**上云上平台接口规范**

**版本修订批准记录**

【 历次版本修订记录，版本修订描述重要说明】

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **提交日期** | **版本号** | **版本修订描述** | **修改人** | **备注** |
| 2019-2-25 | 0.9 | 初稿 | 潘锡均 |  |
| 2019-03-07 | 1.0 | 完善 | 杨钊 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

# 接口功能说明

## 概述

### 设计目的

本规范明确上云上平台与相关的供应商的接口协议，为开发人员在进行接口设计、开发、使用过程中提供指导性意见。

### 加密协议

使用rsa 和 aes 相结合的方法加密接口返回数据，上云上平台通过请求头方式将公钥告知各个供应商接口，各个供应商接口返回数据时，需要通过aes算法对返回数据进行加密，同时也需要利用公钥，对aes密钥进行加密处理返回。 相关代码见附录

### 编码格式

统一是用UTF-8

## 接口功能说明

### 使用说明

用户id、操作开始时间、操作结束时间，获取对应用户操作数据信息

### 请求

##### 请求参数

统一请求头

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **说明** | **数据类型** | **可空** |
| Content-Type | 固定数据类型 application/json; charset=UTF-8 | String | N |
| RSAPublicKey | Rsa 公钥 | String | N |

data

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **说明** | **数据类型** | **可空** |
| userId | 系统用户id ，可以多个，以逗号分割 | String | N |
| beginTime | 操作时间——开始yyyyMMddHHmmss | long | N |
| endTime | 操作时间——结束yyyyMMddHHmmss | long | Y |

##### 请求的json文档格式示例

{

"beginTime":20190306000000,

"endTime":20190307000000,

"userId":"13929537026,15994705655"

}

##### 请求的安全问题

建议对固定ip做白名单处理(14.116.209.154, 122.13.1.153,  113.108.192.70)

#### 返回

##### 返回参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **说明** | **数据类型** | **可空** |
| respCode | “0”：成功，其它：失败。 | String | N |
| respMsg | 返回值详细说明 | String | N |
| data | CryptoData实体   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **key** | *String* | *加密的Aes密钥，N* |  |  | | **content** | *String* | *加密的返回数据（*  *加密时请注意将*  *返回结果转为json*  *字符串再做加密）* |  |  | | Object | N |

###### data

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **说明** | **数据类型** | **可空** |
| companyId | 企业id，上云上平台和服务商统一 | String | Y |
| companyName | 企业名称 | String | Y |
| userId | 用户id，企业自己填报的userId，泛指能够确定用户唯一的数据，如用户编码，电话号码等 | String | N |
| connectMethod | 链接方式，  1—物联网卡，  2-手机app，  3-以太网 | long | N |
| mechineId | 设备id，物联网卡中的设备号 | String | 如果链接方式是物联网卡，必填，其他类型，可以为空 |
| source | 源地址，  物联网卡中指卡号  手机app中指手机号  以太网中指源ip |  | N |
| target | 目标，目标ip或域名 |  | N |
| reqDuring | 请求时长，服务端收到请求到返回数据的间隔时间，单位：毫秒，没有就不传，不做要求 |  | Y |
| contLen | 请求的文本的长度，服务端返回的数据的长度，单位，kb，没有就不传，不做要求 |  | Y |
| operateTime | 发生时间，格式yyyyMMddHHmmss | long | N |

##### 返回json文档格式示例

{

"respCode": "0",

"respMsg": "SUCCESS",

"data": {

"key": "bBw8sRM+zDqk7yJOAd0MmeckqoJveVQSlP6vcK0fs54vrDVmZdDpuu7VugW+JvjiP9pDEb/cUBYFjkwgXOFFkQ==",

