# 安全性设计

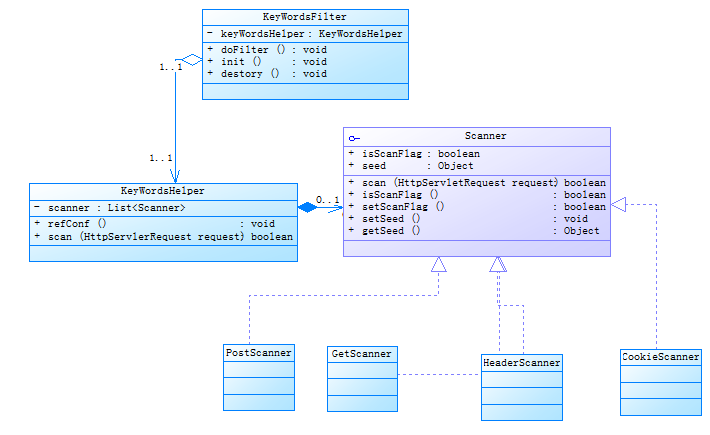
## 内容扫描

### 实现描述

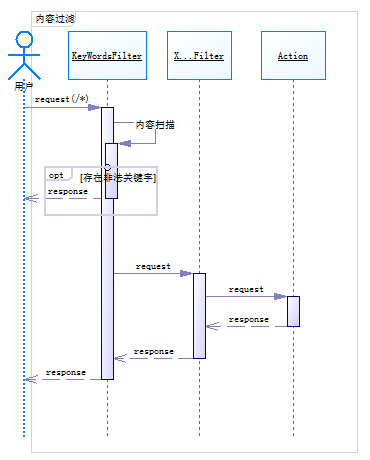
基于B/S的WEB应用都受着XSS,SQL注入等攻击的风险，严格有效的把控输入来源是避免这些攻击发现的风险。内容过滤主要是对所有输入信息来源(Http Header,Post Data(Form),Get Data(QueryString),Cookie Data)进行关键字扫描（通过正则表达式），逐一检查数据的合法性，当发现数据有不合法的关键字内容时进行拦截处理，记录日志并响应至一个配置的指定页面。

* 1. 支持灵活的过滤信息配置
  2. 支持动态的拦截项的启停（不停机或重启服务）
  3. 支持强大的正则表达式

### 结构类图



### 处理时序

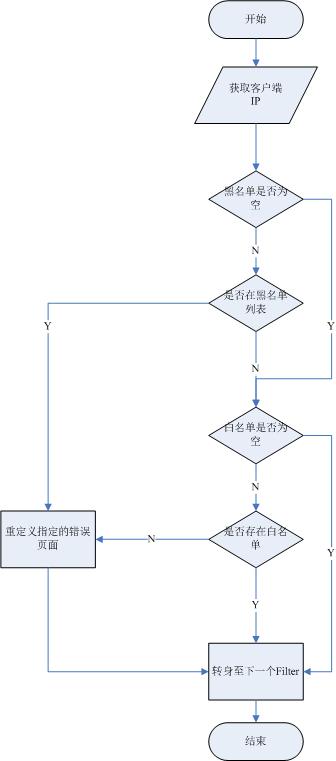
****

## IP拦截

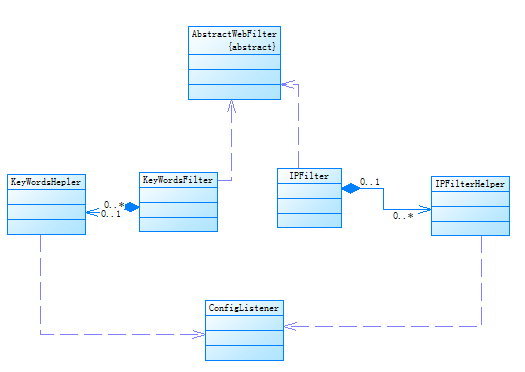
应用安全中对客户端的请求处理严格的检查是保证安全的一个重要环节，除身份认证、鉴权以外IP过滤拦截同样是重要的一环节，它起着进一步加固系统安全的效果。

