慶應義塾

Shibboleth IdP ver.3との戦い

NII学術情報基盤オープンフォーラム2016

慶應義塾ITC本部

細川 達己

Shibboleth IdP ver.3との戦い

はじめに

慶應義塾基盤認証システム

- **学認用IdP（IdP3移行済）**
  - 開発開始：2013年2月
  - 運用開始：2014年10月
  - 対象：大学学生、大学教員、職員
  - ユーザ数：約44,000人
  - uid数：約130,000
- **塾内システム用IdP（IdP3移行テスト中）**
  - 開発開始：2014年5月
  - 運用開始：2014年11月
  - 対象：大学学生、教員、職員、一部卒業生（塾員）他
  - ユーザ数：約60,000人
  - uid数：約163,000
  - アプリ：ポータル、各種塾内サービス、Google Apps、Box等

慶應義塾のShibboleth IdPの特色

- **複数種類IDの乗り入れ**
  - 塾内の3種の主要ID(慶應ID、ITCアカウント、SFC-CNSアカウント)で利用可能
  - どのアカウントでログインしても同一ユーザは同一ユーザとして扱われる（ePPN含む全属性が同じ、ePPN≠uid）
- **Shibboleth専用のOpenLDAPサーバを開発**
  - 学認の全属性（ePTID, isMemberOf等除く）をそのまま提供
  - 参照をロックせずにトランザクション的な高速全件更新が可能
  - 全件更新中もパスワード変更のリアルタイム同期・参照が可能
- **TOTPによる多要素認証（塾内システムIdPのみ）**
  - サイオステクノロジー様開発
- **ロードバランサのみでIdPを冗長化**
  - Cookie Persistenceのみを利用（memcached等は使用せず）

慶應義塾におけるIdP3化・時系列

- **2015年12月上旬**
  - AXIES年次大会@名古屋で「IdP3化って結構大変そう」な危機感
  - テストフェデレーションを申請して実験開始
- **2015年12月中旬**
  - 学認テストフェデレーションIdPとしての最低限の機能は動作
- **2015年12月中旬～2月中旬**
  - 忙しいのだが時々作業をして今回話すネタ等の問題を１つずつクリア
- **2016年2月中旬**
  - IdP3化した学認運用フェデレーションサーバをITC内部で試験公開
- **2016年2月中旬～4月中旬**
  - 本気で忙しかったのでしばらく放置
- **2016年4月下旬**
  - IdP3版学認運用フェデレーションサーバの公開、塾内IdPのIdP3化作業開始
- **2016年5月上旬**
  - 塾内システム用IdPとして基本的な機能が動作
- **実は正味の時間としては大したことない？**

今回のIdP3構築環境

- **CentOS 7**
- **Oracle JDK 8+JCE(Java Cryptography Extension)**
- **Tomcat 8**
- **Shibboleth IdP 3.2.1（2015年12月公開）**

- **備考**
  - wiki.shibboleth.netでの推奨環境に従った
  - OpenJDKの場合はJCEは不要
  - 構築手順の観点からは、CentOS 6やOpenJDK 1.7やTomcat 7を利用する場合でもあまり変わりない（試した）
    - attribute-*.xmlでスクリプトを使っていないので、Tomcatのバージョンへの依存は薄い

Shibboleth IdP ver.3との戦い

IdP2の設定、データはどの程度そのままIdP3に使える？

IdP2用設定はそのまま使える？

- **基本設定で大きな書き換えが必要「ない」のは attribute-filter.xmlとattribute-resolver.xml**
  - サイトごとに書きかえることが多いこの2つのファイルがほぼそのまま使えるので楽
- **uApprove-jpと相当機能のconsent-interceptとは全く設定が異なる**
  - それほど設定の手間は変わらない
- **FPSPと相当機能のcontext-checkとは全く設定が異なる**
  - とても記述が複雑になっている
- **画面・文言のカスタマイズ方法は全く変わっている**
  - 楽になった部分もある

IdP2用データはそのまま使える？

- **LDAPサーバはそのまま使える**
  - よかった…
- **StoredIDのDBはそのままでは使えない**
  - 対応は難しくない
- **uApprove-jpのDBはconsent-interceptに単純には引き継げない**
  - 不可能ではなさげですが、かなり面倒そうなので諦めました