"content": "mk6xF16WxJAKwVBI5d7ULGmpajGuPEa59SRibZrGYrQ4gDmcir7/BKmWmsFt82VCnWAPItQz3nQR\nnGDCSJ9tMxRBdkpXH76OwD7mKPV++s+alfaA+tuUtYSM/omTFHdc8kjxA6V+sFK5rDDtbV6amT+5\nJwWYtlqtMHCL34L1m6hE+agF5nTDvgaqhJzKkNwy2BecybxIeFo6f1LJXySv46q3ZvUmIBhwceTl\nnWoWisBPAAbjR/rwhJv0oEc4nNKWRILOD83RQSs9YqV3noqd0Aq2UTm0lWIbBFO2tW0Wm9yC1Unf\nb2e6MRdkLOK2FApp6Fk4+YEXK6NwmdrvhZEN7I7+mfM+kWEF45jW1vhzJUI3JqjaMZ2oLacl3shf\nM79LQSjrJZqxHxzlMcCxd3HMgZlIe6Cw3JgEajsRmzs4Ni0nVg+uFgQUsJPufUq0mnWOAsH4mZoP\nadC030Q2gssRv7vSF6PG75WrnGzYdGlpOselJHlBWVfwrGP3jS4K7pp/Jczk/K6p54QRjKbsNQqI\nSCzpC8dBcQxZRB42KlwMiy7OKkek7Sslbc20LlANvaSFAkO481Az4g6KLlzkAtSrmCL3perrYDAH\ne6B0/6YiZ+nuq0GHkeKmDG9mfEDqU/wWZUAMGC+NG6T3uWp5wnT5fezuifKjNqaXRHWtbxjTeApm\nLd8AQG1aTiHlURxwbOV3pnVrwrPiaNEsY332CMSqHoU6lMNzp8LELEseqkS3TjEYFtFOMfCVJ91B\nhycsr7TPhsU/fx1DIc+PLkUBsODCBi8WEaHK6FB4zciemmVgJ4bWl+icq52D226eDZ7A+xFRKWra\nHQFcKpK0hf1Ak8K6K3s4H3buDQmNpS5bS2SozSetPJRixImw6w/9Y3OLHR/cIlAzzUB6372YDbs4\nr5t/ipTAYIhGjWj58GjCFwd+Jtjn1FwGXVpYSADJdrWZS0ZkRRRWjAjB2BVKeQ14CeLhJFjEcyQ7\n3MR+gfMw/1iQ6fhOya/yC1uMvkihg9VlkohHjRliA8qdKkQTaVr1kXEE+/LiBK5NnbL6Z+/FDFpH\nzJCm5WWihkh7XEGCUoTvpHIdxa0bQp9FXTkdVSdd/6fm6J3ukL3PJ3YMO/U/kDxza8euc4EJ1R77\naiSGXWuZ/Xfge7kCrYxeU3Sg/A4VVkbxmqj8V2FjMeYH9vJtBNUPbg2vuP0dSOt0G1yj0ffLHcGu\naCmd1zK75knQBWNseQNhGwDdU9LEGKqrDfUPfGk50XnzxWAwLPF5B6p4jaba03jXt/W8z1Dkao+0\nNJZP9PLaxc469zEE3rEU1jPxQYsSTolKeovrJgPeWAcCsfGj/1zb59RcBl1aWEgAyXa1mUtGZEUU\nVowIwdgVSnkNeAni4SRYxHMkO9zEfoHzMP9YkOn4Tsmv8gtbjL5IoYPVZZKIR40ZYgPKnSpEE2la\n9ZFxBPvy4gSuTZ2y+mfvxQxaR8yQpuVlooZIe1xBglKE76RyHfdxKg6j9SgDxtctDJ3CF3WAf1Y/\nlQ+2mIdQeL0nOCoQhTqUw3OnwsQsSx6qRLdOMRgW0U4x8JUn3UGHJyyvtM+GxT9/HUMhz48uRQGw\n4MIGLxYRocroUHjNyJ6aZWAnhtaX6JyrnYPbbp4NnsD7EVEpatodAVwqkrSF/UCTwrorezgfdu4N\nCY2lLltLZKjNJy4DUKMFnUB2qfSfHP9t3QHNlTmgzbaRZLhPV0Z42TFVlMBgiEaNaPnwaMIXB34m\n2OfUXAZdWlhIAMl2tZlLRmRFFFaMCMHYFUp5DXgJ4uEkWMRzJDvcxH6B8zD/WJDp+E7Jr/ILW4y+\nSKGD1WWSiEeNGWIDyp0qRBNpWvWRcQT78uIErk2dsvpn78UMWkfMkKblZaKGSHtcQYJShO+kch1G\nTimAO9fg+QVKyms7Qinxne6Qvc8ndgw79T+QPHNrx65zgQnVHvtqJIZda5n9d+B7uQKtjF5TdKD8\nDhVWRvGadehqX5rQZzBabMqdvoj13aMx6Njwun9YWNmdIRn4MhNoKZ3XMrvmSdAFY2x5A2EbAN1T\n0sQYqqsN9Q98aTnRefPFYDAs8XkHqniNptrTeNfto1iNmRj6RlT4X4FckOIVzjr3MQTesRTWM/FB\nixJOiUp6i+smA95YBwKx8aP/XNvn1FwGXVpYSADJdrWZS0ZkRRRWjAjB2BVKeQ14CeLhJOQbqI5v\nDpJPHwoKuoM15ZhOya/yC1uMvkihg9VlkohHjRliA8qdKkQTaVr1kXEE+/LiBK5NnbL6Z+/FDFpH\nzJCm5WWihkh7XEGCUoTvpHIdDZeILbBYCtKA0pMHdn6jE53ukL3PJ3YMO/U/kDxza8euc4EJ1R77\naiSGXWuZ/Xfge7kCrYxeU3Sg/A4VVkbxmqj8V2FjMeYH9vJtBNUPbg2vuP0dSOt0G1yj0ffLHcGu\naCmd1zK75knQBWNseQNhGwDdU9LEGKqrDfUPfGk50XnzxWAwLPF5B6p4jaba03jXoQqpOTlzXv2F\nq9OwvudRgpQw6yYkJMVj5fQtTt3FYpcJxzQ78GOda5apPt/TVpO/QJ8AuDg7kbFCqqhYzqeHmZ2O\njTcmL+w+9zLP/m8X6W9nYMpkEsz8POz5CfiCv7q19m0WtNuxYwOX4Impf3wTp+zRt3uRDt1lwBGc\nmYlooPTP09YPnr/Uo/6dvxSpmxjz1VI7jCxThiazCyvKXvcAoyUjB04oVanBFyBHhEeAIkompyYA\nuWSVDVNsZtkrSMeernOBCdUe+2okhl1rmf134Hu5Aq2MXlN0oPwOFVZG8Zp16GpfmtBnMFpsyp2+\niPXdozHo2PC6f1hY2Z0hGfgyE2gpndcyu+ZJ0AVjbHkDYRsA3VPSxBiqqw31D3xpOdF588VgMCzx\neQeqeI2m2tN412v6yOzeePwD/qHF8rMW3kDOOvcxBN6xFNYz8UGLEk6JSnqL6yYD3lgHArHxo/9c\n2+fUXAZdWlhIAMl2tZlLRmRFFFaMCMHYFUp5DXgJ4uEkWMRzJDvcxH6B8zD/WJDp+E7Jr/ILW4y+\nSKGD1WWSiEeNGWIDyp0qRBNpWvWRcQT78uIErk2dsvpn78UMWkfMkKblZaKGSHtcQYJShO+kch25\nNNhYtKc3H80VapNtfCJpgH9WP5UPtpiHUHi9JzgqEIU6lMNzp8LELEseqkS3TjEYFtFOMfCVJ91B\nhycsr7TPhsU/fx1DIc+PLkUBsODCBi8WEaHK6FB4zciemmVgJ4bWl+icq52D226eDZ7A+xFRKWra\nHQFcKpK0hf1Ak8K6K3s4H3buDQmNpS5bS2SozScyBv3bJf/CqOWVN/SqsUVED0LIpuXPP8UUYJyc\n1CoDCCyTqdlohijxZgSM/yPVXKEs6QvHQXEMWUQeNipcDIsuzipHpO0rJW3NtC5QDb2khaPPSCpj\nXTQTm7ad71mgQ/ci96Xq62AwB3ugdP+mImfp7qtBh5HipgxvZnxA6lP8FmVADBgvjRuk97lqecJ0\n+X3s7onyozaml0R1rW8Y03gKmO6MA14NB0VtIcyZZ2Yf6GZfJk9ZkrlTIIKpVSru11jwgvl5nnJt\nhLTQ8wCJwNBejv6Z8z6RYQXjmNbW+HMlQjcmqNoxnagtpyXeyF8zv0tJf65333oG4ZqFfS/O1pO5\nmUh7oLDcmARqOxGbOzg2LSdWD64WBBSwk+59SrSadY4CwfiZmg9p0LTfRDaCyxG/u9IXo8bvlauc\nbNh0aWk6xzOaFCffugC61hQ7tp5QHNPGfAgz3lW6q+Q5AKK4ySeR8kjxA6V+sFK5rDDtbV6amT+5\nJwWYtlqtMHCL34L1m6h5/kw3grZMHsu1ccp5QvkbvC22B4vQp2MZ2fvr2XSCjKq3ZvUmIBhwceTl\nnWoWisBPAAbjR/rwhJv0oEc4nNKWRILOD83RQSs9YqV3noqd0In2Wgs4BYXOAmFLRYE2JrWC1Unf\nb2e6MRdkLOK2FApp6Fk4+YEXK6NwmdrvhZEN7I7+mfM+kWEF45jW1vhzJUI3JqjaMZ2oLacl3shf\nM79LSX+ud996BuGahX0vztaTuZlIe6Cw3JgEajsRmzs4Ni0nVg+uFgQUsJPufUq0mnWOAsH4mZoP\nadC030Q2gssRv7vSF6PG75WrnGzYdGlpOse+iKS0z2Y6i+/vYqa8dR3QxnwIM95VuqvkOQCiuMkn\nkfJI8QOlfrBSuaww7W1empk/uScFmLZarTBwi9+C9Zuoef5MN4K2TB7LtXHKeUL5G7wttgeL0Kdj\nGdn769l0goyqt2b1JiAYcHHk5Z1qForATwAG40f68ISb9KBHOJzSlkSCzg/N0UErPWKld56KndAh\nYMy5k82J0lHAD1ime8QQgtVJ329nujEXZCzithQKaehZOPmBFyujcJna74WRDeyO/pnzPpFhBeOY\n1tb4cyVCNyao2jGdqC2nJd7IXzO/S0l/rnffegbhmoV9L87Wk7mZSHugsNyYBGo7EZs7ODYtJ1YP\nrhYEFLCT7n1KtJp1jgLB+JmaD2nQtN9ENoLLEb+70hejxu+Vq5xs2HRpaTrH5oYP3Nf30uArcsXz\nG6mQCMZ8CDPeVbqr5DkAorjJJ5HySPEDpX6wUrmsMO1tXpqZP7knBZi2Wq0wcIvfgvWbqET5qAXm\ndMO+BqqEnMqQ3DLYF5zJvEh4Wjp/UslfJK/jqrdm9SYgGHBx5OWdahaKwE8ABuNH+vCEm/SgRzic\n0pZEgs4PzdFBKz1ipXeeip3QSpsn0XKJ3h0NlHhae6K2KoLVSd9vZ7oxF2Qs4rYUCmnoWTj5gRcr\no3CZ2u+FkQ3sjv6Z8z6RYQXjmNbW+HMlQjcmqNoxnagtpyXeyF8zv0tBKOslmrEfHOUxwLF3ccyB\nmUh7oLDcmARqOxGbOzg2LSdWD64WBBSwk+59SrSadY4CwfiZmg9p0LTfRDaCyxG/u9IXo8bvlauc\nbNh0aWk6x594miyGpT9EhAo7F5tNRDPGfAgz3lW6q+Q5AKK4ySeR8kjxA6V+sFK5rDDtbV6amT+5\nJwWYtlqtMHCL34L1m6hE+agF5nTDvgaqhJzKkNwy2BecybxIeFo6f1LJXySv46q3ZvUmIBhwceTl\nnWoWisBPAAbjR/rwhJv0oEc4nNKWRILOD83RQSs9YqV3noqd0HLznTpO4om/PdC56k5OBL/MdDqv\n1L2QJFoWtDYQdFR+fmYjJ/WoS9qmq2CSIIOK3749yuz0sSMoEWPiFseGyRuYwi7PqgMLV/gskxZ2\nwmQvpzcfmJX9OTliirm7oakZkhYnXYn+Sytvqg+7H87XR8ZpqWoxrjxGufUkYm2axmK0OIA5nIq+\n/wSplprBbfNlQp1gDyLUM950EZxgwkifbTNumBFW0SgPvIkO7eHh8AmBYX0YkJLI+ib6YigXi9BJ\nDWeYh1Xu/xvknrjaNtQX/09AnwC4ODuRsUKqqFjOp4eZQm09ZQeV6Zix10LLFW9T+XWycyweAnY/\ngIMOmM4u6HP2bRa027FjA5fgial/fBOn7NG3e5EO3WXAEZyZiWig9M/T1g+ev9Sj/p2/FKmbGPN4\ndqlT8TcmoWvlJYJtVAst0hUmFDfCRgxdE/4R3vPIbCanJgC5ZJUNU2xm2StIx56uc4EJ1R77aiSG\nXWuZ/Xfge7kCrYxeU3Sg/A4VVkbxmqj8V2FjMeYH9vJtBNUPbg2vuP0dSOt0G1yj0ffLHcGuaCmd\n1zK75knQBWNseQNhGwDdU9LEGKqrDfUPfGk50XnzxWAwLPF5B6p4jaba03jXRBFsshaAYe9OaSrC\nG8bRkJQw6yYkJMVj5fQtTt3FYpcJxzQ78GOda5apPt/TVpO/QJ8AuDg7kbFCqqhYzqeHmUJtPWUH\nlemYsddCyxVvU/l1snMsHgJ2P4CDDpjOLuhz9m0WtNuxYwOX4Impf3wTp+zRt3uRDt1lwBGcmYlo\noPTP09YPnr/Uo/6dvxSpmxjz0hRxvkvrP+bpqSq1SPVQ/oIgcJdzpT8XzjrzfEHaCUncQYAE0H6X\nuJWWtamOlG9+vj3K7PSxIygRY+IWx4bJG5jCLs+qAwtX+CyTFnbCZC+nNx+Ylf05OWKKubuhqRmS\nFiddif5LK2+qD7sfztdHxmmpajGuPEa59SRibZrGYrQ4gDmcir7/BKmWmsFt82VCnWAPItQz3nQR\nnGDCSJ9tM88tuKxd7zgjz83m71pntAialfaA+tuUtYSM/omTFHdc8kjxA6V+sFK5rDDtbV6amT+5\nJwWYtlqtMHCL34L1m6hE+agF5nTDvgaqhJzKkNwy2BecybxIeFo6f1LJXySv46q3ZvUmIBhwceTl\nnWoWisBPAAbjR/rwhJv0oEc4nNKWRILOD83RQSs9YqV3noqd0Cbjcux9dBRiviY++LmLNpCC1Unf\nb2e6MRdkLOK2FApp6Fk4+YEXK6NwmdrvhZEN7I7+mfM+kWEF45jW1vhzJUI3JqjaMZ2oLacl3shf\nM79LQSjrJZqxHxzlMcCxd3HMgZlIe6Cw3JgEajsRmzs4Ni0nVg+uFgQUsJPufUq0mnWOAsH4mZoP\nadC030Q2gssRv7vSF6PG75WrnGzYdGlpOsfRJ6BAQ6ySGRJD7G7pQhoAJczk/K6p54QRjKbsNQqI\nSCzpC8dBcQxZRB42KlwMiy7OKkek7Sslbc20LlANvaSFAkO481Az4g6KLlzkAtSrmCL3perrYDAH\ne6B0/6YiZ+nuq0GHkeKmDG9mfEDqU/wWZUAMGC+NG6T3uWp5wnT5fezuifKjNqaXRHWtbxjTeAoD\n+NBsSCod7ReVpEuHg2i9Zl8mT1mSuVMggqlVKu7XWPCC+Xmecm2EtNDzAInA0F6O/pnzPpFhBeOY\n1tb4cyVCNyao2jGdqC2nJd7IXzO/S0Eo6yWasR8c5THAsXdxzIGZSHugsNyYBGo7EZs7ODYtJ1YP\nrhYEFLCT7n1KtJp1jgLB+JmaD2nQtN9ENoLLEb+70hejxu+Vq5xs2HRpaTrHLwZD9sGlYGHIpCB7\nYXsvTyXM5PyuqeeEEYym7DUKiEgs6QvHQXEMWUQeNipcDIsuzipHpO0rJW3NtC5QDb2khQJDuPNQ\nM+IOii5c5ALUq5g8gApEyUewM94sxPVIh3A1"

