深圳四方精创资讯股份有限公司

Java EE开发平台——登录模块

使用手册



|  |  |  |  |
| --- | --- | --- | --- |
| **文档名称** | Java EE开发平台——登录模块使用手册 | **文档编号** |  |
| **文档类别** | 使用手册 | **版本信息** | V1.0 |
| **内部密级** | 公开 | **外部密级** | 自用 |
| **创 建 人** | 黄钟 | **创建日期** | 2016-6-13 |
| **审 批 人** |  | **审批日期** |  |

修改历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 日期 | \*状态 | 修订人 | 摘要 |
| V1.0 | 2016-6-13 | C | 黄钟 | 创建文档 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

\*状态：C – 创建 A – 增加 M – 修改 D – 删除

# 目录

[目录 I](#_Toc28865)

[第1章 登录流程说明 1](#_Toc6200)

[§1.1 登录校验流程图 1](#_Toc29743)

[§1.2 前端加密说明 2](#_Toc24168)

[§1.3 登录处理器链说明 2](#_Toc1493)

[§1.3.1 引入登录处理器链的原因 2](#_Toc31734)

[§1.3.2 引入登录处理器链的好处 3](#_Toc14410)

[§1.3.3 调用API 3](#_Toc15190)

[§1.4 登录监听器链说明 7](#_Toc3461)

[§1.4.1 调用API 8](#_Toc15026)

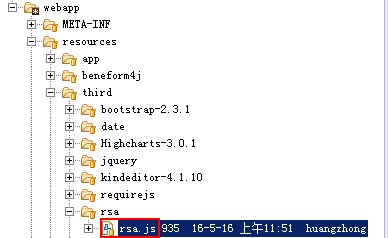
# 登录流程说明

## 登录校验流程图



## 前端加密说明

采用RSA加密方式对登录敏感信息进行加密，需要引用的JS路径见下图：



在登录前，前端通过AJAX（或其它方式）从后台获取公钥信息，前端在拿到公钥后对需要加密的信息进行加密提交，服务端通过RSA解密，服务端RSA解密的JAVA实现JAR包路径(beneform4j-webapp项目下):



其中rsa-1.0.jar为平台组提供，bouncycastle-1.0.jar为第三方必须包

## 登录处理器链说明

### 引入登录处理器链的原因

通常我们做登录检验都是硬编码在代码中，比如需要做IP登录限制、重复登录控制等，这样的坏处是扩展性差，比如后续应用需要增加一个校验项，则需要改动已有代码，作为平台的实现，需要考虑到不同应用系统会有不同的登录校验需求，比如并不是所有应用都需要对用户的登录IP进行控制。

### 引入登录处理器链的好处

综合来说，用登录处理器链来做登录校验有以下好处：

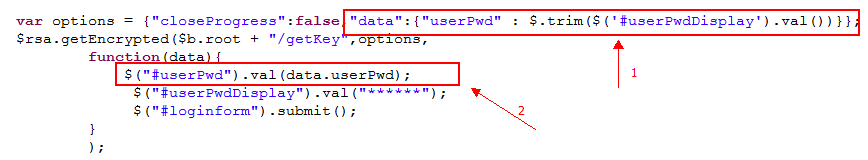
* 灵活性：校验方式可配置，可根据需要启用/禁用某些校验
* 代码风格：校验实现为Spring容器类，配置在XML文件中，代码风格简洁
* 扩展性：可根据需要配置更多的校验类而不需要改动原有代码

### 调用API

这里拿平台DEMO项目作为说明样例，实现一个完整的加密登录流程如下：

1. 加密登录信息

见beneform4j-webapp4easyui项目下的view/system/index.jsp文件:



箭头1标识的内容为待加密数据，JSON对象数据格式

箭头2标识的内容为获取加密返回数据

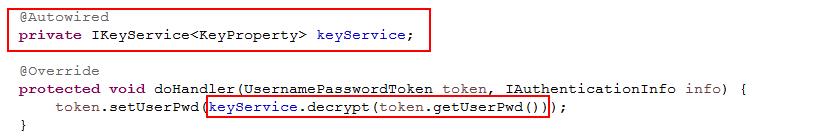
Function(data){}为加密成功后的回调函数，这里需要说明的是加密返回数据的格式和传入的待加密数据的格式是完全一致的，区别在于KEY值对应的数据变成了加密后的数据，比如：

传入数据：{“userPwd”:”111111”,”userId”:”dev”}

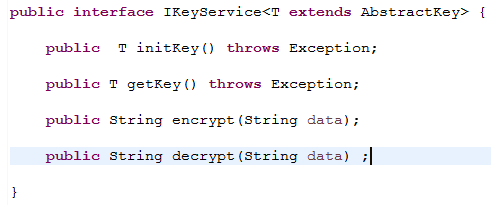
加密返回数据：{“userPwd”:”加密后的密码”,”userId”:”加密后的用户ID”}

1. 后台解密

后台提供加解密工具类API进行解密操作，代码片段如下：



IKeyService<T>是平台提供的加解密接口，该接口提供四个方法，源码如下：

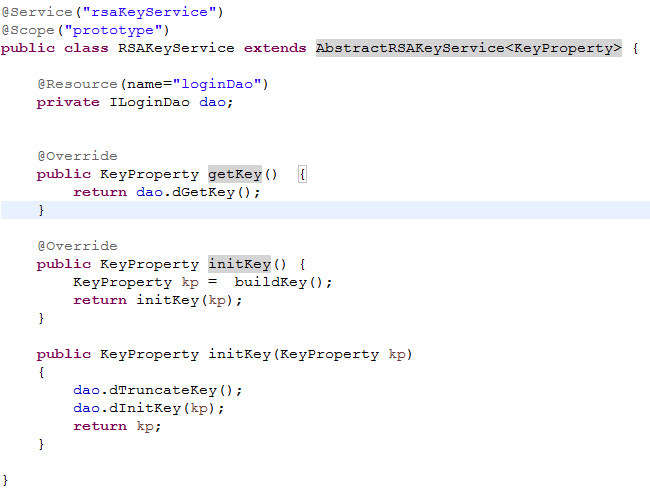


接口的泛型参数AbstractKey是一个抽象类，表示加解密所使用的KEY对象

除了这个接口外，平台还提供一个基于RSA的抽象实现类AbstractRSAKeyService，该类实现了IKeyService接口，代码如下：



那么应用如果要调用平台的RSA接口，并且要实现自己的加解密策略，只需要继承AbstractRSAKeyService类,并重载具体的方法即可，代码如下：



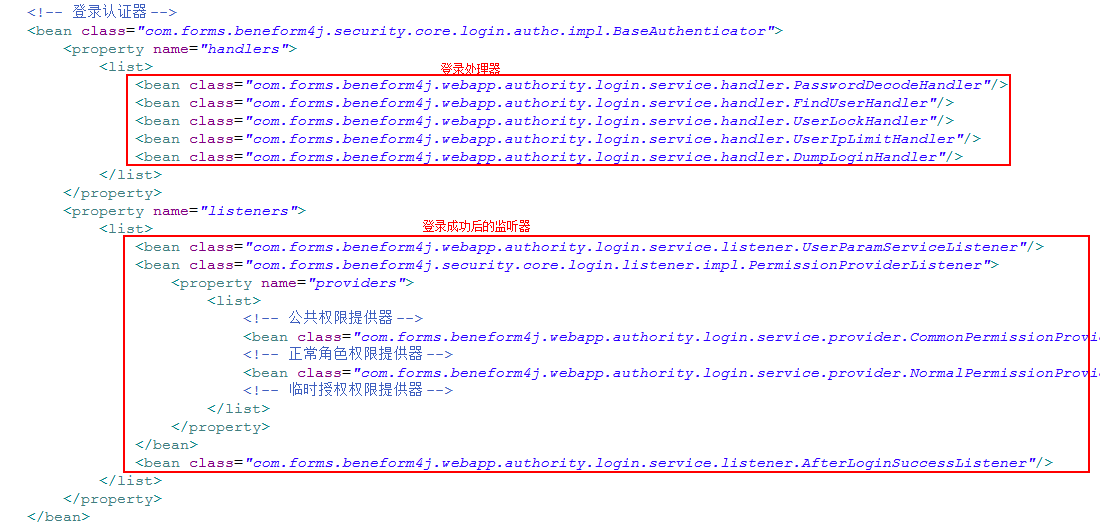
getKey()：应用自己实现的获取具体公钥和私钥的方法

initKey()：应用自己实现的初始化公钥私钥的方法

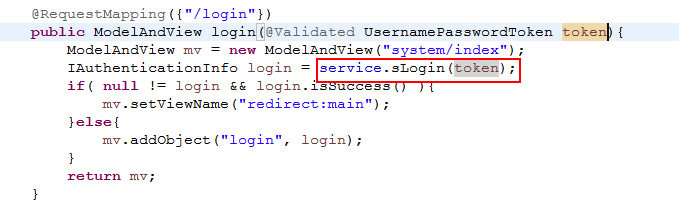
KeyProperty类：继承了AbstractKey类的应用自己实现的KEY对象

1. 调用登录处理器链处理