IdP3でハマったポイント(1)

StoredID関連

既存のStoredIDのDBに接続すると…

- **エラーが出てIdPが起動しない**
  - java.sql.SQLException: Duplicate insertion succeeded, primary key missing from table
- **3.2.0からテーブルの仕様が変更になった**
  - localEntity, peerEntity, persistentIDをプライマリキー指定する必要がある
- **既存のDBを使う必要がある場合**
  - 制約を追加する必要がある
  - 一度もStoredIDを手動で変更したことがなければ、新たに空のDBを作ってもいい

shibpidテーブルの定義例（MySQL）

- 学認サイトの例

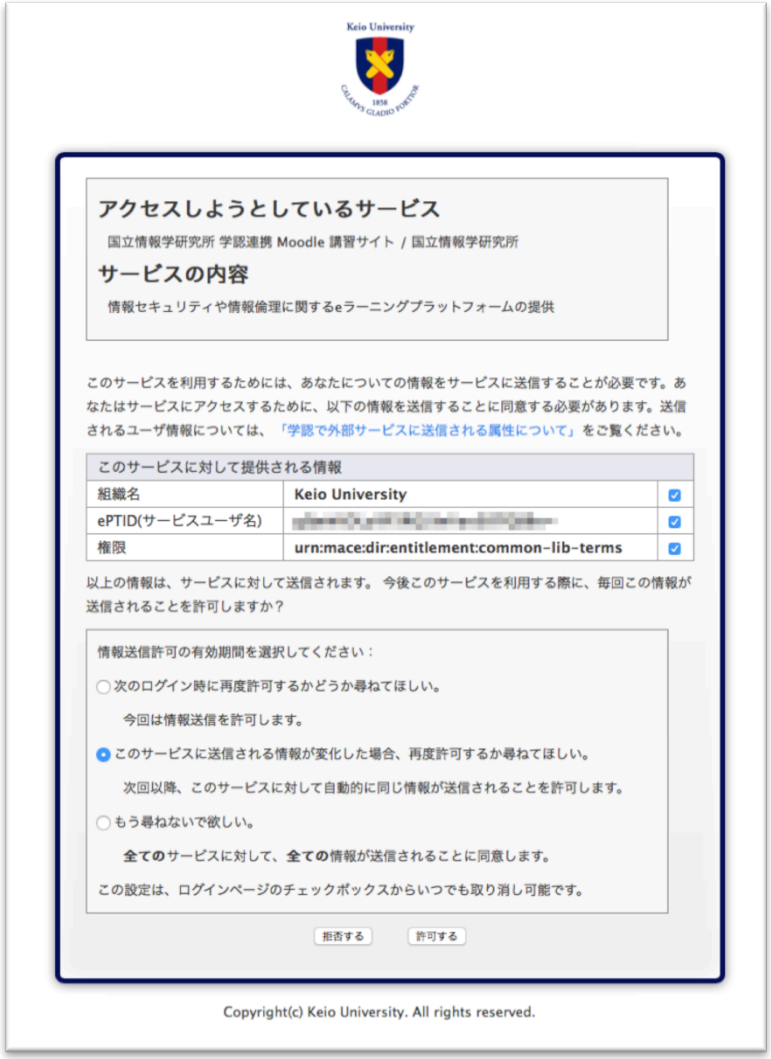
```
CREATE TABLE shibpid (
    localEntity VARCHAR(255) NOT NULL,
    peerEntity VARCHAR(255) NOT NULL,
    persistentId VARCHAR(50) NOT NULL,
    principalName VARCHAR(50) NOT NULL,
    localId VARCHAR(50) NOT NULL,
    peerProvidedId VARCHAR(50) NULL,
    creationDate TIMESTAMP NOT NULL,
    deactivationDate TIMESTAMP NULL,
    PRIMARY KEY (localEntity, peerEntity, persistentId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

IdP3でハマったポイント(2)

uApprove-jpとconsent-intercept

consent-interceptとは？

- **uApprove-jp相当の機能がIdP3で標準サポートされたもの**
  - SP毎にユーザの属性送信に対してユーザに許可を取ることができる
  - 利用規約に関して同意を取ることもできる
  - 細かいところは色々違う…
  - デフォルトで有効



consent-interceptの設定

- **relying-party.xml中の、@p:postAuthenticationFlowsに指定する**

属性送信のみ承認を得る場合のrelying-party.xml（デフォルト）