}

}

附录A，

**package** com.cdc.workcloud.manage.util;  
  
**import** com.cdc.common.util.JsonMapper;  
**import** org.apache.commons.codec.binary.Base64;  
**import** org.springframework.stereotype.Component;  
**import** org.springframework.util.Base64Utils;  
  
**import** javax.crypto.Cipher;  
**import** java.security.\*;  
**import** java.security.interfaces.RSAPrivateKey;  
**import** java.security.interfaces.RSAPublicKey;  
**import** java.security.spec.PKCS8EncodedKeySpec;  
**import** java.security.spec.X509EncodedKeySpec;  
**import** java.util.Date;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
  
*/\*\*  
 \* 非对称加密算法RSA算法组件  
 \* 非对称算法一般是用来传送对称加密算法的密钥来使用的，相对于DH算法，RSA算法只需要一方构造密钥，不需要  
 \* 大费周章的构造各自本地的密钥对了。DH算法只能算法非对称算法的底层实现。而RSA算法算法实现起来较为简单  
 \*  
 \** ***@author*** *kongqz  
 \*/*@Component  
**public class** RSACoder {  
 *//非对称密钥算法* **public static final** String ***KEY\_ALGORITHM*** = **"RSA"**;  
  
  
 */\*\*  
 \* 密钥长度，DH算法的默认密钥长度是1024  
 \* 密钥长度必须是64的倍数，在512到65536位之间  
 \*/* **private static final int *KEY\_SIZE*** = 512;  
 *//公钥* **private static final** String ***PUBLIC\_KEY*** = **"RSAPublicKey"**;  
  
 *//私钥* **private static final** String ***PRIVATE\_KEY*** = **"RSAPrivateKey"**;  
  
