Design document for User Session Management API in Kaeru

Introduction

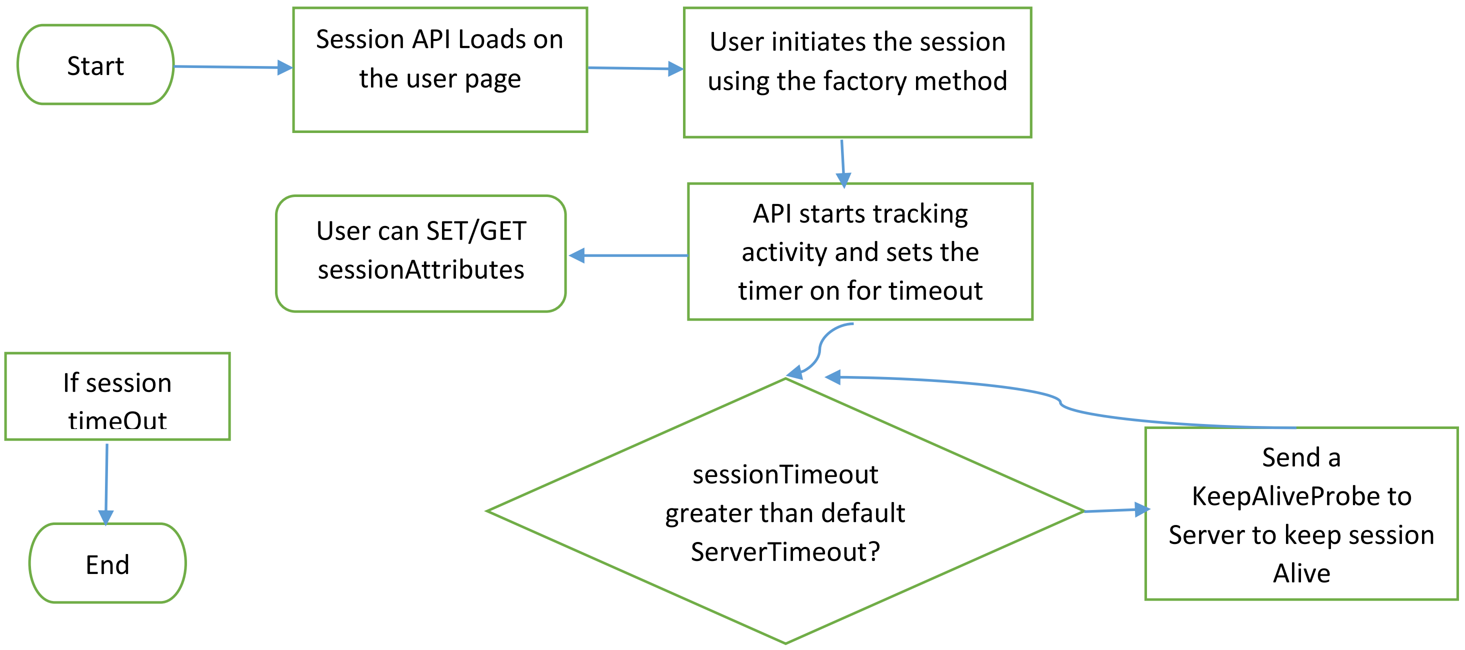
A mechanism needs to be developed to allow the user to handle session on top of the application server session for user web pages. An API needs to be created that allows the user to create and invalidate sessions.

Design

The Session API is written in JavaScript which acts as a second layer of session for the user web pages. Sessions created through this API will run along with the normal session of the existing webserver. User would be able to manage session using setting up server timeout, timeout redirect page different than the one of the web server. Use would also be able to set and get session attributes. All these is done locally using the JavaScript and the code resides on the users' page.

Since there could be only one session object in the application, the API for the User session is Singleton in nature. The user won't be able to do a "new Session()". There is a factory method provided through which the user can request a session instance. As soon as the session instance is requested, the timer/tracking of user activities starts. The API manages the conflict between the difference of server timeout durations. Further details are provided below.