```
<bean parent="Shibboleth.SSO" p:postAuthenticationFlows="attribute-release" />
...(略)...
<bean parent="SAML2.SSO" p:postAuthenticationFlows="attribute-release" />
```

属性送信と利用規約の承認を得る場合のrelying-party.xml

```
<bean parent="Shibboleth.SSO"
      p:postAuthenticationFlows="#{ {'terms-of-use', 'attribute-release'} }" />
...(略)...
<bean parent="SAML2.SSO"
      p:postAuthenticationFlows="#{ {'terms-of-use', 'attribute-release'} }" />
```

- **属性の表示名はuApprove-jpと同様に、attribute-resolver.xml中のresolver:DisplayNameが利用可能**

利用規約に関する設定

利用規約の承認に関するconf/intercept/consent-intercept-config.xml
の設定部分

```
<alias alias="shibboleth.consent.terms-of-use.Key" name="shibboleth.SingleTermsOfUseText" />

<bean id="shibboleth.SingleTermsOfUseText"
      class="com.google.common.base.Functions" factory-method="constant">
    <constructor-arg value="my-terms-of-use-0" />
</bean>
```

mesages/consent-messages.properties中に利用規約自体を定義する

```
my-terms-of-use-0 = my-tou-0
my-tou-0.title = 慶應義塾大学学術認証フェデレーションシステム利用規則
my-tou-0.text = <p>\
（目的）<br/> \
第１条<br/> \
本規則は、慶應義塾大学学術認証フェデレーションシステムの利用に関する事項を定めることにより、慶應義塾大学における\
情報セキュリティの確保と学認対応サービスの円滑な利用に資することを目的とする。<br/> \
....(略)....
```

利用規約が改定された場合は、別の名前（my-terms-of-use-0の代わり）
で定義を行えば、再度利用規約への同意をユーザに求めることができる。

送信確認画面での属性の表示順指定

- conf/intercept/consent-intercept-config.xml で指定（3.2.1以降）

```
<util:list id="shibboleth.consent.attribute-release.AttributeDisplayOrder">
   <value>givenName</value>
   <value>surname</value>
   <value>displayName</value>
   <value>jaSurname</value>
   <value>jaGivenName</value>
   <value>jaDisplayName</value>
   <value>mail</value>
   <value>gakuninScopedPersonalUniqueCode</value>
   <value>eduPersonPrincipalName</value>
   <value>eduPersonAffiliation</value>
   <value>eduPersonScopedAffiliation</value>
   <value>organizationName</value>
   <value>jaOrganizationName</value>
   <value>eduPersonTargetedID</value>
   <value>eduPersonEntitlement</value>
</util:list>
```

ユーザ承認データの格納場所

- **標準ではクライアント側（特別な設定不要）**
  - Cookie
  - HTML5 Web Storage (3.2.0以降)

- **簡単でいいけど、本当にそれでいいの？**
  - 別のマシンやブラウザからアクセスすると、再承認が要求される
  - 属性送信承認や利用規約同意の情報がサーバ側に残らない

承認結果をサーバ側DBに格納

- **JPA(Java Persistence API)を利用**
  - バックエンドにMySQLを利用可能
  - Tomcat7と8では多少設定が異なる
- **uApprove-jpのDBとは全く互換性がない**
  - 許可内容はJSON的な形式で”value”に設定される

```
CREATE TABLE StorageRecords (
  context VARCHAR(255) NOT NULL,
  id VARCHAR(255) NOT NULL,
  expires BIGINT(20) DEFAULT NULL,
  value LONGTEXT NOT NULL,
  version BIGINT(20) NOT NULL,
  PRIMARY KEY (context, id)
);
```

global.xml設定例(Tomcat8)