 */\*\*  
 \* 初始化密钥对  
 \*  
 \** ***@return*** *Map 甲方密钥的Map  
 \*/* **public static** Map<String, Object> initKey() **throws** Exception {  
 *//实例化密钥生成器* KeyPairGenerator keyPairGenerator = KeyPairGenerator.*getInstance*(***KEY\_ALGORITHM***);  
 *//初始化密钥生成器* keyPairGenerator.initialize(***KEY\_SIZE***);  
 *//生成密钥对* KeyPair keyPair = keyPairGenerator.generateKeyPair();  
 *//甲方公钥* RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();  
 *//甲方私钥* RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();  
 *//将密钥存储在map中* Map<String, Object> keyMap = **new** HashMap<String, Object>();  
 keyMap.put(***PUBLIC\_KEY***, publicKey);  
 keyMap.put(***PRIVATE\_KEY***, privateKey);  
 **return** keyMap;  
  
 }  
  
  
 */\*\*  
 \* 私钥加密  
 \*  
 \** ***@param data*** *待加密数据  
 \** ***@param key*** *密钥  
 \** ***@return*** *byte[] 加密数据  
 \*/* **public static byte**[] encryptByPrivateKey(**byte**[] data, String key) **throws** Exception {  
 **byte**[] keyBytes = Base64Utils.*decodeFromString*(key);  
 *//取得私钥* PKCS8EncodedKeySpec pkcs8KeySpec = **new** PKCS8EncodedKeySpec(keyBytes);  
 KeyFactory keyFactory = KeyFactory.*getInstance*(***KEY\_ALGORITHM***);  
 *//生成私钥* PrivateKey privateKey = keyFactory.generatePrivate(pkcs8KeySpec);  
 *//数据加密* Cipher cipher = Cipher.*getInstance*(keyFactory.getAlgorithm());  
 cipher.init(Cipher.***ENCRYPT\_MODE***, privateKey);  
 **return** cipher.doFinal(data);  
 }  
  
 */\*\*  
 \* 公钥加密  
 \*  
 \** ***@param data*** *待加密数据  
 \** ***@param key*** *密钥  
 \** ***@return*** *byte[] 加密数据  
 \*/* **public static byte**[] encryptByPublicKey(**byte**[] data, String key) **throws** Exception {  
  
 **byte**[] keyBytes = Base64Utils.*decodeFromString*(key);  
 *//实例化密钥工厂* KeyFactory keyFactory = KeyFactory.*getInstance*(***KEY\_ALGORITHM***);  
 *//初始化公钥  
 //密钥材料转换* X509EncodedKeySpec x509KeySpec = **new** X509EncodedKeySpec(keyBytes);  
 *//产生公钥* PublicKey pubKey = keyFactory.generatePublic(x509KeySpec);  
  
 *//数据加密* Cipher cipher = Cipher.*getInstance*(keyFactory.getAlgorithm());  
 cipher.init(Cipher.***ENCRYPT\_MODE***, pubKey);  
 **return** cipher.doFinal(data);  
 }  
  
 */\*\*  
 \* 私钥解密  
 \*  
 \** ***@param data*** *待解密数据  
 \** ***@param key*** *密钥  
 \** ***@return*** *byte[] 解密数据  
 \*/* **public static byte**[] decryptByPrivateKey(**byte**[] data, String key) **throws** Exception {  
 **byte**[] keyBytes = Base64Utils.*decodeFromString*(key);  
 *//取得私钥* PKCS8EncodedKeySpec pkcs8KeySpec = **new** PKCS8EncodedKeySpec(keyBytes);  
 KeyFactory keyFactory = KeyFactory.*getInstance*(***KEY\_ALGORITHM***);  
 *//生成私钥* PrivateKey privateKey = keyFactory.generatePrivate(pkcs8KeySpec);  
 *//数据解密* Cipher cipher = Cipher.*getInstance*(keyFactory.getAlgorithm());  
 cipher.init(Cipher.***DECRYPT\_MODE***, privateKey);  
 **return** cipher.doFinal(data);  
 }  
  
 */\*\*  
 \* 公钥解密  
 \*  
 \** ***@param data*** *待解密数据  
 \** ***@param key*** *密钥  
 \** ***@return*** *byte[] 解密数据  
 \*/* **public static byte**[] decryptByPublicKey(**byte**[] data, String key) **throws** Exception {  
 **byte**[] keyBytes = Base64Utils.*decodeFromString*(key);  
 *//实例化密钥工厂* KeyFactory keyFactory = KeyFactory.*getInstance*(***KEY\_ALGORITHM***);  
 *//初始化公钥  
 //密钥材料转换* X509EncodedKeySpec x509KeySpec = **new** X509EncodedKeySpec(keyBytes);  
 *//产生公钥* PublicKey pubKey = keyFactory.generatePublic(x509KeySpec);  
 *//数据解密* Cipher cipher = Cipher.*getInstance*(keyFactory.getAlgorithm());  
 cipher.init(Cipher.***DECRYPT\_MODE***, pubKey);  
 **return** cipher.doFinal(data);  
 }  
  
 */\*\*  
 \* 取得私钥  
 \*  
 \** ***@param keyMap*** *密钥map  
 \** ***@return*** *byte[] 私钥  
 \*/* **public static** String getPrivateKey(Map<String, Object> keyMap) {  
  
 Key key = (Key) keyMap.get(***PRIVATE\_KEY***);  
 **return** Base64Utils.*encodeToString*(key.getEncoded());  
 }  
  
 */\*\*  
 \* 取得公钥  
 \*  
 \** ***@param keyMap*** *密钥map  
 \** ***@return*** *byte[] 公钥  
 \*/* **public static** String getPublicKey(Map<String, Object> keyMap) **throws** Exception {  
 Key key = (Key) keyMap.get(***PUBLIC\_KEY***);  
 **return** Base64Utils.*encodeToString*(key.getEncoded());  
 }  
  