该功能点实现了黑白名单检查的效果，先过滤黑名单，再过滤白名单，具体流程图如下：

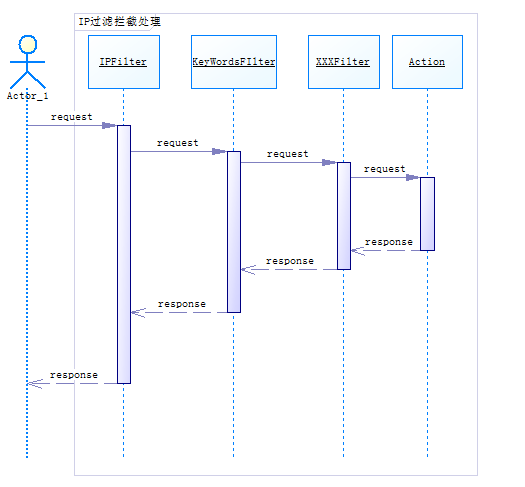
### 流程图



### 结构类图



### 处理时序

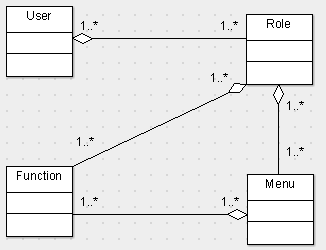


## RBAC权限设计

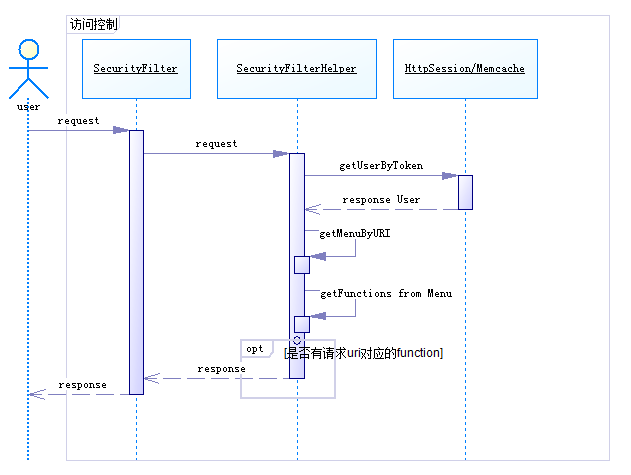
### 实现描述

系统中所有的功能及服务都统一称为功能(Function)；基于WEB的服务访问地址称为资源，在这里又称为菜单(Menu)，对系统访问控制是基于菜单和功能的，该两项可称为权限。配置于角色(Role)上，角色拥有多个菜单或功能权限。一个用户(User)同时可以代表多种角色，角色间的权限（菜单和功能）是一个并集组合。

### 模型结构类图



### 处理时序



## 安全算法

不使用自己开发的加密算法，而使用没有专利的、安全的、公开的标准加密算法。

说明：

1、当前的专利算法包括但不限于：IDEA。已过专利保护期的加密算法已不再受专利保护；

2、不推荐的加密算法包括：DES、MD5、SHA1、HMAC-MD5、HMAC-SHA1；

3、对称加密算法建议使用：AES；

4、密钥交换算法建议使用：DH；

5、数字签名算法建议使用：SHA1withDSA；

6、非对称算法建议使用：ECC、RSA；

7、HASH（哈希）算法建议使用：SHA；

8、HMAC（基于哈希的消息验证码）算法建议使用：HMAC-SHA；

9、建议的各算法建议采用如下的加密强度：

加密算法 最小强度 建议的默认强度 要求更高强度时建议的默认强度

AES 128 128 192、256

DH 1024 1024 2048

DSA 1024 1024 1024

ECC 192 192 216

ECDSA 192 192 216

RSA 1024 1024 2048

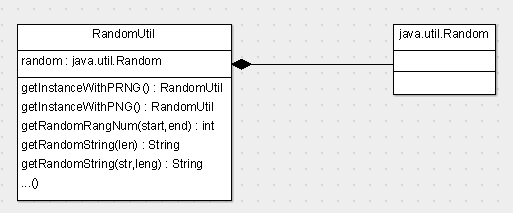
SHA 224 256 384、512

HMAC HMAC-SHA224 HMAC-SHA256 HMAC-SHA384、HMAC-SHA512

### 随机数的考虑

**描述：**通过装适配器的应用，封装了Random的PRNG和PNG方式，同时增强了随机数生成的灵活性和统一性。

**类图：**



**例子：**

//以PRNG方式生成5~15范围里的随机数

**int** start = 5;

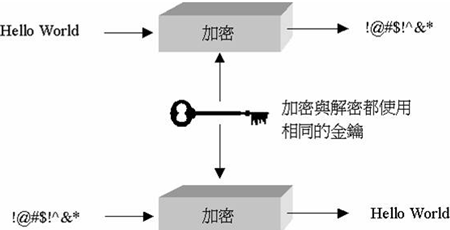
**int** end = 15;

RandomUtil randomUtil = RandomUtil.*getInstanceWithPRNG*();