```
<bean id="shibboleth.JPAStorageService"
  class="org.opensaml.storage.impl.JPAStorageService"
  p:cleanupInterval="%{idp.storage.cleanupInterval:PT10M}"
  c:factory-ref="shibboleth.JPAStorageService.EntityManagerFactory" />
<bean id="shibboleth.JPAStorageService.EntityManagerFactory"
  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="storageservice" />
  <property name="packagesToScan" value="org.opensaml.storage.impl" />
  <property name="dataSource" ref="shibboleth.JPAStorageService.DataSource" />
  <property name="jpaVendorAdapter" ref="shibboleth.JPAStorageService.JPAVendorAdapter" />
  <property name="jpaDialect">
    <bean class="org.springframework.orm.jpa.vendor.HibernateJpaDialect" />
  </property>
</bean>
<bean id="shibboleth.JPAStorageService.JPAVendorAdapter"
  class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
  <property name="database" value="MYSQL" />
</bean>
<bean id="shibboleth.JPAStorageService.DataSource"
  class="org.apache.tomcat.dbcp.dbcp2.BasicDataSource"
  p:driverClassName="com.mysql.jdbc.Driver"
  p:url="jdbc:mysql://DBのIPアドレス:3306/DB名"
  p:username="DBユーザ名"
  p:password="DBパスワード"
  p:maxIdle="5"
  p:maxTotal="10"
  p:maxWaitMillis="15000"
  p:testOnBorrow="true"
  p:validationQuery="select 1"
  p:validationQueryTimeout="5" />
```

コネクションプールとして、
Commons DBCPを利用する例。
その他にもTomcat JDBC Pool,
BoneCP, HikariCPなどが利用可能。

Tomcat7:
`class="org.apache.tomcat.dbcp.dbcp.BasicDataSource"`

Tomcat7:
`p:maxActive="10"`
`p:maxWait="15000"`

idp.properties設定例

consentのバックエンドを標準のクライアントサイドではなくJPAに設定

```
# Set to "shibboleth.StorageService" or custom bean for alternate storage of consent
#idp.consent.StorageService = shibboleth.ClientPersistentStorageService
idp.consent.StorageService = shibboleth.JPAStorageService
```

承認を記録するキーをuidではなくePPNに設定する場合（IdP 3.2.0以降）

```
# Set to "shibboleth.consent.AttributeConsentStorageKey" to use an attribute
# to key user consent storage records (and set the attribute name)
#idp.consent.userStorageKey = shibboleth.consent.PrincipalConsentStorageKey
idp.consent.userStorageKey = shibboleth.consent.AttributeConsentStorageKey
idp.consent.userStorageKeyAttribute = eduPersonPrincipalName
```

IdP3でハマったポイント(3)

FPSPとcontext-check

context-checkとは？

- **FPSP(Filter Per SP)相当の機能がIdP3で標準サポートされたもの**
  - 属性の値に応じてSPへの属性送信許可を制御できる
  - 利用例
    - 学生にのみDreamSparkを利用させたい
    - 学生・教職員にのみBoxを利用させたい

context-checkの設定

- **relying-party.xmlの設定部分は簡単**
  - @p:postAuthenticationFlowsに設定する

```
<bean parent="Shibboleth.SSO"
      p:postAuthenticationFlows=
           "#{ {'context-check', 'terms-of-use', 'attribute-release'} }" />
...(略)...
<bean parent="SAML2.SSO"
      p:postAuthenticationFlows=
           "#{ {'context-check', 'terms-of-use', 'attribute-release'} }" />
```

この例では、利用規約（terms-of-use）と属性送信許可(attribute-release)も同時に指定している

context-checkの条件定義が複雑

- **2つの方法**
  1. context-checkを適用するSPを限定して、それらのSPに対する条件のみを適用する
  2. 全SPに対してcontext-checkを適用する（前スライドの定義）

- **それぞれの欠点**
  - 1.は認証方式として特殊な扱いをする例外SPのリストにcontext-check対象を加えるのだが、設定が2つのファイルにまたがるなどの理由でわかりにくい
  - 2.は1つのファイルでSP毎の設定が完結するのだが、普通に設定すると対象SPのentityIDを設定中の2箇所に書く必要がありこれはこれでわかりにくい
    - 今回は2.の例を紹介する

SP全体をcontext-check対象とする

- 「対象SPのentityIDを設定中の2箇所に書く必要がある」意味

```
entityID in (“SP-A”) and attribute1 in (“X”, “Y”) or
entityID in (“SP-B”) and attribute1 in (“Z”) or
not (entityID in (“SP-A”, “SP-B”))
```

context-checkのロジックの指定法を仮想的なコードで記述した例
（本当はXMLで記述するためもっと記述が複雑）

SP-AとSP-BのentityIDを2箇所に書く必要があり、わかりにくい

否定ロジックで簡略化？

- 「ダメな条件」全体の否定を書くことでSPのentityIDを書く場所を1箇所に
  - やや直感的なわかりやすさが損なわれる？

