Multi-session in single browser

**What is multi user’ session in single browser?**

Normally, when we sign in to any website or account. The credentials managed by session. Storing current user information in session using cookies at client(browser). When a browser sends a request to access any content, a new session is started for that browser. Session is kept live until session is expired or invalidated by signing out.

In single session, a browser can send multiple request for various content available. But in this case, login credential is validated per session only. So that, a browser is bound to only one session.

Problems with this approach is –

* A browser cannot have multiple session simultaneously accessing the same service.
* If a user wishes to sign in with multiple login credentials then he/she has start with new session, i.e., must start new browser.
* For example, we may have multiple gmail account. To all accounts simultaneously, a user has to start using multiple browser.

A new approach is given by Spring to manage multiple session simultaneously in same browser in Spring Session project.

**Why, Spring Session?**

Spring Session provides an API and implementations for managing a user’s session information. As spring is amongst most used framework, it is always to manage other features along with like:

Features

* API and implementations for managing a user's session
* HttpSession - allows replacing the HttpSession in an application container (i.e. Tomcat) neutral way
  + Clustered Sessions - Spring Session makes it trivial to support clustered sessions without being tied to an application container specific solution.
  + Multiple Browser Sessions - Spring Session supports managing multiple users' sessions in a single browser instance (i.e. multiple authenticated accounts similar to Google).
  + RESTful APIs - Spring Session allows providing session ids in headers to work with RESTful APIs
* WebSocket - provides the ability to keep the HttpSession alive when receiving WebSocket messages

Along with these features, latest releases also provide access to store session at specified store also, like Database, In-memory Redis, Hazelcast, GemFire, … etc.

**Using Spring Session?**

How can we use spring session to manage http session?

1. **Required jars**
2. **Configuration**
3. **Creating multiple users**
4. **Applying interceptor**
5. **Required Jars**

Session information can be stored anywhere to access it later. Spring Session facilitates to store this session info at various data store schemes. We can store it in Database, Redis in-memory data store, GemFire, HazelCast … etc. To use it with

Relational database, -- add their jars as MySQL-connector & spring-session-jdbc jar.

compile group:'org.springframework.session',name:'spring-session-jdbc',version: '1.3.1.RELEASE'

Redis data store, -- add spring-session-jdbc & lettuce jars.

compile group:'org.springframework.session',name:'spring-session-jdbc',version: '1.3.1.RELEASE'

compile group: 'biz.paluch.redis', name: 'lettuce', version: '4.2.1.Final'

compile group:'org.springframework.data,name:'spring-data-redis,version: '1.8.6.RELEASE'

GemFire backed spring session

compile group:'org.springframework.session',name:'spring-session-data-gemfire,version: '1.3.1.RELEASE'

1. **Configuration**

Spring uses servlet filter ***SessionRepositoryFilter*** to wrap up the container given HttpSession managed by underlying container in its own HttpSession implementation. Spring session registers a filter which create HttpSession by manipulating existing javax.servlet.htt.HttpSession. This could achieve depending on the configuration i.e., where is the required session stored as stated above ( db, redis, gemfire etc…).

To specify the store for session info, Connection properties are required along with the one of suitable annotation.

**@EnableRedisHttpSession**

To expose the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by Redis. To leverage the annotation, a single RediConnectionFactory must be provided.

For example:

@Configuration

@EnableRedisHttpSession

public class RedisHttpSessionConfig {

@Bean

public JedisConnectionFactory connectionFactory() throws Exception {

return new JedisConnectionFactory();

}

}

**@EnableJdbcHttpSession**

To expose the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by a relational database. In order to leverage the annotation, a single javax.sql.DataSource must be provided.

For example:

@Configuration

@EnableJdbcHttpSession

public class JdbcHttpSessionConfig {

@Bean

public DataSource dataSource() {

return new EmbeddedDatabaseBuilder()

.setType(EmbeddedDatabaseType.H2)

.addScript("org/springframework/session/jdbc/schema-h2.sql")

.build();

}

@Bean

public PlatformTransactionManager transactionManager(DataSource dataSource) {

return new DataSourceTransactionManager(dataSource);

}

}

**@EnableGemFireHttpSession**

To expose the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by Pivotal GemFire or Apache Geode. In order to leverage the annotation, a single Pivotal GemFire/Apache Geode com.gemstone.gemfire.cache.Cache or com.gemstone.gemfire.cache.client.ClientCache instance must be provided.

For example:

@Configuration

@EnableGemFireHttpSession

public class GemFirePeerCacheHttpSessionConfiguration {

@Bean

public Properties gemfireProperties() {

Properties gemfireProperties = new Properties();

gemfireProperties.setProperty("name", "ExamplePeer");

gemfireProperties.setProperty("mcast-port", "0");

gemfireProperties.setProperty("log-level", "warning");

return gemfireProperties;

}

@Bean

public CacheFactoryBean gemfireCache() throws Exception {

CacheFactoryBean cache = new CacheFactoryBean();

cache.setProperties(gemfireProperties());

return cache;

}

}

Now, to register the filter ***SessionRepositoryFilter*** in the application container, we have to register a DelegatingFilterProxy with highest order so that this filter would be register to be first filter in the filter chain to followed by every request/response to application.