**int** result = randomUtil.getRandomRangNum(start, end);

## 加密解密

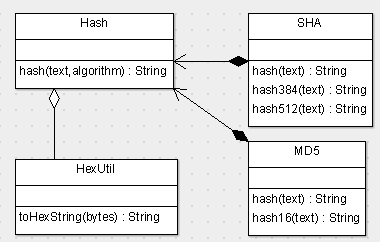
对网络中传输的敏感数据需进行加密后再传输，避免明文传输带来的不安全性问题。



### Hash加密

**描述：推荐使用SHA为hash算法，默认实现sha-256**

**类图：**

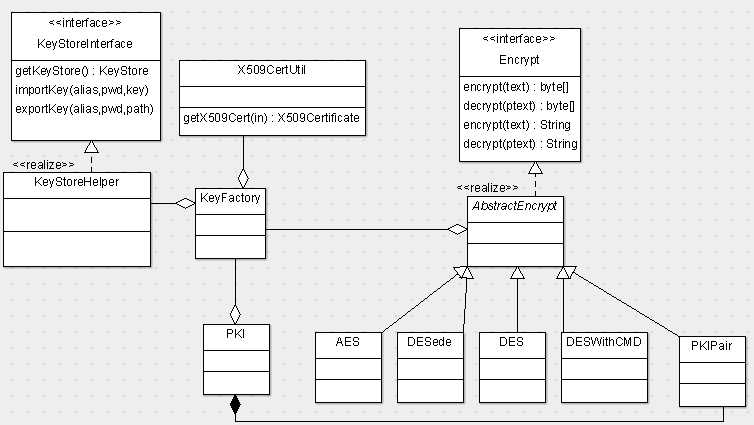
****

**例子：**

String text = "admin123";

String result = SHA.*hash*(text);

### 对称和非对称加解密

****

1. **对称加密算法**

**描述：推荐使用AES(128位)对称加密算法，同时实现有DES、DESWithCMD和DESede**

**例子：**

//原文

String text = "Hi,leo，How are you?";

Encrypt encrypt = AES.*getInstance*();

//加密

String mtext = encrypt.encrypt(text);

System.*out*.println(mtext);

//解密

String ptext = encrypt.decrypt(mtext);

1. **非对称加密算法**

**描述：推荐使用RSA-1024进行安全加密。使用JCEKS密钥库进行密钥对存储管理。**

**例子：**

String mText = "是1234567a中国asdfdsafdsafdsab";

String alias = "testNmae";

String keyPwd = "testPwd";

PKI pki =**new** PKI();

pki.init(alias, keyPwd);

//私钥加密，公钥解密

String m = pki.getPrivatePKI().encrypt(mText);

System.*out*.println("密文：" + m);

String mw = pki.getPublicPKI().decrypt(m);

System.*out*.println("解密得明文：" + mw);

//公钥加密，私钥解密

String m = pki.getPublicPKI().encrypt(mText);

System.*out*.println("密文：" + m);

String mw = pki.getPrivatePKI().decrypt(m);

System.*out*.println("解密得明文：" + mw);

## **数字签名**

### 推荐使用DSA+SHA1算法签名

签名数据以base64字符串方式编码再进行发送

String signStr = pki.sign(mText);

System.*out*.println("原文:"+mText+" 签名:"+signStr);

**boolean** flag = pki.verify(mText,signStr);

### 几种服务模型交互的签名过程

**前提：**

* 服务端生成公钥对A1,B1，公钥A1给客户端，私钥B1服务端保存
* 客户端生成公钥对A2,B2，公钥A2给服务端，私钥B2客户端保存

1. one-way模型：

**A.请求流程**

1).客户端用私钥B2签名，并将原文和数字签名发送给服务端

2).服务端用公钥A2验证签名，通过验证签名，可进行业务交易，否 则结束处理,并记录异常日志

**B.响应流程**

**无**

1. request-response模型：

**A.请求流程**

1).客户端用私钥B2签名，并将原文和数字签名发送给服务端

2).服务端用公钥A2验证签名，通过验证签名，可进行业务交易，否 则结束处理

**B.响应流程**

3).对有响应的请求，服务端用私钥B1签名，并将原文和签名响应给 客户端

4).客户端用公钥A1验证签名，通过验证签名，可进行数据处理，否 则前台页面提示业务操作失败之类的错误，并记录异常日志

1. request-AsynResponse模型：

**A.请求流程**

1).客户端用私钥B2签名，并将原文和数字签名发送给服务端

2).服务端用公钥A2验证签名，通过验证签名，可进行业务交易，否 则结束处理

**B.响应流程**

3).对有响应的请求，服务端用私钥B1签名，并将原文和签名响应给 客户端，**这里的响应通常是调用一个额外的回调服务地址**

4).客户端用公钥A1验证签名，通过验证签名，可进行数据处理，否 则前台页面提示业务操作失败之类的错误，并记录异常日志