```
not (entityID in (“SP-A”) and not attribute1 in (“X”, “Y”) or
     entityID in (“SP-B”) and not attribute1 in (“Z”))
```

参考：実際の記述例

- **前スライドの論理をXMLで記述した例**
  - conf/intercept/context-check-intercept-config.xml
  - 実は元の論理を記述するほうが短いのだが、SPのentityID比較が2箇所になるので微妙…
- **これは直感的に読めない！**
  - FPSPは簡単だったのに…
  - コンパイラ的なものが欲しい…

```
not (entityID in (“SP-A”) and
       not attribute1 in (“X”, “Y”) or
     entityID in (“SP-B”) and
       not attribute1 in (“Z”))
```

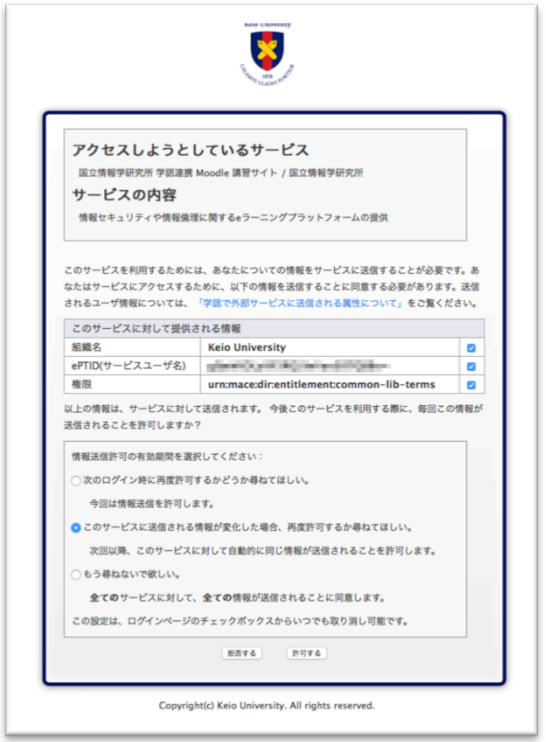
```
<bean id="shibboleth.context-check.Condition" parent="shibboleth.Conditions.NOT">
  <constructor-arg>
    <list>
      <bean id="shibboleth.context-check.Condition" parent="shibboleth.Conditions.OR">
        <constructor-arg>
          <list>
            <bean id="shibboleth.context-check.Condition" parent="shibboleth.Conditions.AND">
              <constructor-arg>
                <list>
                  <bean parent="shibboleth.Conditions.RelyingPartyId" c:candidates="#{ 'SP-A' }" />
                  <bean id="shibboleth.context-check.Condition" parent="shibboleth.Conditions.NOT">
                    <constructor-arg>
                      <list>
                            <bean class="net.shibboleth.idp.profile.logic.SimpleAttributePredicate"
                              p:useUnfilteredAttributes="true">
                          <property name="attributeValueMap">
                            <map>
                              <entry key="attribute1">
                                <list>
                                  <value>X</value>
                                  <value>Y</value>
                                </list>
                              </entry>
                            </map>
                          </property>
                        </bean>
                      </list>
                    </constructor-arg>
                  </bean>
                </list>
              </constructor-arg>
            </bean>
            <bean id="shibboleth.context-check.Condition" parent="shibboleth.Conditions.AND">
              <constructor-arg>
                <list>
                  <bean parent="shibboleth.Conditions.RelyingPartyId" c:candidates="#{ 'SP-B' }" />
                  <bean id="shibboleth.context-check.Condition" parent="shibboleth.Conditions.NOT">
                    <constructor-arg>
                      <list>
                        <bean class="net.shibboleth.idp.profile.logic.SimpleAttributePredicate"
                              p:useUnfilteredAttributes="true">
                          <property name="attributeValueMap">
                            <map>
                              <entry key="attribute1">
                                <list>
                                  <value>Z</value>
                                </list>
                              </entry>
                            </map>
                          </property>
                        </bean>
                      </list>
                    </constructor-arg>
                  </bean>
                </list>
              </constructor-arg>
            </bean>
          </list>
        </constructor-arg>
      </bean>
    </list>
  </constructor-arg>
</bean>
```

IdP3でハマったポイント(4)

画面カスタマイズ・多言語化

ログイン画面等のカスタマイズ

- IdP2: JSPの書き換え
- IdP3: Velocityテンプレートの書き換え



画面全体の構成はviews/で設定

- **例：views/login.vmの一部**
  - #springMessageText(): Spring FrameworkによるJMS(Java Message Service)を用いたメッセージ置換
  - $変数名.メソッド名():メソッドを実行して結果に置換

