独立应用程序中的 getSubject () 调用可能基于特定于应用程序的位置中的用户数据返回主题,  
 并且在服务器环境 (例如 web 应用程序) 中, 它根据与当前线程或传入请求关联的用户数据获取主题

If deploying inside a web application, by default the Session will be HttpSession based. But, in a non-web environment, like this simple tutorial application, Shiro will automatically use its Enterprise Session Management by default

# [Apache Shiro Authorization](http://shiro.apache.org/authorization.html#apache-shiro-authorization)

通常用户允许或不允许做的事情是根据分配给他们的角色或权限决定的。，Subject API 让你可以很容易的执行角色和权限检查

Authorization

授权 (也称为访问控制) ，控制谁有权访问应用程序中的内容。

Authorization有三个核心元素, 我们在 Shiiro 中引用了很多: 权限、角色和用户。

权限表示安全策略中最原子的元素。它们从根本上说是关于行为的陈述, 明确地表示了在应用程序中可以做什么

大部分资源允许CRUD操作。但对特定资源类型有意义的任何操作都可以

基本思想是, 权限语句至少基于资源和操作。

权限语句不表示谁可以执行所表示的行为。它们只是定义了在应用程序中可以执行什么操作的语句。但权限可以分配给 "角色", 并且该角色可以与一个或多个用户对象相关联。

关于如何向用户授予权限, 有许多不同的变体

## [Authorizing Subjects](http://shiro.apache.org/authorization.html#authorizing-subjects)

在 Shiiro 中执行授权可以通过3种方式完成:

以编程方式-您可以在 java 代码中使用类似于 if and else blocks执行授权检查。

JDK annotations：您可以将授权批注附加到 Java 方法

JSP/GSP TagLibs：您可以根据角色和权限控制 JSP 或 GSP 页面输出

### [Roles](http://shiro.apache.org/authorization.html#roles)

有两种类型的角色, Shiiro 支持这两个概念:

**Implicit Roles**

Subject currentUser = SecurityUtils.getSubject();

if (currentUser.hasRole("administrator")) {

//show the admin button

} else {

//don't show the button? Grey it out?

}

**Explicit Roles**

Shiiro 团队提倡使用权限和显式角色, 而不是旧的隐式方法。您将对应用程序的安全体验有更大的控制力。

if ( subject.isPermitted(“user:create”) ) {

// 显示‘Create User’按钮

} else {

// 按钮置灰?

}

角色检查有个很大的缺陷：你无法在运行时增删角色。角色名字在这里是硬编码，所以，如果你修改了角色名字或配置，你的代码就会乱套！如果你需要在运行时改变角色含义，或想要增删角色，你必须另辟蹊径。

#### Role-Based Authorization

Subject上关于role的方法很少

|  |
| --- |
| [hasRole(String roleName)](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/subject/Subject.html#hasRolejava.lang.String-) |
| [hasRoles(List<String> roleNames)](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/subject/Subject.html#hasRoles-java.util.List-) |
| [hasAllRoles(Collection<String> roleNames)](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/subject/Subject.html#hasAllRoles-java.util.Collection-) |
| 检查布尔值以查看主题是否具有角色的替代方法 |
| CheckRole |

[AuthorizationException](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/AuthorizationException.html) will be thrown

#### Permission-Based Authorization

与类似的基于角色的授权代码相比, 代码受到影响的频率要低得多。

检查权限的主要方法有两种--基于对象的权限实例或表示权限的字符串

Object-based Permissions are useful if:

您需要编译时类型安全

您需要显式控制权限解析逻辑的执行方式，based on the Permission interface’s [implies](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/Permission.html#implies-org.apache.shiro.authz.Permission-) method

###### String-based permission checks

Subject currentUser = SecurityUtils.getSubject();

if (currentUser.isPermitted("printer:print:laserjet4400n")) {

//show the Print button

} else {

//don't show the button? Grey it out?

