Shiro提供了完整的企业级会话管理功能，不依赖于底层容器（如web容器tomcat），不管JavaSE还是JavaEE环境都可以使用，提供了会话管理、会话事件监听、会话存储/持久化、容器无关的集群、失效/过期支持、对Web的透明支持、SSO单点登录的支持等特性。即直接使用Shiro的会话管理可以直接替换如Web容器的会话管理。

**会话**

所谓会话，即用户访问应用时保持的连接关系，在多次交互中应用能够识别出当前访问的用户是谁，且可以在多次交互中保存一些数据。如访问一些网站时登录成功后，网站可以记住用户，且在退出之前都可以识别当前用户是谁。

Shiro的会话支持不仅可以在普通的JavaSE应用中使用，也可以在JavaEE应用中使用，如web应用。且使用方式是一致的。

login("classpath:shiro.ini", "zhang", "123");

Subject subject = SecurityUtils.getSubject();

Session session = subject.getSession();

登录成功后使用Subject.getSession()即可获取会话；其等价于Subject.getSession(true)，即如果当前没有创建Session对象会创建一个；另外Subject.getSession(false)，如果当前没有创建Session则返回null（不过默认情况下如果启用会话存储功能的话在创建Subject时会主动创建一个Session）。

session.getId();

session.getHost();

session.setTimeout(毫秒);

session.getStartTimestamp();

session.getLastAccessTime();

获取会话的启动时间及最后访问时间；如果是JavaSE应用需要自己定期调用session.touch()去更新最后访问时间；如果是Web应用，每次进入ShiroFilter都会自动调用session.touch()来更新最后访问时间。

session.touch();

session.stop();

更新会话最后访问时间及销毁会话；当Subject.logout()时会自动调用stop方法来销毁会话。如果在web中，调用javax.servlet.http.HttpSession. invalidate()也会自动调用Shiro Session.stop方法进行销毁Shiro的会话。

session.setAttribute("key", "123");

Assert.assertEquals("123", session.getAttribute("key"));

session.removeAttribute("key");

设置/获取/删除会话属性；在整个会话范围内都可以对这些属性进行操作。

**会话管理器**

会话管理器管理着应用中所有Subject的会话的创建、维护、删除、失效、验证等工作。是Shiro的核心组件，顶层组件SecurityManager直接继承了SessionManager，且提供了SessionsSecurityManager实现直接把会话管理委托给相应的SessionManager，DefaultSecurityManager及DefaultWebSecurityManager默认SecurityManager都继承了SessionsSecurityManager。

**SecurityManager提供了如下接口：**

Session start(SessionContext context); //启动会话

Session getSession(SessionKey key) **throws** SessionException; //根据会话Key获取会话

另外用于Web环境的WebSessionManager又提供了如下接口：

**boolean** isServletContainerSessions();//是否使用Servlet容器的会话

Shiro还提供了ValidatingSessionManager用于验资并过期会话：

**void** validateSessions();//验证所有会话是否过期

**会话存储/持久化**

Shiro提供SessionDAO用于会话的CRUD，即DAO（Data Access Object）模式实现：

//如DefaultSessionManager在创建完session后会调用该方法；如保存到关系数据库/文件系统/NoSQL数据库；即可以实现会话的持久化；返回会话ID；主要此处返回的ID.equals(session.getId())；

Serializable create(Session session);

//根据会话ID获取会话

Session readSession(Serializable sessionId) **throws** UnknownSessionException;

//更新会话；如更新会话最后访问时间/停止会话/设置超时时间/设置移除属性等会调用

**void** update(Session session) **throws** UnknownSessionException;

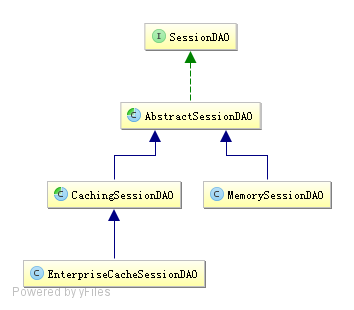
//删除会话；当会话过期/会话停止（如用户退出时）会调用

**void** delete(Session session);

//获取当前所有活跃用户，如果用户量多此方法影响性能

Collection<Session> getActiveSessions();

Shiro内嵌了如下SessionDAO实现：



AbstractSessionDAO提供了SessionDAO的基础实现，如生成会话ID等；CachingSessionDAO提供了对开发者透明的会话缓存的功能，只需要设置相应的CacheManager即可；MemorySessionDAO直接在内存中进行会话维护；而EnterpriseCacheSessionDAO提供了缓存功能的会话维护，默认情况下使用MapCache实现，内部使用ConcurrentHashMap保存缓存的会话。