```
<dt>#springMessageText("idp.login.username", "Username")</dt>
<dd id="box_iptName"><input type="text" name="j_username" id="iptName”
     class="ipt_text_m" autocapitalize="off" autocorrect="off" />
  <img id="mark_help_name"
       src="$request.getContextPath()/common/images/mark_help01.jpg"
       onclick="openFolder('box_help_name', 'mark_help_name')" alt="" />
  <div id="box_help_name" class="box_help">
    #springMessageText("idp.keio.usernameDescription", "Username")
  </div>
</dd>
```

- **サンプルを見れば勘でなんとかなるレベル**
- **vmファイルを書き換えると即時反映される**
  - js, css, 画像ファイルなどの追加・更新は、idp.warの再構築・コピーが必要

メッセージカタログはmessages/で設定

- **多言語化対応も可能**
  - その場合、例えばデフォルトのmessages/authn-message.propertiesは、messages/authn-message_en.propertiesという名前のファイルにコピー

messages/
authn-messages_en.properties
（英語・コピーする）

```
idp.login.login = Login
idp.login.pleasewait = Logging in, please wait...
idp.login.forgotPassword = Forgot your password?
idp.login.needHelp = Need Help?
```

messages/
authn-messages_ja.properties
（日本語・追加する）

```
idp.login.login = ログイン
idp.login.pleasewait = ログインしています。お待ち下さい。
idp.login.forgotPassword = パスワードを忘れた場合
idp.login.needHelp = ヘルプ
```

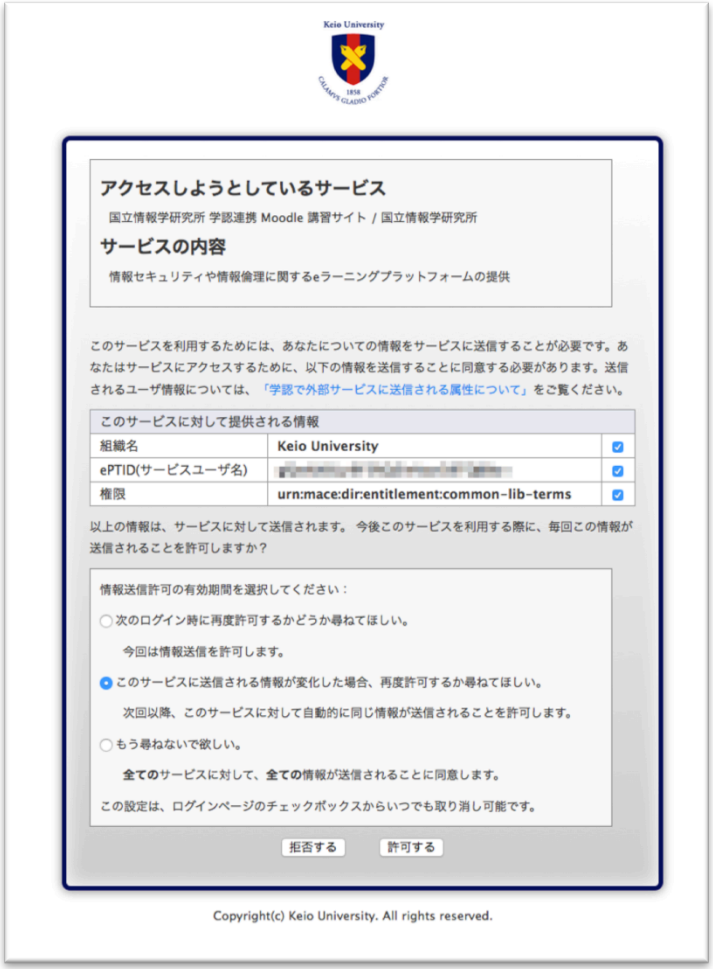
- **独自のメッセージを追加することも可能**