}

printer (resource type), print (action), and laserjet4400n (instance id)

由 Shiiro 的默认 org.apache.shiro.authz.permission.WildcardPermission 实现定义的特殊的冒号分隔格式,

基于字符串的权限是有益的, 因为您不会被迫实现接口, 简单的字符串通常很容易阅读。

对应的check权限检查

|  |  |
| --- | --- |
| [checkPermission(Permission p)](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/subject/Subject.html#checkPermission-org.apache.shiro.authz.Permission-) | Returns quietly if the Subject is permitted to perform an action or access a resource summarized by the specified Permission instance, or throws an AuthorizationException if not. |

### Annotation-based Authorization

#### RequiresAuthentication annotation

批注要求current Subject在其当前会话期间已对其进行身份验证, 以便访问或调用，带注释的类/实例/方法。

@RequiresAuthentication

public void updateAccount(Account userAccount) {

//this method will only be invoked by a

//Subject that is guaranteed authenticated

...

}

This is mostly equivalent to the following Subject-based logic:

public void updateAccount(Account userAccount) {

if (!SecurityUtils.getSubject().isAuthenticated()) {

throw new AuthorizationException(...);

}

//Subject is guaranteed authenticated here

...

}

#### RequiresGuest annotation

#### The RequiresGuest annotation requires the current Subject to be a “guest”, that is, they are not authenticated or remembered from a previous session for the annotated class/instance/method to be accessed or invoked.

This is mostly equivalent to the following Subject-based logic:

public void signUp(User newUser) {

Subject currentUser = SecurityUtils.getSubject();

PrincipalCollection principals = currentUser.getPrincipals();

if (principals != null && !principals.isEmpty()) {

//known identity - not a guest:

throw new AuthorizationException(...);

}

//Subject is guaranteed to be a 'guest' here

...

}

#### RequiresPermissions annotation

要求允许当前主题具有一个或多个权限才能执行批注的方法

@RequiresPermissions("account:create")

public void createAccount(Account account) {

//this method will only be invoked by a Subject

//that is permitted to create an account

...

}

This is mostly equivalent to the following Subject-based logic:

public void createAccount(Account account) {

Subject currentUser = SecurityUtils.getSubject();

if (!subject.isPermitted("account:create")) {

throw new AuthorizationException(...);

}

//Subject is guaranteed to be permitted here

...

}

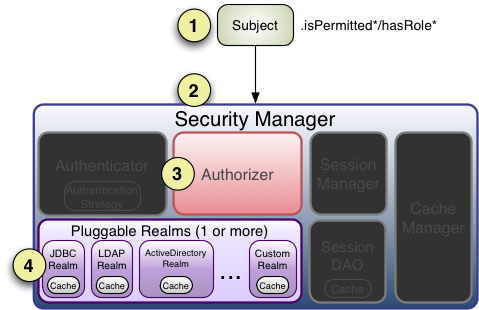
#### RequiresRoles annotation

#### The RequiresUser annotation

### [JSP TagLib Authorization](http://shiro.apache.org/authorization.html#jsp-taglib-authorization)

Shiro offers a Tag Library for controlling JSP/GSP page output based on Subject state. This is covered in the [Web](http://shiro.apache.org/web.html) chapter’s [JSP/GSP Tag Library](http://shiro.apache.org/web.html#Web-taglibrary) section.

## [Authorization Sequence](http://shiro.apache.org/authorization.html#authorization-sequence)



1, 应用程序或框架代码调用任何Subject 的hasRole\*, checkRole\*, isPermitted\*, or checkPermission\* 方法变体。

2，Subject instance,委托securityManager处理，通过调用securityManager中同名方法hasRole\*, checkRole\*, isPermitted\*, or checkPermission\* 方法变体

3，The SecurityManager交由内部[org.apache.shiro.authz.Authorizer](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/Authorizer.html) 实例处理。通过调用同名方法

默认的Authorizer实例是一个ModularRealmAuthorizer instance。它支持在任何授权操作过程中协调一个或多个 Realm 实例

4，每个配置的 Realm检查是否实现了相同的Authorizer interface。If so, the Realm’s own同名方法hasRole\*, checkRole\*, isPermitted\*, or checkPermission\* 方法变体

Realm中的方法处理步骤

1. 如果Realm实现 了[Authorizer](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/Authorizer.html) interface,
   1. 如果 Realm 的方法导致异常, 则该异常将作为授权异常传播给Subject调用方，剩余的 Realms不会执行授权检查
   2. 如果 Realm 的方法是一个，hasRole\* or isPermitted\* variant它返回一个布尔值, 并且返回值为真。true立即被返回, 任何剩余的Realms被短路
2. 未实现则忽略

#### Configuring a global PermissionResolver

在执行基于字符串的权限检查时, Shiro 的大多数默认 Realm 实现都将此 String 转换为实际的[Permission](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/Permission.html)实例, 然后再执行权限实例的隐式逻辑。

#### 为了帮助这种转换, Shiiro 支持 PermissionResolver的概念。

所有Realm使用默认的[WildcardPermissionResolver](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/permission/WildcardPermissionResolver.html)来转换Shiro’s [WildcardPermission](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/permission/WildcardPermission.html) String format.

