APISession

APISession.

APISession what's that ?

net.anotheria.anoplass.api.session.APISession - interface which serves as internal session inside anotheria - API (net.anotheria.anoplass.api.API, ano-plass project), it provides for us (our needs) more functionality than the javax.servlet.http.HttpSession, and should be preferred to the http session in ano-technologies.

Session attributes policies.

In interface section of current document - we can see policies constants. What's that ?

This constants provide different states for attributes in APISession.

Let's look closer :

Policy name	Integer value	Short description
POLICY_LOCAL	1	This policy states that the attribute is for local use only and shouldn't be distributed.
POLICY_DISTRIBUTED	2	This policy states that the attribute will be distributed to other servers. Only usable with Serializable attributes. Please refer to "Session distribution mechanism" for more information. (**Currently not fully implemented**)
POLICY_PERSISTENT	4	This policy states that the attribute will survive system or server restart. Only usable with Serializable attributes. **This feature isn't yet implemented.**
POLICY_AUTOEXPIRE	8	If set this policy defines that after the specified expire period the attribute will be reseted and not visible to the caller. The attribute itself will not be deleted until explicitly overwritten or removed, so don't use this policy on timer-bound attributes, since that can lead to unexpected behaviour. However, you shouldn't put timer-bound attributes in the session either way. note : DEFAULT_EXPIRE_PERIOD = 5 minutes!
POLICY_REUSE_WRAPPER	16	This policy ensures that instead of making a new AttributeWrapper, the old one will be reused (if available). This is important if you want to keep some wrapper attributes like auto-expire. If you put an attribute with an auto-expire policy on but without the reuse_wrapper policy, the auto-expiring will be reset on each setAttribute call. If you want to keep a 'global' auto-expiring.
POLICY_SURVIVE_LOGOUT	32	This policy enables the attribute to survive the session clean-up on logOut.
POLICY_COOKIE_PERSISTENT	64	Those attributes are persistent in cookies.
POLICY_FLASH	128	Flashing attributes are used exactly one time, after first usage they disappear from session.(Available since 2.1.1 version)

And also there is POLICY_DEFAULT - for this default POLICY_LOCAL is used.

APISession interface, and small method description.

```
package net.anotheria.anoplass.api.session;

import net.anotheria.util.TimeUnit;
```

```
import java.util.Locale;


/**
 * The API's internal session. This session provides more functionality than the HttpSession and therefore
should be preferred to the http session.
 *
 * @author another
 *
 */
public interface APISession {

    /**
     * Default expiration period for attributes with policy auto-expire.
     */
    public static final long DEFAULT_EXPIRE_PERIOD = TimeUnit.MINUTE.getMillis() * 5; // 5minutes

    /**
     * This policy states that the attribute is for local use only and shouldn't be distributed.
     */
    public int POLICY_LOCAL = 1;
    /**
     * This policy states that the attribute will be distributed to other servers. Only usable with
     * Serializable attributes. <b>This feature isn't yet implemented</b>
     */
    public int POLICY_DISTRIBUTED = 2;

    /**
     * This policy states that the attribute will survive system or server restart. Only usable with
     * Serializable attributes. <b>This feature isn't yet implemented</b>
     */
    public int POLICY_PERSISTENT = 4;
    /**
     * If set this policy defines that after the specified expire period the attribute will be reseted and not
visible
     * to the caller. The attribute itself will not be deleted until explicitly overwritten or removed, so
don't use this
     * policy on timer-bound attributes, since that can lead to unexpected behaviour. However, you shouldn't
put timer-bound
     * attributes in the session either way.
     */
    public int POLICY_AUTOEXPIRE = 8;

    /**
     * This policy ensures that instead of making a new AttributeWrapper, the old one will be reused (if
available).
     * This is important if you want to keep some wrapper attributes like auto-expire. If you put an attribute
with an
     * auto-expire policy on but without the reuse_wrapper policy, the auto-expiring will be reset on each
setAttribute call.
     * If you want to keep a 'global' auto-expiring.
     */
    public int POLICY_REUSE_WRAPPER = 16;

    /**
     * This policy enables the attribute to survive the session clean-up on logOut.
     */
    public int POLICY_SURVIVE_LOGOUT = 32;

    /**
     * Those attributes are persistent in cookies.
     */
    public int POLICY_COOKIE_PERSISTENT = 64;
        /**
         * Flashing attributes are used exactly one time, after first usage they disappear from session.
         */
        public int POLICY_FLASH = 128;

        /**
     * Default attribute policy.
     */
```

```
    public int POLICY_DEFAULT = POLICY_LOCAL;

    /**
     * Sets the object in the session under the given name under the usage of the default policy.
     * @param key the name of the attribute in the session
     * @param value the object to store in the session.
     */
    void setAttribute(String key, Object value);

    /**
     * Sets the object as session attribute into the session under the given policy.
     * @param key the name under which the attribute is stored.
     * @param policy the policy to be applied to the attribute. See POLICY_ constants for details.
     * @param value the value to store in the session.
     */
    void setAttribute(String key, int policy, Object value);

    /**
     * Adds an attribute with given key, policy, value and expiration time.
     * @param key the name under which the attribute is stored.
     * @param policy the policy to be applied to the attribute. See POLICY_ constants for details.
     * @param value the value to store in the session.
     * @param expiresWhen expiration time
     */
    void setAttribute(String key, int policy, Object value, long expiresWhen);

    /**
     * Returns the session attribute with the given key.
     * @param key the key under which the attribute is stored.
     * @return the stored attribute or null if no attribute under that name is stored.
     */
    Object getAttribute(String key);
    /**
     * Removes the attribute with under the given key. If no such attribute is present the method returns
without failing.
     * @param key the attribute name(key) to remove
     */
    void removeAttribute(String key);

    /**
     * Returns the session id.
     * @return id
     */
    String getId();
    /**
     * Returns the ip address.
     * @return ip address
     */
    String getIpAddress();
    /**
     * Sets the ip address which is associated with this session.
     * @param anIpAddress user ip
     */
    void setIpAddress(String anIpAddress);

    /**
     * Returns the user agent string submitted by the browser.
     * @return  user agent
     */
    String getUserAgent();
    /**
     * Sets the user agent string for this session. Called by APIFilter.
     * @param anUserAgent user agent
     */
    void setUserAgent(String anUserAgent);

    /**
     * Called whenever the user performs an explicit logOut.
     */
    void cleanupOnLogout();
```

```
    /**
     * Returns the id of the currently logged in user.
     * @return current user id
     */
    String getCurrentUserId();
    /**
     * Returns the id of the current CMS editor if applicable.
     * @return current cms editor id
     */
    String getCurrentEditorId();

    /**
     * Returns the locale associated with this session. Used to restore the server given locale between the
requests in case we don't use browser supplied locale.
     * @return  locale
     */
    Locale getLocale();

    /**
     * Sets the locale.
     * @param toSet locale
     */
    void setLocale(Locale toSet);
}
}
```