可以通过如下配置设置SessionDAO：

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

sessionManager.sessionDAO=$sessionDAO

Shiro提供了使用Ehcache进行会话存储，Ehcache可以配合TerraCotta实现容器无关的分布式集群。

首先在pom.xml里添加如下依赖：

**Java代码  [收藏代码](javascript:void())**

1. <dependency>
2. <groupId>org.apache.shiro</groupId>
3. <artifactId>shiro-ehcache</artifactId>
4. <version>1.2.2</version>
5. </dependency>

接着配置shiro-web.ini文件：

**Java代码  [收藏代码](javascript:void())**

1. sessionDAO=org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO
2. sessionDAO. activeSessionsCacheName=shiro-activeSessionCache
3. sessionManager.sessionDAO=$sessionDAO
4. cacheManager = org.apache.shiro.cache.ehcache.EhCacheManager
5. cacheManager.cacheManagerConfigFile=classpath:ehcache.xml
6. securityManager.cacheManager = $cacheManager

sessionDAO.activeSessionsCacheName：设置Session缓存名字，默认就是shiro-activeSessionCache；

cacheManager：缓存管理器，用于管理缓存的，此处使用Ehcache实现；

cacheManager.cacheManagerConfigFile：设置ehcache缓存的配置文件；

securityManager.cacheManager：设置SecurityManager的cacheManager，会自动设置实现了CacheManagerAware接口的相应对象，如SessionDAO的cacheManager；

然后配置ehcache.xml：

1. <cache name="shiro-activeSessionCache"
2. maxEntriesLocalHeap="10000"
3. overflowToDisk="false"
4. eternal="false"
5. diskPersistent="false"
6. timeToLiveSeconds="0"
7. timeToIdleSeconds="0"
8. statistics="true"/>

Cache的名字为shiro-activeSessionCache，即设置的sessionDAO的activeSessionsCacheName属性值。

另外可以通过如下ini配置设置会话ID生成器：

sessionIdGenerator=org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator

sessionDAO.sessionIdGenerator=$sessionIdGenerator

用于生成会话ID，默认就是JavaUuidSessionIdGenerator，使用java.util.UUID生成。

如果自定义实现SessionDAO，继承CachingSessionDAO即可：

**public** **class** MySessionDAO **extends** CachingSessionDAO {

**private** JdbcTemplate jdbcTemplate = JdbcTemplateUtils.jdbcTemplate();

**protected** Serializable doCreate(Session session) {

        Serializable sessionId = generateSessionId(session);

        assignSessionId(session, sessionId);

        String sql = "insert into sessions(id, session) values(?,?)";

        jdbcTemplate.update(sql, sessionId, SerializableUtils.serialize(session));

**return** session.getId();

    }

**protected** **void** doUpdate(Session session) {

**if**(session **instanceof** ValidatingSession && !((ValidatingSession)session).isValid()) {

**return**; //如果会话过期/停止 没必要再更新了

    }

        String sql = "update sessions set session=? where id=?";

        jdbcTemplate.update(sql, SerializableUtils.serialize(session), session.getId());

    }

**protected** **void** doDelete(Session session) {

        String sql = "delete from sessions where id=?";

        jdbcTemplate.update(sql, session.getId());

    }

**protected** Session doReadSession(Serializable sessionId) {

        String sql = "select session from sessions where id=?";

        List<String> sessionStrList = jdbcTemplate.queryForList(sql, String.**class**, sessionId);

**if**(sessionStrList.size() == 0) **return** **null**;

**return** SerializableUtils.deserialize(sessionStrList.get(0));

    }

}

doCreate/doUpdate/doDelete/doReadSession分别代表创建/修改/删除/读取会话；此处通过把会话序列化后存储到数据库实现；接着在shiro-web.ini中配置：

sessionDAO=com.github.zhangkaitao.shiro.chapter10.session.dao.MySessionDAO

其他设置和之前一样，因为继承了CachingSessionDAO；所有在读取时会先查缓存中是否存在，如果找不到才到数据库中查找。

**会话验证**

Shiro提供了会话验证调度器，用于定期的验证会话是否已过期，如果过期将停止会话；出于性能考虑，一般情况下都是获取会话时来验证会话是否过期并停止会话的；但是如在web环境中，如果用户不主动退出是不知道会话是否过期的，因此需要定期的检测会话是否过期，Shiro提供了会话验证调度器SessionValidationScheduler来做这件事情。

可以通过如下ini配置开启会话验证：

sessionValidationScheduler=org.apache.shiro.session.mgt.ExecutorServiceSessionValidationScheduler

sessionValidationScheduler.interval = 3600000

sessionValidationScheduler.sessionManager=$sessionManager

sessionManager.globalSessionTimeout=1800000

sessionManager.sessionValidationSchedulerEnabled=**true**

sessionManager.sessionValidationScheduler=$sessionValidationScheduler

sessionValidationScheduler：会话验证调度器，sessionManager默认就是使用ExecutorServiceSessionValidationScheduler，其使用JDK的ScheduledExecutorService进行定期调度并验证会话是否过期；

sessionValidationScheduler.interval：设置调度时间间隔，单位毫秒，默认就是1小时；

sessionValidationScheduler.sessionManager：设置会话验证调度器进行会话验证时的会话管理器；

sessionManager.globalSessionTimeout：设置全局会话超时时间，默认30分钟，即如果30分钟内没有访问会话将过期；

sessionManager.sessionValidationSchedulerEnabled：是否开启会话验证器，默认是开启的；

sessionManager.sessionValidationScheduler：设置会话验证调度器，默认就是使用ExecutorServiceSessionValidationScheduler。