In order for our Filter to do its magic, Spring needs to load our Config class. Last we need to ensure that our Servlet Container (i.e. Tomcat) uses our springSessionRepositoryFilter for every request. Fortunately, Spring Session provides a utility class named AbstractHttpSessionApplicationInitializer both these steps extremely easy.

As given below:

public class Initializer extends *AbstractHttpSessionApplicationInitializer* {

public Initializer() {

super(Config.class);

}

}

Or, configuration class can be loaded by other means also, as it is done in spring mvc.

1. **Creating Multiple Users’ Session**

All of the above description is all about managing http session in spring application. Which doesn’t make any drastic change to approach or significance of the application. One of the greatest application so spring session is managing multiple users' sessions in a single browser instance (i.e. multiple authenticated accounts like Google). To allow having multiple active sessions from a single browser at the same time.

Managing a single session:

Spring Session keeps track of the HttpSession by adding a value to a cookie named SESSION. For example, the SESSION cookie might have a value of:

7e8383a4-082c-4ffe-a4bc-c40fd3363c5e

Adding a new Session:

We can add another session by requesting a URL that contains a special parameter in it. By default, the parameter name is \_s. For example, the following URL would create a new session:

http://localhost:8080/?\_s=1

The parameter value does not indicate the actual session id. Rather than creating the URL ourselves, we can utilize the HttpSessionManager to do this for us.

*HttpSessionManager*

We can obtain the HttpSessionManager from the HttpServletRequest using the following:

HttpSessionManager sessionManager = (HttpSessionManager) httpRequest

.getAttribute(HttpSessionManager.class.getName());

//Creating a new alias for session

String addAlias = unauthenticatedAlias == null ?

sessionManager.getNewSessionAlias(httpRequest) :

unauthenticatedAlias;

String addAccountUrl = sessionManager.encodeURL(contextPath, addAlias);

HttpSessionManager keeps the mapping of created session in SESSION cookie. We can get the session alias & session id from hhtsessionmanager. Now, SESSION cookie looks something like this:

0 7e8383a4-082c-4ffe-a4bc-c40fd3363c5e 1 1d526d4a-c462-45a4-93d9-84a39b6d44ad

*SessionRepository*

To manage all the session instance created by a user, Spring session has given an interface SessionRepository. This interface allows to create, get, delete or save the spring session instances. We can obtain the SessionRepository from HttpServletRequest as earlier.

SessionRepository<Session> repo = (SessionRepository<Session>) request.getAttribute(SessionRepository.class.getName());

The HttpServletRequest’ getAttribute() Returns the value of the named attribute as an Object, or null if no attribute of the given name exists. At the time of request make by application, the registered *springSessionRepositoryFilter* will add this attribute to each generated request object.

* To get the current session or any session, first we need to get session id from httpsessionmanager & then can fetch the session from sessionrepository using the same session id as below:

Map<String, String> sessionIds = sessionManager.getSessionIds(request);

**or**

String currentAlias = sessionManager.getCurrentSessionAlias(request);

* To get the session from session repository,

Session session = sessionRepository.getSession(sessionId);

Likewise, we can make use of httpsessionmanager to acquire alias to add a new session. The benefit of specifying the session alias in the URL is that we can have multiple tabs open with different active sessions. The bad thing is that we need to include the session alias in every URL of our application. For this, Spring Session will automatically include the session alias in any URL that passes through as:

String addAlias = sessionManager.getNewSessionAlias(request);

String addAccountUrl = sessionManager.encodeURL(contextPath, addAlias);

This will add url to add another session for current browser like --

**?\_s=1** where 1 is session alias and \_s is default url parameter.

1. **Applying interceptor**

Now, the question arises where should I do these management of sessions. In spring environment to make to make this work efficiently, we can add these session management to handler interceptor as it can easily access the coming request & response object in prehandle or posthandle of interceptor.

It also allows to add to add necessary parameter to request or session scope that can be used to identify the session attributes as per requirement. However, it’s mandatory to use this way. To register a handler interceptor in spring using configuration, if using WebMvcConfigurerAdapter to configure spring mvc then can override addInterceptors() method to add interceptor as follows:

@Configuration

@EnableWebMvc

public class MvcConfig extends WebMvcConfigurerAdapter {

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(new SessionInterceptor());

}

}

Now, can override the postHandle() or preHandle() to manipulate the sessions. Instead, if we are using Servlet to manager user login or session then same can be done by registering a filter.

***For References:***

<http://docs.spring.io/spring-session/docs/current/reference/html5/guides/users.html>

<http://docs.spring.io/spring-session/docs/current/reference/html5/guides/httpsession.html>

<https://www.infoq.com/articles/Next-Generation-Session-Management-with-Spring-Session>