多要素認証 CAS サービス
利用マニュアル

第 1.1 版 2022 年 2 月 3 日

情報連携推進本部

変更履歴

版数	日付	内容
第 1.0 版	2021 年 5 月 10 日	新規作成
第 1.1 版	2022 年 2 月 3 日	誤字修正

目次

1. 多要素認証 CAS サービスとは

1.1. 背景

現在、名古屋大学の情報サービスにおいては、CAS と呼ばれるシングルサインオン環境のもと、名古屋大学 ID とパスワードによって認証を行い様々な情報サービスが利用できます。

しかし、近年では ID とパスワードのみによる認証は、セキュリティの観点から不十分とされており、複数の要素で本人確認を行うことが求められています。

そこで、新たな CAS では、名古屋大学 ID とパスワードに加え、OATH-TOTP 規格に従い、各自のスマートフォンや PC、情報推進本部から貸し出すハードウェアドングルを要素とした認証をできるようにし、2021 年 7 月にサービスを開始しました。

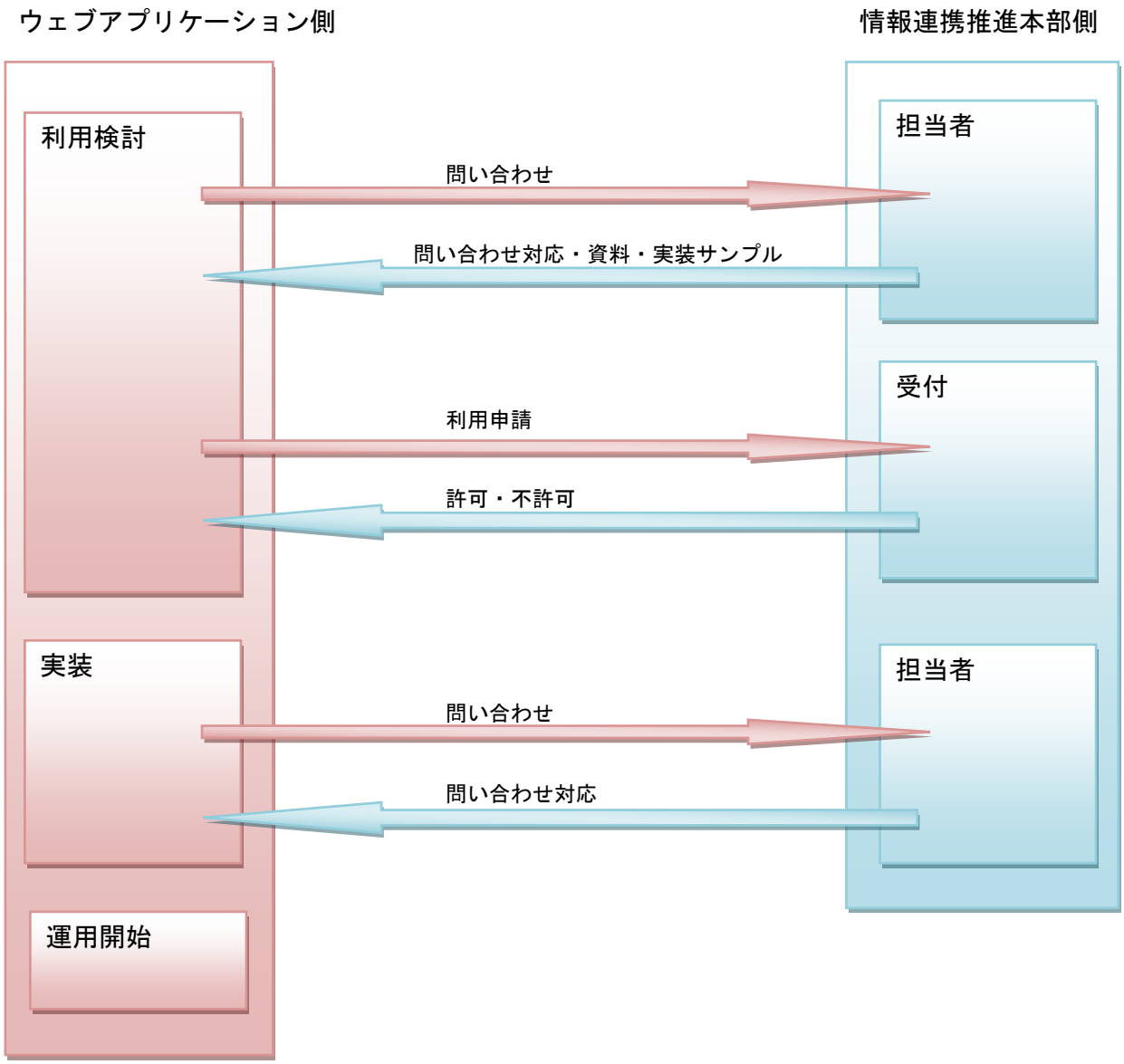
1.2. 多要素認証 CAS サービスでできること

多要素認証 CAS サービスを利用するウェブアプリケーションは次のことを実現できます。

- 名古屋大学 ID とパスワードによる認証
- 名古屋大学離籍者は自動でログインを制限
- 許可したロールをもつユーザのみにログインを制限
- 多要素認証 CAS サービスを利用するウェブアプリケーション同士のシングルサインオン
- ログインユーザの属性情報（名古屋大学 ID、氏名、所属、身分、ロール情報など）の取得と利用
- ウェブアプリケーション側でロール情報と権限を紐づけることによる権限管理

多要素認証 CAS サービス利用までの流れ

多要素認証 CAS サービスを利用するためには、利用申請とウェブアプリケーション側の実装が必要です。次に利用までの流れを示します。