Shiro也提供了使用Quartz会话验证调度器：

使用时需要导入shiro-quartz依赖：

<dependency>

     <groupId>org.apache.shiro</groupId>

     <artifactId>shiro-quartz</artifactId>

     <version>1.2.2</version>

</dependency>

如上会话验证调度器实现都是直接调用AbstractValidatingSessionManager 的validateSessions方法进行验证，其直接调用SessionDAO的getActiveSessions方法获取所有会话进行验证，如果会话比较多，会影响性能；可以考虑如分页获取会话并进行验证，如com.github.zhangkaitao.shiro.chapter10.session.scheduler.MySessionValidationScheduler：

**Java代码  [收藏代码](javascript:void())**

//分页获取会话并验证

String sql = "select session from sessions limit ?,?";

**int** start = 0; //起始记录

**int** size = 20; //每页大小

List<String> sessionList = jdbcTemplate.queryForList(sql, String.**class**, start, size);

**while**(sessionList.size() > 0) {

**for**(String sessionStr : sessionList) {

**try** {

      Session session = SerializableUtils.deserialize(sessionStr);

      Method validateMethod =

        ReflectionUtils.findMethod(AbstractValidatingSessionManager.**class**,

            "validate", Session.**class**, SessionKey.**class**);

      validateMethod.setAccessible(**true**);

      ReflectionUtils.invokeMethod(validateMethod,

        sessionManager, session, **new** DefaultSessionKey(session.getId()));

    } **catch** (Exception e) {

        //ignore

    }

  }

 start = start + size;

  sessionList = jdbcTemplate.queryForList(sql, String.**class**, start, size);

}

其直接改造自ExecutorServiceSessionValidationScheduler，如上代码是验证的核心代码，可以根据自己的需求改造此验证调度器器；ini的配置和之前的类似。

如果在会话过期时不想删除过期的会话，可以通过如下ini配置进行设置：

sessionManager.deleteInvalidSessions=**false**

默认是开启的，在会话过期后会调用SessionDAO的delete方法删除会话：如会话时持久化存储的，可以调用此方法进行删除。

如果是在获取会话时验证了会话已过期，将抛出InvalidSessionException；因此需要捕获这个异常并跳转到相应的页面告诉用户会话已过期，让其重新登录，如可以在web.xml配置相应的错误页面：

**Java代码  [收藏代码](javascript:void())**

1. <error-page>
2. <exception-type>org.apache.shiro.session.InvalidSessionException</exception-type>
3. <location>/invalidSession.jsp</location>
4. </error-page>

**sessionFactory**

sessionFactory是创建会话的工厂，根据相应的Subject上下文信息来创建会话；默认提供了SimpleSessionFactory用来创建SimpleSession会话。

首先自定义一个Session：

**Java代码  [收藏代码](javascript:void())**

**public** **class** OnlineSession **extends** SimpleSession {

**public** **static** **enum** OnlineStatus {

        on\_line("在线"), hidden("隐身"), force\_logout("强制退出");

**private** **final** String info;

**private** OnlineStatus(String info) {

**this**.info = info;

        }

**public** String getInfo() {

**return** info;

        }

    }

**private** String userAgent; //用户浏览器类型

**private** OnlineStatus status = OnlineStatus.on\_line; //在线状态

**private** String systemHost; //用户登录时系统IP

    //省略其他

}

OnlineSession用于保存当前登录用户的在线状态，支持如离线等状态的控制。

接着自定义SessionFactory：

**Java代码  [收藏代码](javascript:void())**

**public** **class** OnlineSessionFactory **implements** SessionFactory {

 @Override

**public** Session createSession(SessionContext initData) {

         OnlineSession session = **new** OnlineSession();

**if** (initData != **null** && initData **instanceof** WebSessionContext) {

       WebSessionContext sessionContext = (WebSessionContext) initData;

      HttpServletRequest request = (HttpServletRequest) sessionContext.getServletRequest();

**if** (request != **null**) {

                session.setHost(IpUtils.getIpAddr(request));

                session.setUserAgent(request.getHeader("User-Agent"));

                session.setSystemHost(request.getLocalAddr() + ":" + request.getLocalPort());

            }

        }

**return** session;

    }

}

根据会话上下文创建相应的OnlineSession。

最后在shiro-web.ini配置文件中配置：

sessionFactory=org.apache.shiro.session.mgt.OnlineSessionFactory

sessionManager.sessionFactory=$sessionFactory