}

**package** com.cdc.workcloud.manage.util;  
**import** java.security.Key;  
**import** java.util.UUID;  
  
**import** javax.crypto.Cipher;  
**import** javax.crypto.spec.SecretKeySpec;  
  
**import** sun.misc.BASE64Decoder;  
**import** sun.misc.BASE64Encoder;  
  
**public class** AESSecurityUtil {  
  
 *// 加密算法  
 /\*\* 指定加密算法为RSA \*/* **private static final** String ***ALGORITHM*** = **"AES"**;  
  
 *// 加密密钥  
 // private static final byte[] keyValue = new byte[] { 'T', 'h', 'e',  
 // 'B','e', 's', 't', 'S', 'e', 'c', 'r', 'e', 't', 'K', 'e', 'y' };  
 // 16位的加密密钥  
// private byte[] keyValue;  
  
 /\*\*  
 \* 用来进行加密的操作  
 \*  
 \** ***@param*** *\** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public static** String encrypt(String keyString, String data)  
 **throws** Exception {  
 Key key = *generateKey*(keyString);  
 Cipher c = Cipher.*getInstance*(***ALGORITHM***);  
 c.init(Cipher.***ENCRYPT\_MODE***, key);  
 **byte**[] encVal = c.doFinal(data.getBytes());  
 String encryptedValue = **new** BASE64Encoder().encode(encVal);  
 **return** encryptedValue;  
 }  
  
 */\*\*  
 \* 用来进行解密的操作  
 \*  
 \** ***@param encryptedData*** *\** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public static** String decrypt(String keyString, String encryptedData) **throws** Exception {  
 Key key = *generateKey*(keyString);  
 Cipher c = Cipher.*getInstance*(***ALGORITHM***);  
 c.init(Cipher.***DECRYPT\_MODE***, key);  
 **byte**[] decordedValue = **new** BASE64Decoder().decodeBuffer(encryptedData);  
 **byte**[] decValue = c.doFinal(decordedValue);  
 String decryptedValue = **new** String(decValue);  
 **return** decryptedValue;  
 }  
  
 **public static** String generateKeyString()  
 {  
 *//必须长度为16* **return** UUID.*randomUUID*().toString().replaceAll(**"-"**, **""**).substring(0, 16);  
 }  
  
 */\*\*  
 \* 根据密钥和算法生成Key  
 \*  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* **private static** Key generateKey(String keyString) **throws** Exception {  
 Key key = **new** SecretKeySpec(keyString.getBytes(), ***ALGORITHM***);  
 **return** key;  
 }  
  
 **public static void** main(String [] args) **throws** Exception  
 {  
 String keyString = *generateKeyString*();  
*// String keyString = "1234567890123456";* System.***out***.println(**"密钥："** + keyString);  
  
 String source = **"恭喜发财!"**;*// 要加密的字符串* System.***out***.println(**"准备用密钥加密的字符串为："** + source);  
  
 String cryptograph = *encrypt*(keyString, source);*// 生成的密文* System.***out***.print(**"用密钥加密后的结果为:"** + cryptograph);  
 System.***out***.println();  
  
 String target = *decrypt*(keyString, cryptograph);*// 解密密文* System.***out***.println(**"用密钥解密后的字符串为："** + target);  
 System.***out***.println();  
 }  
  
}

**package** com.cdc.workcloud.manage.util;  
  
**import** com.cdc.common.util.JsonMapper;  
**import** com.cdc.workcloud.manage.domain.jxw.CryptoData;  
**import** org.apache.commons.codec.binary.Base64;  
  
**import** java.util.Date;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@ClassName*** *CryptoUtil  
 \** ***@Description TODO*** *\** ***@Author*** *lenovo  
 \** ***@Date*** *2019/2/26 9:26  
 \** ***@Version*** *1.0  
 \*/***public class** CryptoUtil {  
 */\*\*  
 \*  
 \** ***@param data*** *返回数据  
 \** ***@param publicKey*** *公钥  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public static** CryptoData encrypt(String data,String publicKey) **throws** Exception  
 {  
 *//1、产生AES密钥* String keyString = AESSecurityUtil.*generateKeyString*();  
  
 *//2、用AES法加密数据* String cryptograph = AESSecurityUtil.*encrypt*(keyString, data);  
  
 *//3、用RSA加密AES密钥* JsonMapper mapper = **new** JsonMapper();  
 **byte**[] finalKey = RSACoder.*encryptByPublicKey*(Base64.*decodeBase64*(keyString),publicKey);  
*// System.out.print("用RSA加密AES密钥为:" + finalKey);  
// System.out.print("加密数据:" + cryptograph);* CryptoData cryptoData = **new** CryptoData();  
 cryptoData.setKey(Base64.*encodeBase64String*(finalKey));  
 cryptoData.setContent(cryptograph);  
  
 *//4、返回数据* **return** cryptoData;  
 }  
  
 */\*\*  
 \*  
 \** ***@param cryptoData*** *加密数据  
 \** ***@param privateKey*** *rsa私钥  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public static** String decrypt(CryptoData cryptoData,String privateKey) **throws** Exception  
 {  
 *//1、解密密钥* **byte**[] decryptKey = RSACoder.*decryptByPrivateKey*(Base64.*decodeBase64*(cryptoData.getKey()),privateKey);  
  
 *//2、解密内容* String decryptData = AESSecurityUtil.*decrypt*(Base64.*encodeBase64String*(decryptKey),cryptoData.getContent() );  
  
 *//3、返回* **return** decryptData;  
  