利用申請時に次の項目を指定します。

- ウェブアプリケーションの URL
- ウェブアプリケーションへのログインを許可するロール
- ウェブアプリケーションが利用する、ユーザの属性情報
- シングルサインオンの許可、不許可
- 離籍者の認証許可、不許可

1.3. 制限事項

ウェブアプリケーションは、認証サーバと HTTPS で通信できる必要があります。

1.4. ロールとは

多要素認証 CAS サービスでは、ユーザの身分をロールとして定義します。
なお、身分は、以下の 10 個のうちユーザがどれに該当するかで表されます。
また、ユーザは複数の身分を持つことがあります。

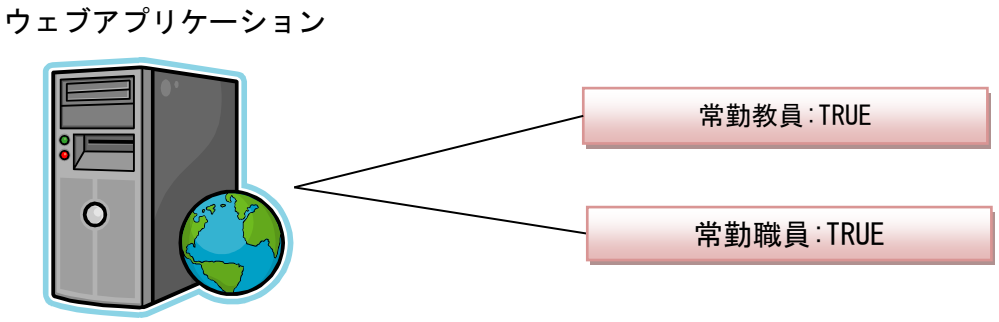
roleStudentFulltime	正規学生
roleStudentParttime	非正規学生
roleProfFulltime	常勤教員
roleProfParttime	非常勤教員
roleStaffFulltime	常勤職員
roleStaffParttime	非常勤職員
roleExecutiveFulltime	常勤役員
roleExecutiveParttime	非常勤役員
roleTeacher	教諭
roleProfEmeritus	名誉教授

1.5. ロールによる認証の仕組み

多要素認証 CAS サービスを利用するウェブアプリケーションには、ログインを許可するロールを設定します。

ウェブアプリケーションにログインできる人は、ウェブアプリケーションに許可したロールに該当する人です。

例えば、ウェブアプリケーションに常勤職員と常勤教員に対して利用許可をする場合は、次の図のように、ロールを設定します。



ウェブアプリケーションが権限によって動作を変えたい場合、多要素認証 CAS サーバから受け取ることができるログインした人のロールの真偽値(TRUE または FALSE)を使用できます。