  - views/からの参照は標準メッセージと同様に#springMessageText()で可能

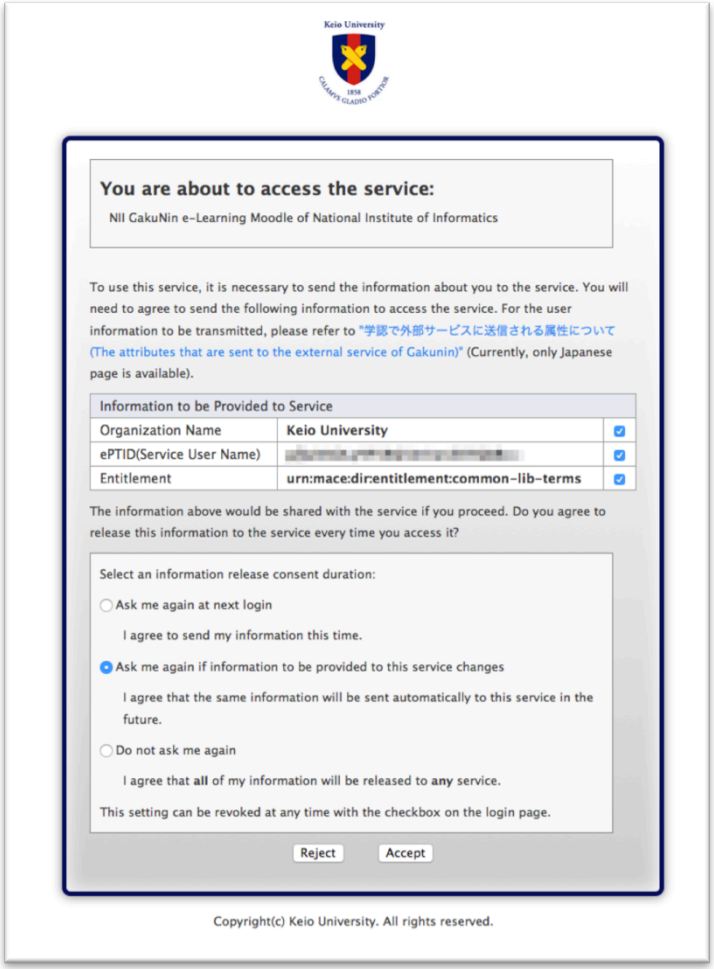
```
idp.keio.pagetitle = 慶應義塾大学学術認証フェデレーションシステム
```

- **書き換え後はTomcat再起動が必要**

多言語化の対応例

- ブラウザの言語設定で切り替わる





参考：ログインエラーメッセージ

- **標準ではユーザ名間違いとパスワード間違いで別のエラーメッセージが出てしまう**
  - 同じメッセージが出るように簡単に変更できる
    - _enや_jaのファイルの変更も忘れずに

オリジナルのmessages/authn-messages.propertiesの一部

```
UnknownUsername = bad-username
InvalidPassword = bad-password
... (略) ...

bad-username.message = The username you entered cannot be identified.

bad-password.message = The password you entered was incorrect.
```

変更したmessages/authn-messages.propertiesの例

```
UnknownUsername = bad-username-or-password
InvalidPassword = bad-username-or-password
... (略) ...

bad-username-or-password.message = Username or password was incorrect.
```

IdP3でハマったポイント(5)

学認以外のサービス用IdP設定

メタデータ定義の置き場所

- **学認以外のSPに認証を提供している場合**
  - SPメタデータの定義場所が変更になった点に注意
    - IdP2: relying-party.xml
    - IdP3: metadata-providers.xml
  - FileBackedHTTPMetadataProviderやその証明書などの標準的な定義法なども大きく変更になっているようなので注意
    - デフォルトの設定ファイルや学認メタデータのための設定例を参照(metadata-providers.xml)

非暗号化アサーションSPの定義法

- **Google Apps, Box等**
  - relying-party.xmlで定義

IdP2におけるGoogle AppsのためのRP定義