 }  
  
 **public static void** main(String[] args)**throws** Exception {  
 Map<String, Object> keyMap = RSACoder.*initKey*();  
 *//公钥* String publicKey = RSACoder.*getPublicKey*(keyMap);  
 *//私钥* String privateKey = RSACoder.*getPrivateKey*(keyMap);  
 *//请求数据* Map<String, Object> requestMap = **new** HashMap<>();  
  
 requestMap.put(**"userId"**, **"1,2,3,4"**);  
 requestMap.put(**"beginTime"**, **new** Date());  
 requestMap.put(**"endTime"**, **new** Date());  
 System.***out***.println(**"元数据："**+requestMap);  
 JsonMapper mapper = **new** JsonMapper();  
  
 CryptoData cryptoData= CryptoUtil.*encrypt*(mapper.toJson(requestMap),publicKey);  
  
 System.***out***.println(**"加密后数据:"**+cryptoData);  
  
 System.***out***.println(**"---解密--"**);  
  
 String result = CryptoUtil.*decrypt*(cryptoData,privateKey);  
  
 System.***out***.println(**"解密后数据:"**+result);  
  
  
  
  
  
  
 }  
}

**package** com.cdc.workcloud.manage.domain.jxw;  
  
**public class** CryptoData **implements** java.io.Serializable{  
  
 **private static final long *serialVersionUID*** = -4774469372648172844L;  
  
 **private** String **key**;  
  
 **private** String **content**;  
  
 **public** String getKey() {  
 **return key**;  
 }  
  
 **public void** setKey(String key) {  
 **this**.**key** = key;  
 }  
  
 **public** String getContent() {  
 **return content**;  
 }  
  
 **public void** setContent(String content) {  
 **this**.**content** = content;  
 }  
  
 **public** String toString() {  
 **return "CryptoData [key="** + **key** + **", content="** + **content** + **"]"**;  
 }  
  
}

测试 :接口

@RequestMapping(**"testAESandRsa"**)  
@ResponseBody  
**public** Response4App testAESandRsa(HttpServletRequest request)**throws** Exception {  
 Response4App app = **new** Response4App();  
 String RSAPublicKey = request.getHeader(**"RSAPublicKey"**);  
 System.***out***.println(RSAPublicKey);  
 *//返回参数* Map<String, Object> requestMap = **new** HashMap<>();  
  
 requestMap.put(**"userId"**, **"1,2,3,4"**);  
 requestMap.put(**"beginTime"**, **new** Date());  
 requestMap.put(**"endTime"**, **new** Date());  
 System.***out***.println(**"元数据："**+requestMap);  
 JsonMapper mapper = **new** JsonMapper();  
  
 CryptoData cryptoData= CryptoUtil.*encrypt*(mapper.toJson(requestMap),RSAPublicKey);  
 app.setData(cryptoData);  
 app.setStatus(1);  
  
 **return** app;  
}

测试：**package** com.cdc.workcloud.manage.web;  
  
**import** com.cdc.common.util.JsonMapper;  
**import** com.cdc.common.util.Response4App;  
**import** com.cdc.workcloud.manage.domain.jxw.CryptoData;  
**import** com.cdc.workcloud.manage.util.CryptoUtil;  
**import** com.cdc.workcloud.manage.util.RSACoder;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.test.context.junit4.SpringRunner;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.springframework.test.web.servlet.MvcResult;  
**import** org.springframework.web.context.WebApplicationContext;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**import static** org.junit.Assert.\*;  
**import static** org.springframework.restdocs.mockmvc.RestDocumentationRequestBuilders.*get*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
@RunWith(SpringRunner.**class**)  
@SpringBootTest  
**public class** CommonControllerTest **extends** BaseControllerTest{  
  
 @Test  
 **public void** testAESandRsa() **throws** Exception{  
 RSACoder rsaCoder = **new** RSACoder();  
 Map<String,Object> map = **new** HashMap<String,Object>();  
 *//生成密钥对* Map<String, Object> keyMap = RSACoder.*initKey*();  
 *//公钥* String publicKey = RSACoder.*getPublicKey*(keyMap);  
  
 *//私钥* String privateKey = RSACoder.*getPrivateKey*(keyMap);  
 System.***out***.println(**"公钥：/n"** + publicKey);  
  
 System.***out***.println(**"私钥：/n"** + privateKey);  
  
 MvcResult result = **mockMvc**.perform(*get*(**"/testAESandRsa"**).header(**"RSAPublicKey"**,publicKey).contentType(MediaType.***APPLICATION\_JSON***))  
 .andExpect(*status*().isOk())*// 模拟向testRest发送get请求  
 // 预期返回值的媒体类型text/plain;charset=UTF-8* .andReturn();*// 返回执行请求的结果* String resultJson= result.getResponse().getContentAsString();  
 JsonMapper mapper = **new** JsonMapper();  
 Response4App app = mapper.fromJson(resultJson,Response4App.**class**);  
 System.***out***.println(app.getData());  
 ObjectMapper objectMapper = **new** ObjectMapper();  
 CryptoData data = objectMapper.convertValue(app.getData(), CryptoData.**class**);  
 System.***out***.println(**"结果:"**+data);  
 System.***out***.println(CryptoUtil.*decrypt*(data,privateKey));  
  
、}  
  
  
}

*/\*\*  
 \* Copyright (c) 2005-2012 springside.org.cn  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \*/***package** com.cdc.common.util;  
  
**import** com.fasterxml.jackson.annotation.JsonInclude.Include;  
**import** com.fasterxml.jackson.core.JsonProcessingException;  
**import** com.fasterxml.jackson.databind.DeserializationFeature;  
**import** com.fasterxml.jackson.databind.JavaType;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** com.fasterxml.jackson.databind.SerializationFeature;  
**import** com.fasterxml.jackson.databind.util.JSONPObject;  
**import** org.apache.commons.lang3.StringUtils;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
**import** java.io.IOException;  
  