例えば、上図を例にすると、ログインした人の常勤教員属性が TRUE の場合は、教員用の権限で動作させ、ログインした人の常勤職員属性が TRUE の場合は、職員権限で動作させるように実装します。

2. CAS 認証メカニズム

2.1. CAS 認証メカニズムを理解する必要性

多要素認証 CAS サービスは、CentralAuthenticationService（CAS）を用いて実現しています。よって多要素認証 CAS サービスをウェブアプリケーションが実装するには、CAS 認証メカニズムの理解が必要です。また、認証でエラーになる場合の調査においても、CAS 認証メカニズムを理解していることが解決に役立ちます。

2.2. ウェブアプリケーションにユーザが初めてアクセスする場合

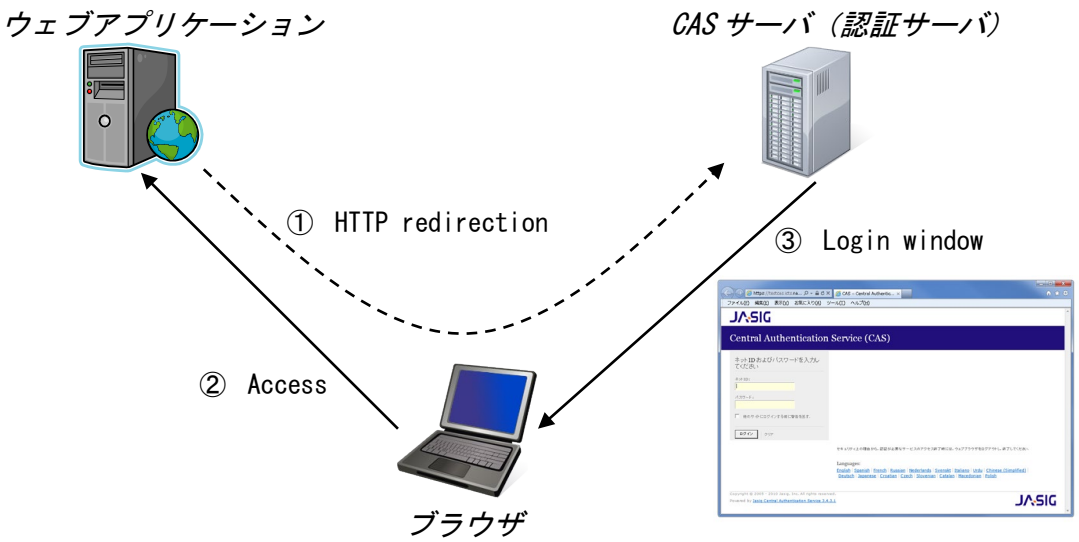
2.2.1. CAS 認証その 1

ユーザは Web ブラウザで、利用するウェブアプリケーションの URL（例えば、http://test.jp/index）にアクセスします（図の①）。

ウェブアプリケーション側は CAS サーバへ HTTP リダイレクション機能を使って、アクセスを転送します（図の②）。その際、service パラメータを用いて、認証すべきサービスの URL を伝えます。（例：https://auth-mfa.nagoya-u.ac.jp/cas/login?service=http://test.jp/index）