使用自定义PermissionResolver实现

globalPermissionResolver = com.foo.bar.authz.MyPermissionResolver

...

securityManager.authorizer.permissionResolver = $globalPermissionResolver

#### Configuring a global RolePermissionResolver

当需要将角色名称转换为一组具体的权限实例时, Realm 可以在内部使用 RolePermissionResolver。

# Understanding Permissions in Apache Shiro

如何向用户授予权限有许多不同的情况

## [Wildcard Permissions](http://shiro.apache.org/permissions.html#wildcard-permissions)通配符权限

## 假设您要保护对公司打印机的访问, 以便某些人可以打印到特定的打印机, 而其他人可以查询队列中当前的作业。

### [Multiple Parts](http://shiro.apache.org/permissions.html#multiple-parts)

通配符权限支持多个级别或部件的概念printer:query

## 在本例中,第一部分是正在操作的主体(打印机), 第二部分是正在执行的操作 (查询)。

## 可以使用的部件数量没有限制

#### [Multiple Values](http://shiro.apache.org/permissions.html#multiple-values)

## 每个部件可以包含多个值。授予用户 printer:print和printer:query可以写成 printer:print,query

#### [All Values](http://shiro.apache.org/permissions.html#all-values)

## 如果以后向应用程序添加了新操作, 则不需要更新使用该部分中通配符的权限。

printer:\*

## 最后, 还可以在通配符权限字符串的任何部分中使用通配符标记

## 如果您想在所有主体 (而不仅仅是打印机) 中授予用户 "视图" 操作, 则可以授予此:

\*:view

### Instance-Level Access Control

在这种情况下, 您使用三个部分-第一部分是域, 第二部分是操作, 第三个部分是要对其执行操作的实例。

#### [Missing Parts](http://shiro.apache.org/permissions.html#missing-parts)

One final thing to note about permission assignments: missing parts imply that the user has access to all values corresponding to that part. In other words,

printer:print

is equivalent to

printer:print:\*

但是, 您只能从字符串的末尾离开部分, 因此, 您可以这样做:

printer:lp7200

is ***not*** equivalent to

printer:\*:lp7200

但运行时的权限检查应尽可能基于最具体的权限字符串。

## [Implication, not Equality](http://shiro.apache.org/permissions.html#implication-not-equality)

为什么运行时权限检查应该尽可能具体, 但权限分配可能更通用一些？这是因为权限检查是通过隐含逻辑 (而不是相等检查) 来计算的。

为了支持隐含规则, 所有权限都转换为实现 org.apache.shiro.authz.Permission 接口的对象实例。Shiro默认提供的通配符行为实际上都是由 org.apache.shiro.authz.permission.WildcardPermission 类实现的。

## [Realm Authorization](http://shiro.apache.org/realm.html#realm-authorization)

3然后授权者逐一引用所有授权领域, 直到它找到分配给主体的角色. 如果没有任何领域授予主题特定角色, 则通过返回 false 来拒绝访问

4Realm返回的AuthorizationInfo.getRoles获取分配给Subject的所有角色

5如果调用AuthorizationInfo.getRoles方法返回的角色列表中找到给定的角色, 则授予访问权限。

Realm授权功能执行以下操作, 以检查是否允许主题

首先, 它通过在AuthorizationInfo上调用getStringPermissions和 GetObjectPermissions 方法并聚合结果来获取直接分配给主题的所有权限。

如果注册了 RolePermissionResolver, 它将用于通过调用 RolePermissionResolver.resolvePermissionsInRole(). 根据分配给object的所有角色检索权限。