Spring配置如下：

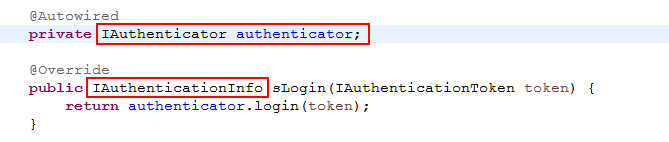


控制层调用代码：



控制层方法参数类UsernamePassowrdToken由平台安全包提供，该类实现了IAuthenticationToken接口，提供用户名和密码两个属性，如果应用有更多的属性，比如验证码等，需要应用自行实现IAuthenticationToken接口

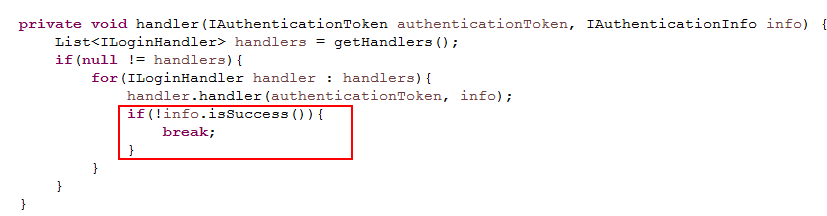
服务层调用代码：



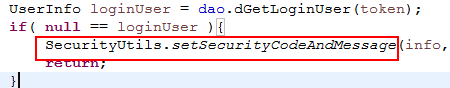
IAuthenticator认证器接口的login方法返回一个IAuthenticationInfo接口对象，控制层通过判断该对象的isSuccess()是否为true来判断登录是否成功。在调用IAuthenticator 的login方法时，平台后台会做这样的操作：

1. 获取Spring配置文件中配置的处理器类对象
2. 依次调用处理器类的handler方法执行处理

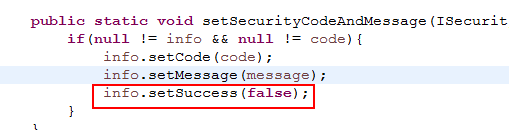
代码如下：



从代码中可以看到,如果某个处理器在handler方法中将info对象（该info对象为所有处理器共享）的success属性置为false,则表示校验没有通过，跳出处理，比如平台的FindUserHandler这个登录时查找用户信息的处理器，代码如下：



红色标识的方法会将info的success置为false:



1. 调用登录监听器处理器链处理

## 登录监听器链说明

登录监听器实现原理和登录监听器基本类似，同样也是把登录成功后的系列动作作为可配置项，而不是硬编码在程序中，这里不再复述

### 调用API

登录监听器无需手动调用，会在登录监听器成功执行完毕后自动调用Spring配置文件中配置的监听器列表