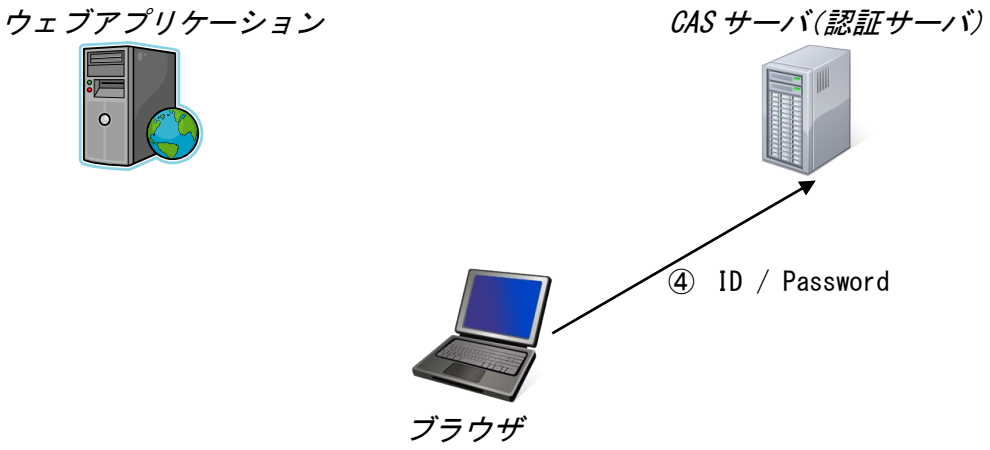
CAS サーバは、ブラウザに保存されている TicketGrantingCookie（TGC）を確認し、TGC がない場合は、CAS 認証がまだ終わっていないと判断し、認証画面を表示します（図の③）。

なお、service パラメータに設定する URL は、URL エンコードする必要があります。



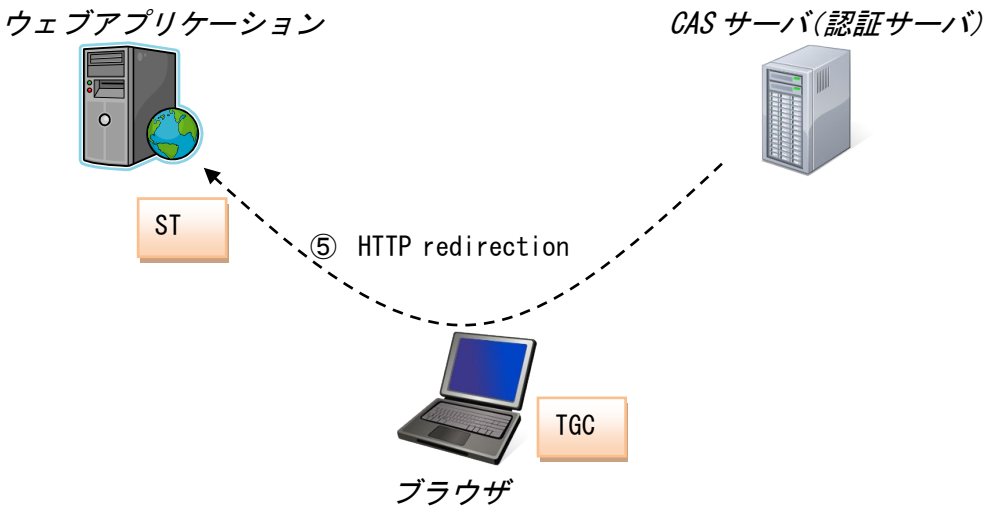
2.2.2. CAS 認証その２

ユーザは認証画面で、名古屋大学 ID とパスワードを入力し送信します（図の④）。



2.2.3. CAS 認証その３

ユーザが正しく認証されると、CAS サーバはブラウザに対し TGC を発行するとともに、URL パラメータ ticket に、ServiceTicket（ST）をセットし、再度呼び出されたアウェブアプリケーションへ HTTP リダイレクトをします（図の⑤）。（例：http://test.jp/index?ticket=ST-668-ySrjgIgdRiOVeTWdO7OdZzIwzlOtiz9ZO9vtbKcXuAreVhzH90-cas）