聚合上面的权限, 调用 "implies ()" 方法来检查这些权限中是否有任何一个包含选中的权限。

# [Session Management](http://shiro.apache.org/session-management.html#session-management)

提供了适用于任何应用程序的完整企业级会话解决方案, 从最简单的命令行和智能手机应用程序到最大的群集企业 web 应用程序。

在shiro之前，如果您需要会话支持, 则需要在 web 容器中部署应用程序或使用 EJB 状态会话 Bean。与这两种机制相比, shiiro 的会话支持的使用和管理要简单得多, 无论容器如何, 它都可在任何应用程序中使用。

即使您在 Servlet 或 EJB 容器中部署应用程序, 仍有令人信服的理由使用 Shiiro 的 session 支持而不是容器的支持。以下是 Shiiro 会话支持提供的最理想功能的列表:

## [Using Sessions](http://shiro.apache.org/session-management.html#using-sessions)

Subject currentUser = SecurityUtils.getSubject();

Session session = currentUser.getSession();

session.setAttribute( "someKey", someValue);

getSession()方法是调用当前用户. getsession (true) 的快捷方式。

## [SessionManager](http://shiro.apache.org/session-management.html#the-sessionmanager)

管理应用程序中所有object的会话-创建、删除、不活动和验证等。它是由SecurityManager维护的顶级组件

默认的SecurityManager使用默认[DefaultSessionManager](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/mgt/DefaultSecurityManager.html)作为SessionManager。

[main]

...

sessionManager = com.foo.my.SessionManagerImplementation

securityManager.sessionManager = $sessionManager

### [Session Timeout](http://shiro.apache.org/session-management.html#session-timeout)

sessionManager默认30分钟会话超时

设置sessionManager 的globalSessionTimeout属性来定义所有会话的默认超时值

[main]

...

# 3,600,000 milliseconds = 1 hour

securityManager.sessionManager.globalSessionTimeout = 3600000

单个会话的超时，设置Session的[timeout](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/session/Session.html" \l "setTimeout-long-) 属性

### [Session Listeners](http://shiro.apache.org/session-management.html#session-listeners)

Shiro支持会话监听器的概念, 允许您在重要的会话事件发生时做出反应。您可以实现 Sessionli管机接口 , 并对会话操作做出相应的反应。

SessionManager的sessionListeners属性是一个集合，可以添加多个监听器到SessionManager

[main]

...

aSessionListener = com.foo.my.SessionListener

anotherSessionListener = com.foo.my.OtherSessionListener

securityManager.sessionManager.sessionListeners = $aSessionListener, $anotherSessionListener

当任何会话 (而不仅仅是特定会话) 发生事件时, 都会通知sessionListener。

### [Session Storage](http://shiro.apache.org/session-management.html#session-storage)

Session数据的CRUD交由SessionManager的内部组件[SessionDAO](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/session/mgt/eis/SessionDAO.html)来实现。[SessionDAO](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/session/mgt/eis/SessionDAO.html)反映了数据访问对象 (DAO) 的设计模式。

SessionDAO 的强大功能在于, 您可以实现此接口, 以便与您希望的任何数据存储进行交互

这意味着会话数据可以驻留在内存、文件系统、关系数据库或 NoSQL 数据存储中, 也可以驻留在所需的任何其他位置

Shiroo 已经有一些很好的 SessionDAO 实现, 可以开箱即用, 以满足自己的需要

Shiro 的默认SessionDAO将会话保存在内存中。

**Configuring a SessionDAO in shiro.ini**

[main]

...

sessionDAO = com.foo.my.SessionDAO

securityManager.sessionManager.sessionDAO = $sessionDAO

securityManager.sessionManager.sessionDAO = $sessionDAO只在使用Shiro自身的session manager有效

web 应用程序默认使用servlet 容器的默认会话管理器，此会话管理器不支持sessionDAO。

需要修改使用shiro的session manager

#### [EHCache SessionDAO](http://shiro.apache.org/session-management.html#ehcache-sessiondao)

EHCache SessionDAO 将会话存储在内存中, 并支持在内存受到限制时保存到磁盘。还可以缓存authentication and authorization data

First, ensure that you have the shiro-ehcache-<version>.jar file in your classpath

