有的订阅实例上，并在本地缓存中保存以提高访问效率。

1. 核心类：com.zpaas.cache.local.LocalCacheSVC

com.zpaas.cache.local.LocalCacheSVCImpl

（2）扩展实现方法清单

public Object getConfiguration(String confName)

public String getConfigurationString(String confName)

public JSONObject getConfigurationJSONObject(String confName)

public JSONArray getConfigurationJSONArray(String confName)

（3）Spring配置文件

<bean id=*"localCacheSVC"* class=*"com.zpaas.cache.local.LocalCacheSVCImpl"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/com/zpaas/cache/localCacheSVC"*/>

</bean>

（4）配置信息

create */com/zpaas/cache/localCacheSVC* {" appName":"TEST"}

（6）代码示例

LocalCacheSVC localCache = (LocalCacheSVC)PaasContextHolder.*getContext*().getBean("localCacheSVC");

System.***out***.println(localCache.getCacheItemString("appName"));

## 14、对称加密工具

通过zookeeper以及mongodb实现DES对称加密工具类的配置，主要提供加密和解密功能。

1. 核心类：

com.zpaas.security.ciper.CiperSVC

com.zpaas.security.ciper.DesCiperSVCImpl

（2）扩展实现方法清单

public String encrypt(String data);

public String decrypt(String data);

（3）Spring配置文件

<bean id=*"desCipherSVC"* class=*"com.zpaas.security.cipher.DesCipherSVCImpl"* init-method=*"init"*>

<property name=*"cc"* ref=*"cc"*/>

<property name=*"confPath"* value=*"/com/zpaas/cipher/desCipherSVC"*/>

<property name=*"secureFileRepo"* ref=*"secureFileSVC"*/>

</bean>

（4）配置信息

create */com/zpaas/cipher/desCipherSVC* {"ciper.cfg.file":"547bd907806af3a1eb2e3b08"}

（6）代码示例

CipherSVC desCipher = (CipherSVC) PaasContextHolder.*getContext*().getBean("desCipherSVC");

String encyptedText = desCipher.encrypt("plainText");

System.***out***.println(desCipher.decrypt(encyptedText));

## 15、其他工具类

（1）CipherUtil（DES加密工具类）

（2）CommandUtil（Unix/Linux命令执行工具类）

（3）CommonUtil（getHostAddr、getClientAddr、hex2Ascii、ascii2Hex）

（4）EqualUtil

（5）Md5Util（MD5加密工具）

（6）ObjectCopyUtil（对象复制）

（7）SerializeUtil（序列化工具）

（8）SignatureUtil（签名、验签工具）

（9）StringUtil（字符串类工具）

（10）ThreadId（执行堆栈重现辅助类）

## 16、其他独立的服务

zPaaS平台还提供其他独立的服务，包括分布式数据库（分库、分表、主备、读写分离、最终事务一致以及保障性弱事务框架）