Flow


Start
Session API Loads on the user page
User initiates the session using the factory method
API starts tracking activity and sets the timer on for timeout
User can SET/GET sessionAttributes
sessionTimeout greater than default ServerTimeout?
Send a KeepAliveProbe to Server to keep session Alive
If session timeOut
End

Component/Function details

createSession()

The session function written in JavaScript is Singleton in nature. This is implemented by setting the instance constructor property to null.

The user can use the function *createSession*() to receive an instance. If the call to *createSession*() has happened for the first time, the API will create a new Session instance and return. However after that if the user calls the factory method multiple times, the same instance is returned.

trackUserActivity()

As soon as the instance is created and/or returned, the API calls an internal function called *trackUserActivity*(). This function initiates a timer which would go off after the sessioTimeout duration occurs without any user activity.

sendKeepAliveProbe()

Since this API helps create a separate session, if the user sets a session time out which is greater than the server timeout, the server will forger the user much before the user would expect the system to do that. To overcome this issue, the API checks if the session timeout is greater than the server timeout, if so, it sends a keep alive probe to the server just before the session is about to die.

Function *sendKeepAliveProbe()* is called to do the above mentioned operation. The URL pattern */keepAliveProbe* is called which calls a View method k*aeru.views.keep_alive_probe .* The challenge here is that the servercall will wipe off all the variables that we have set so far in the JavaScript API because of the page refresh. To avoid that, the *sendKeepAliveProbe()* method uses *AJAX* to make a server call.

Session Attributed

User can store session attributes using the session API. The API internally uses a MAP data structure to store session attributes. This Map gets invalidated when the session expires. More details about the Map data structure implement in JavaScript could be found the ‘Custom Types for Kaeru’ document.

Usage

//Start a session
//createSession takes two parameters. 1) sessionTimeout in minutes 2) Page to redirect to when timeout occurs

```
var session = Session.createSession(10, '/login');

//SET/Get session attributes. These are wiped off at every timeout

session.setAttribute(key,value)

session.getAttribute(key) //returns the value at "key"
```

Test Cases

No	Test Case details	Expected Result	Actual Result
1	Use should not be able to create a new object of the Session class.	JavaScript error.	As expected
2	When the user requests the session object multiple times, the same object should be returned and no new instances should be created	Same object returns. Session parameters should remain the same.	As expected.
3	Session times out according to the value passed by the user in minutes.	Session times out with an alert box in JavaScript if no activity is performed until the amount of time user entered	As expected
4	Get the server timeout and check if the user session expires prematurely if the limit given by the user exceeds that of the server.	Irrespective of the server timeout, session should expire only after the amount of time entered by the user	Encountered a bug initially. Current status is Fixed and results are as expected.
5	NEGATIVE: How does the system behave if the user doesn't give any timeout limit while requesting the session object.	Session should expire after a default timeout which should be equal to the server timeout.	As expected
6	Does the system automatically redirect to the page mentioned while requesting the session object. Note: Since we are using Django as web server, this test case assumes the view URLs configured in the URL.py file.	Auto redirect to the page described by the user. Error page if URL pattern not maintained in Django configuration.	As expected.
7	Are the values of the session attributes retained when the GET/SET operations are done. This test case also tests the robustness of the Map implementation that we have done.	Return the right value of attributes when GET operation is called for already saved attributes.	As expected.
8	Is the user able to save two attributes with the same name.	Use should not be able to save attributes with	As expected. Session attribute values are

		the same name. Since we do not have any error reporting mechanism, what we expect is that the duplicate attribute wont be stored and session would update the value of the attribute.	updated and no duplicate keys are added.
9	Are the session attributes cleaned up is the session expires. Read an already saved session attribute after session expires.	Such a read should return null. The Map that stores the session attributes should be flushed on session expiry.	As expected