[main]

sessionDAO = org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO

securityManager.sessionManager.sessionDAO = $sessionDAO

#### [Custom Session IDs](http://shiro.apache.org/session-management.html#custom-session-ids)

Shiiro 的 Session dao 使用内部的 SessionIdGenerator组件在每次创建新会话时生成新的会话 ID。 默认的 SessionIdGenerator是一个 JavaUuidSessionIdGenerator, 它基于 Java Uuid 生成字符串 id。

可以自己实现SessionIdGenerator 接口

**Configuring a SessionIdGenerator in shiro.ini**

[main]

...

sessionIdGenerator = com.my.session.SessionIdGenerator

securityManager.sessionManager.sessionDAO.sessionIdGenerator = $sessionIdGenerator

### Session Validation & Scheduling

出于性能原因, 会话仅在访问时被验证为是否已停止或过期 (object. getsession ())。这意味着, 如果没有额外的定期验证, 孤立的会话将填充会话数据存储。

当用户登录网站，并拥有了一个session存储。但用户没有logout也没有关闭浏览器。SessionManager 无法检测用户是否不再使用其浏览器, 并且永远不会再次访问该会话 (它是孤立的)。

SessionValidationScheduler负责以周期性速率验证会话, 以确保根据需要对其进行清理

# [Apache Shiro Web Support](http://shiro.apache.org/web.html#apache-shiro-web-support)

将 Shiiro 集成到web 应用程序中的最简单方法是在 web. xml 中配置一个 Servlet ContextListener和 Filter, 该筛选器能够理解如何读取 Shiiro 的 INI 配置。

<listener>

<listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>

</listener>

...

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

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

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

<dispatcher>INCLUDE</dispatcher>

<dispatcher>ERROR</dispatcher>

</filter-mapping>

这假定 Shiroi Ini 配置文件位于以下两个位置之一

1. /WEB-INF/shiro.ini
2. shiro.ini file at the root of the classpath.

shiro.ini 文件中的[urls] 部分

为应用程序中的任何匹配 URL 路径定义临时筛选器链!

[urls]部分中每行的格式如下所示:

\_URL\_Ant\_Path\_Expression\_ = \_Path\_Specific\_Filter\_Chain\_

等号左侧的标记是相对于 web 应用程序的上下文的根 ant 样式路径表达式。（相对于[HttpServletRequest.getContextPath()](http://docs.oracle.com/javaee/1.3/api/javax/servlet/http/HttpServletRequest.html#getContextPath--)的值）

/account/\*\* = ssl, authc

任何相对于应用程序路径/account，或它的子路径（/account/foo, /account/bar/baz, etc）都将触发过滤器链‘ssl, authc’ 。[urls]部分按照定义的顺序匹配请求。第一个匹配到的过滤器链将短路后面的配置。

等号右边的过滤链定义

filter1[optional\_config1], filter2[optional\_config2], ..., filterN[optional\_configN]

filterN是在 [main] 部分中定义的filter bean 的名称,

自定义的过滤器要想识别[optional\_configN]，需要实现[org.apache.shiro.web.filter.PathMatchingFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/PathMatchingFilter.html)接口

## [Default Filters](http://shiro.apache.org/web.html#default-filters)

运行 web 应用时, Shiro 将创建一些有用的默认筛选器实例, 并使其在 [main] 部分中自动可用

[main]

...

# Notice how we didn't define the class for the FormAuthenticationFilter ('authc') - it is instantiated and available already:

authc.loginUrl = /login.jsp

...

[urls]

...

# make sure the end-user is authenticated. If not, redirect to the 'authc.loginUrl' above,

# and after successful authentication, redirect them back to the original account page they

# were trying to view:

/account/\*\* = authc

...

默认的过滤器如下

| **Filter Name** | **Class** |
| --- | --- |
| anon | [org.apache.shiro.web.filter.authc.AnonymousFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/AnonymousFilter.html) |
| authc | [org.apache.shiro.web.filter.authc.FormAuthenticationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/FormAuthenticationFilter.html) |
| authcBasic | [org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/BasicHttpAuthenticationFilter.html) |
| logout | [org.apache.shiro.web.filter.authc.LogoutFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/LogoutFilter.html) |
| noSessionCreation | [org.apache.shiro.web.filter.session.NoSessionCreationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/session/NoSessionCreationFilter.html) |
| perms | [org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/PermissionsAuthorizationFilter.html) |
| port | [org.apache.shiro.web.filter.authz.PortFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/PortFilter.html) |
| rest | [org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/HttpMethodPermissionFilter.html) |
| roles | [org.apache.shiro.web.filter.authz.RolesAuthorizationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/RolesAuthorizationFilter.html) |
| ssl | [org.apache.shiro.web.filter.authz.SslFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/SslFilter.html) |
| user | [org.apache.shiro.web.filter.authc.UserFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/UserFilter.html) |