```
<rp:RelyingParty id="google.com" provider="IdPのentityIDを入れる"
    defaultSigningCredentialRef="IdPCredential">
  <rp:ProfileConfiguration xsi:type="saml:SAML2SSOProfile" encryptAssertions="never”
      encryptNameIds="never" />
  <rp:ProfileConfiguration xsi:type="saml:SAML2LogoutRequestProfile”
      signResponses="conditional" />
</rp:RelyingParty>
```

IdP3におけるGoogle AppsのためのRP Overrides定義

```
<util:list id="shibboleth.RelyingPartyOverrides">
  <bean parent="RelyingPartyByName" c:relyingPartyIds="google.com">
    <property name="profileConfigurations">
      <list>
        <bean parent="SAML2.SSO"
          p:nameIDFormatPrecedence="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified”
          p:encryptAssertions="false" p:postAuthenticationFlows="#{ {'context-check'} }"/>
      </list>
    </property>
  </bean>
</util:list>
```

Box等では@p:nameIDFormatPrecedenceは不要

@p:postAuthenticationFlowsには、他SPと同じFlowを記述しないと、
このSPに対してFlowが機能しなくなる（この例ではcontext-checkのみを指定）

注: Google Apps用のattribute-resolver.xmlのNameID定義は変更しなくてよい

Shibboleth IdP ver.3との戦い

その他

IdP冗長化の標準機能化

- **IdP2時代のStateless Clustering相当機能が標準で組み込まれた**
  - 従来はIdPをメンテナンスする際などにユーザに再認証を要求していたが、不要となった
  - わりと嬉しい

auditログにクライアントのIPを追加

- **IdP3で指定法が変わった**
  - logback.xml中のid="IDP_AUDIT"と id="IDP_CONSENT_AUDIT"の以下の部分の<Pattern/>を書き換えれば良い
  - クライアントのIPアドレスとJSESSIONIDが記録される

```
<encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
  <charset>UTF-8</charset>
  <!--
  <Pattern>%msg%n</Pattern>
  -->
  <Pattern>%msg%mdc{idp.remote_addr}|%mdc{idp.jsessionid}|%n</Pattern>
</encoder>
```

attribute-resolver.xmlの変更点

- **IdP3のサンプル設定は、myLDAPコネクタがldap.propertiesの記述を利用するようになっている**
  - 学認のサンプルは現状では昔のまま
  - 私は3.2.1のattribute-resolver-full.xmlをベースに、学認の属性定義をマージして利用
  - ログイン用のLDAP定義と属性取得用LDAP定義が一箇所で済むので好き
    - IdP3ではlogin.configは存在しない

attribute-filter.xmlの変更点

- **3.2.0で付属サンプルのスキーマが変更となった**
  - 以前のスタイルそのままでも動くので、切り替えなくても問題ない
    - 現状では学認テンプレートも旧スタイルのまま
  - 切り替えたら、学認SP一覧のサンプルをコピペできなくなった

旧スタイル例

```
<AttributeFilterPolicy id="SamplePolicy">
  <PolicyRequirementRule xsi:type="basic:AttributeRequesterString"
    value="entityID-sample" />
  <AttributeRule attributeID="attribute1">
    <PermitValueRule xsi:type="basic:ANY" />
  </AttributeRule>
  <AttributeRule attributeID="attribute2">
    <PermitValueRule xsi:type="basic:AttributeValueString" value="文字列"/>
  </AttributeRule>
</AttributeFilterPolicy>
```

新スタイル例

```
<AttributeFilterPolicy id="SamplePolicy">
  <PolicyRequirementRule xsi:type="Requester" value="entityID-sample" />
  <AttributeRule attributeID="eduPersonPrincipalName">
    <PermitValueRule xsi:type="ANY" />
  </AttributeRule>
  <AttributeRule attributeID="attribute2">
    <PermitValueRule xsi:type="Value" value="文字列"/>
  </AttributeRule>
</AttributeFilterPolicy>
```

Shibboleth IdP ver.3との戦い

まとめ

結論

- **IdP2からIdP3への移行にはハマる場所が多くある**
  - …が、ある程度サンプルや勘所のガイドがあればそれほど難しくはない

IdP3対応は（思ったほど）怖くない！

<参考> 今回の内容のベースとなったブログ
「学認IdPをShibboleth3にする際のメモ」
http://memo.itc.keio.ac.jp/blog/?p=474

ご静聴ありがとうございました。