As we can see from interface APISession interface provide for us more possibilities than HttpSession...

In interface we can see overloaded setAttribute method. Lets look closer :

- void setAttribute(String key, Object value) - simply creates new attribute in APISession with APISession.POLICY_DEFAULT;
- void setAttribute(String key, int policy, Object value) - developer may create new attribute, and specify policy;
- void setAttribute(String key, int policy, Object value, long expiresWhen) - now developer can simply specify expiration time (expiresWhen parameter).

LifeCycle.

Please see APISession LifeCycle.

Distribution.

Under next links You will find docs and implementation details.

- Session distribution mechanism;
- APISessionDistribution configuration;
- SessionDistributorService.

How to get ano-plass with SessionDistribution

Features described above available in ano-plass 1.0.4

```
<dependency>
<groupId>net.anotheria</groupId>
<artifactId>ano-plass</artifactId>
<version>1.0.4</version>
</dependency>
```

Don't forget !!!

This data should be added to web.xml.

net.anotheria.anoplass.api.listener.APISessionListener should be added to **web.xml**!

net.anotheria.anoplass.api.filter.APIFilter should be added to **web.xml**.

```
<filter>
            <filter-name>APIFilter</filter-name>

            <filter-class>net.anotheria.anoplass.api.filter.APIFilter</filter-class>
</filter>

<filter-mapping>
            <filter-name>APIFilter</filter-name>
            <url-pattern>/*</url-pattern>
 </filter-mapping>




    <listener>
        <listener-class>
            net.anotheria.anoplass.api.listener.APISessionListener
        </listener-class>
    </listener>
```

Where and how it's used.

APISession inside AbstractAPI.

APISession in ano-plass project used inside net.anotheria.anoplass.api.AbstractAPIImpl. Inside this abstract class APISession holds attributes.

AbstractAPIImpl provide set of methods for attributes management (for details pleases refer to AbstractAPI doc or code):

- protected void setAttributeInSession(String name, Object attribute);
- protected void setAttributeInSession(String name, int policy, Object attribute);
- protected void setAttributeInSession(String name, int policy, Object attribute, long expiresWhen);
- protected void setAttributeInSession(String name, Object attribute, long expiresWhen);
- protected Object getAttributeFromSession(String name);
- public void removeAttributeFromSession(String key);

If You check code - You will find out that AbstractAPIImpl - does not use APISession directly, but via some strange calls, like:

```
@see net.anotheria.anoplass.api.APICallContext
   APISession session = APICallContext.getCallContext().getCurrentSession();
```

What's that ? And for What's that :) (@see below)

APISession inside APICallContext.

net.anotheria.anoplass.api.APICallContext - represents context in which API is executed. The context is assigned to the Thread (as ThreadLocal) and therefore don't need to be synchronized. (For more data please refer to code).

Context always contains link to current APISession behind "currentSession" property, (net.anotheria.anoplass.api.filter.APIFilter care about this - > @see LifeCycle).

So, AbstractAPI - via APICallContext call's to APISession prevent concurrency issues.

> *Now we can say next :*
>
> *Using APISession inside AbstractAPI in current implementation - prevent concurrency issues! :)*

Conclusion.

Thanks :)