## [Enabling and Disabling Filters](http://shiro.apache.org/web.html#enabling-and-disabling-filters)

Shiro 的所有开箱即用的filter都实现了[OncePerRequestFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/servlet/OncePerRequestFilter.html)的抽象父类, 因此可以在不从筛选器链中删除它们的情况下启用或禁用它们。

[main]

...

# configure Shiro's default 'ssl' filter to be disabled while testing:

ssl.enabled = false

### Request-specific Enabling/Disabling

### Path-specific Enabling/Disabling

## [Session Management](http://shiro.apache.org/web.html#session-management)

在 web 环境中, Shiiro 的默认会话管理器 [SessionManager](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/session/mgt/SessionManager.html)是 [ServletContainerSessionManager](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/session/mgt/ServletContainerSessionManager.html) [ServletContainerSessionManager](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/session/mgt/ServletContainerSessionManager.html) 把会话管理交给运行时Servlet container.处理。它本质上是 Shiiro会话 API 连接servlet 容器的的桥梁, 几乎不执行其他操作。

此默认值的缺点是, 您与 servlet 容器的特定会话行为相关联。例如, 如果您想要群集会话, 但在生产中使用 Jetty 进行测试和 Tomcat, 则您的容器特定配置 (或代码) 将不可移植。

#### [Servlet Container Session Timeout](http://shiro.apache.org/web.html#servlet-container-session-timeout)

如果使用默认 servlet 容器支持, 则可以按照 web 应用程序的 web. xml 文件中的预期配置会话超时

<session-config>

<!-- web.xml expects the session timeout in minutes: -->

<session-timeout>30</session-timeout>

</session-config>

### [Native Sessions](http://shiro.apache.org/web.html#native-sessions)

如果希望会话配置设置和群集可跨 servlet 容器进行移植, 您可以启用 Shiiro 的本身的会话管理

Native sessions 是Shiro 自己实现的企业会话管理器。它可以用于支持所有Subject and HttpServletRequest session, 并完全绕过 servlet 容器。

#### 要为 web 应用程序启用Native sessions管理，替换默认的serlet容器会话管理。使用[DefaultWebSessionManager](http://shiro.apache.org/web.html" \l "defaultwebsessionmanager)的实例

[main]

...

sessionManager = org.apache.shiro.web.session.mgt.DefaultWebSessionManager

# configure properties (like session timeout) here if desired

# Use the configured native session manager:

securityManager.sessionManager = $sessionManager

对于 web 应用程序。authc 过滤器是一个 [FormAuthenticationFilter](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authc/FormAuthenticationFilter.html)实例。默认情况下, FormAuthenticationFilter将查找名字是的username, password 和rememberMe 的request parameters。如果这些名字和表单中定义的名字不一样，可以在shiro.ini中修改FormAuthenticationFilter的属性

[main]

...

authc.loginUrl = /whatever.jsp

authc.usernameParam = somethingOtherThanUsername

authc.passwordParam = somethingOtherThanPassword

authc.rememberMeParam = somethingOtherThanRememberMe

...

可以通过设置默认 {Remermemanc} 的各种 cookie 属性来配置 rememberMe cookie 的功能

[main]

...

securityManager.rememberMeManager.cookie.name = foo

securityManager.rememberMeManager.cookie.maxAge = blah

### Custom RememberMeManager

如果默认的基于 cookie 的 Remmemanager 实现不能满足您的需要，可以使用自定义的

[main]

...

rememberMeManager = com.my.impl.RememberMeManager

securityManager.rememberMeManager = $rememberMeManager

将 Apache Shirod 集成到基于Spring的项目

使用Shiiro 的应用程序需要一个SecurityManager 的单利，不管是否static的

在 web 应用程序中, 所有 shiro 可访问的 web 请求都必须经过master Shiro Filter处理。这种过滤器本身非常强大

加密解密