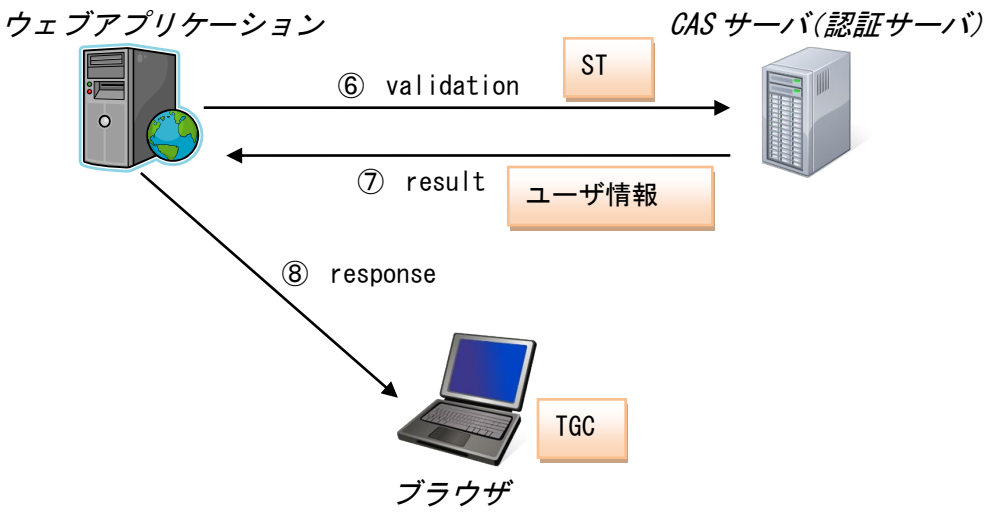


2.2.1. CAS 認証その４

ウェブアプリケーションは取得した ST を検証するため、CAS サーバに対して ST を送信します（図の⑥）。（例：https://auth-mfa.nagoya-u.ac.jp/cas/serviceValidate?service=http://test.jp/index&ticket=ST-668-ySrjgIgdRiOVeTWdO7OdZzIwzlOtiz9ZO9vtbKcXuAreVhzH9O-cas）

CAS サーバでは ST の有効性を検証し、その結果をウェブアプリケーションに送信します（図の⑦）。その際、ログインユーザの属性情報（名古屋大学 ID、氏名、所属、身分、ロール情報など）を XML 形式で送信します。XML の形式については「2.5CAS サーバが返す情報」に示します。

ウェブアプリケーションは CAS サーバからの情報に基づいて、ユーザにサービスを提供します（図の⑧）。

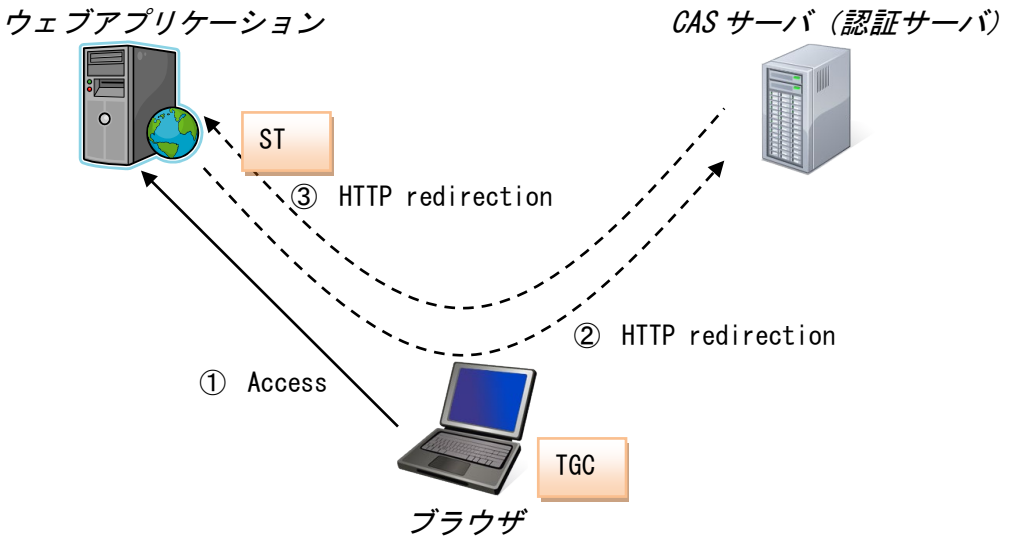


2.3. シングルサインオンでユーザがアクセスする場合

ユーザは Web ブラウザで、利用するウェブアプリケーションの URL にアクセスします（図の①）。

ウェブアプリケーション側は CAS サーバへ HTTP リダイレクション機能を使って、アクセスを転送します（図の②）。その際、service パラメータを用いて、認証すべきサービスの URL を伝えます。