*/\*\*  
 \* 简单封装Jackson，实现JSON String<->Java Object的Mapper.  
 \*   
 \* 封装不同的输出风格, 使用不同的builder函数创建实例.  
 \*   
 \** ***@author*** *calvin  
 \*/***public class** JsonMapper {  
  
 **private static** Logger *logger* = LoggerFactory.*getLogger*(JsonMapper.**class**);  
  
 **private** ObjectMapper **mapper**;  
  
 **public** JsonMapper() {  
 **this**(**null**);  
 }  
  
 **public** JsonMapper(Include include) {  
 **mapper** = **new** ObjectMapper();  
 *//设置输出时包含属性的风格* **if** (include != **null**) {  
 **mapper**.setSerializationInclusion(include);  
 }  
 *//设置输入时忽略在JSON字符串中存在但Java对象实际没有的属性* **mapper**.disable(DeserializationFeature.***FAIL\_ON\_UNKNOWN\_PROPERTIES***);  
 }  
  
 */\*\*  
 \* 创建只输出非Null且非Empty(如List.isEmpty)的属性到Json字符串的Mapper,建议在外部接口中使用.  
 \*/* **public static** JsonMapper nonEmptyMapper() {  
 **return new** JsonMapper(Include.***NON\_EMPTY***);  
 }  
  
 */\*\*  
 \* 创建只输出初始值被改变的属性到Json字符串的Mapper, 最节约的存储方式，建议在内部接口中使用。  
 \*/* **public static** JsonMapper nonDefaultMapper() {  
 **return new** JsonMapper(Include.***NON\_DEFAULT***);  
 }  
  
 */\*\*  
 \* Object可以是POJO，也可以是Collection或数组。  
 \* 如果对象为Null, 返回"null".  
 \* 如果集合为空集合, 返回"[]".  
 \*/* **public** String toJson(Object object) {  
  
 **try** {  
 **return mapper**.writeValueAsString(object);  
 } **catch** (IOException e) {  
 *logger*.warn(**"write to json string error:"** + object, e);  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* 反序列化POJO或简单Collection如List<String>.  
 \*   
 \* 如果JSON字符串为Null或"null"字符串, 返回Null.  
 \* 如果JSON字符串为"[]", 返回空集合.  
 \*   
 \* 如需反序列化复杂Collection如List<MyBean>, 请使用fromJson(String,JavaType)  
 \** ***@see*** *#fromJson(String, com.fasterxml.jackson.databind.JavaType)  
 \*/* **public** <T> T fromJson(String jsonString, Class<T> clazz) {  
 **if** (StringUtils.*isEmpty*(jsonString)) {  
 **return null**;  
 }  
  
 **try** {  
 **return mapper**.readValue(jsonString, clazz);  
 } **catch** (IOException e) {  
 *logger*.warn(**"parse json string error:"** + jsonString, e);  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* 反序列化复杂Collection如List<Bean>, 先使用函數createCollectionType构造类型,然后调用本函数.  
 \** ***@see*** *#createCollectionType(Class, Class...)  
 \*/* **public** <T> T fromJson(String jsonString, JavaType javaType) {  
 **if** (StringUtils.*isEmpty*(jsonString)) {  
 **return null**;  
 }  
  
 **try** {  
 **return** (T) **mapper**.readValue(jsonString, javaType);  
 } **catch** (IOException e) {  
 *logger*.warn(**"parse json string error:"** + jsonString, e);  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* 構造泛型的Collection Type如:  
 \* ArrayList<MyBean>, 则调用constructCollectionType(ArrayList.class,MyBean.class)  
 \* HashMap<String,MyBean>, 则调用(HashMap.class,String.class, MyBean.class)  
 \*/* **public** JavaType createCollectionType(Class<?> collectionClass, Class<?>... elementClasses) {  
 **return mapper**.getTypeFactory().constructParametricType(collectionClass, elementClasses);  
 }  
  
 */\*\*  
 \* 當JSON裡只含有Bean的部分屬性時，更新一個已存在Bean，只覆蓋該部分的屬性.  
 \*/* **public** <T> T update(String jsonString, T object) {  
 **try** {  
 **return** (T) **mapper**.readerForUpdating(object).readValue(jsonString);  
 } **catch** (JsonProcessingException e) {  
 *logger*.warn(**"update json string:"** + jsonString + **" to object:"** + object + **" error."**, e);  
 } **catch** (IOException e) {  
 *logger*.warn(**"update json string:"** + jsonString + **" to object:"** + object + **" error."**, e);  
 }  
 **return null**;  
 }  
  
 */\*\*  
 \* 輸出JSONP格式數據.  
 \*/* **public** String toJsonP(String functionName, Object object) {  
 **return** toJson(**new** JSONPObject(functionName, object));  
 }  
  
 */\*\*  
 \* 設定是否使用Enum的toString函數來讀寫Enum,  
 \* 為False時時使用Enum的name()函數來讀寫Enum, 默認為False.  
 \* 注意本函數一定要在Mapper創建後, 所有的讀寫動作之前調用.  
 \*/* **public void** enableEnumUseToString() {  
 **mapper**.enable(SerializationFeature.***WRITE\_ENUMS\_USING\_TO\_STRING***);  
 **mapper**.enable(DeserializationFeature.***READ\_ENUMS\_USING\_TO\_STRING***);  
 }  
  
 */\*\*  
 \* 取出Mapper做进一步的设置或使用其他序列化API.  
 \*/* **public** ObjectMapper getMapper() {  
 **return mapper**;  
 }  
}