CAS サーバは、ブラウザに保存されている TGC を確認し、TGC があるため認証画面を表示せず、URL パラメータ ticket に、ST をセットし、再度呼び出されたウェブアプリケーションへ HTTP リダイレクトをします（図の③）。



その後の動作は、「2.2.1 CAS 認証その４」と同様です。

2.4. ログアウトする場合

ブラウザが、多要素認証 CAS サーバのログアウト用 URL（https://auth-mfa.nagoya-u.ac.jp/cas/logout）にアクセスすると、多要素認証 CAS サービスからログアウトします。

また、service パラメータに、URL を指定すると、多要素認証 CAS サービスからログアウトした後、指定した URL にリダイレクトされます。（例：https://auth-mfa.nagoya-u.ac.jp/cas/logout?service=http://test.jp/logout）

なお、リダイレクト可能な URL は、多要素認証 CAS サービスを使用している URL のみです。

2.5. CAS サーバが返す情報

2.5.1. ST 検証成功の場合

XML の要素を次に示します。

XML 要素	複数	省略可	内容
cas:serviceResponse			
cas:authenticationSuccess			
cas:user			ユーザ ID。名古屋大学 ID
cas:attributes			属性情報
cas:XXXX（XXXX は属性名）	○	○	NagoyaUnivID など申請した属性を列挙する **※申請した属性名にセミコロン「;」を含む場合は、アンダーバー2 つ「_」に変換しています**
cas:roleStudentFulltime		○	TRUE/FALSE
cas:roleStudentParttime		○	TRUE/FALSE
cas:roleProfFulltime		○	TRUE/FALSE
cas:roleProfParttime		○	TRUE/FALSE
cas:roleStaffFulltime		○	TRUE/FALSE
cas:roleStaffParttime		○	TRUE/FALSE
cas:roleExecutiveFulltime		○	TRUE/FALSE
cas:roleExecutiveParttime		○	TRUE/FALSE
cas:roleTeacher		○	TRUE/FALSE
cas:roleProfEmeritus		○	TRUE/FALSE

XML の例を次に示します。

```
<cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>
 <cas:authenticationSuccess>
  <cas:user>zz0000000</cas:user>
  <cas:attributes>
   <cas:NagoyaUnivID>zz0000000</cas:NagoyaUnivID>
   <cas:fullName__lang-ja>名大 太郎</cas:fullName__lang-ja>
   <cas:departmentNumber>190004003000</cas: departmentNumber >
   <cas:department__lang-ja>情報推進部情報基盤課</cas:department__lang-ja>
   <cas:enrollment>T</cas:enrollment>
   <cas:employeeTypeCode>01</cas:employeeTypeCode>
   <cas:employeeType__lang-ja>事務職員</cas:employeeType__lang-ja>
   <cas:roleStudentFulltime>TRUE</cas:roleStudentFulltime>
   <cas:roleTeacher>FALSE</cas:roleTeacher>
  </cas:attributes>
</cas:authenticationSuccess>
</cas:serviceResponse>
```

2.5.2. ST 検証失敗の場合

XML の要素を次に示します。

XML 要素	複数	省略可	内容
cas:serviceResponse			
cas:authenticationFailure			失敗の理由

XML の例を次に示します。

```
<cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>
    <cas:authenticationFailure code="INVALID_TICKET">
        Ticket ST-1856339-aA5Yuvrxzpv8Tau1cYQ7 not recognized
    </cas:authenticationFailure>
</cas:serviceResponse>
```

2.5.3. 申請可能な属性情報

属性と属性名を示します。値の範囲やコードについては、情報連携推進本部のウェブページ「http://www.icts.nagoya-u.ac.jp/ja/info/nuid.html」を参照してください。

区分	属性		属性名
学生・職員共通	名古屋大学 ID		NagoyaUnivID
	ミドルネーム	（ベース属性）	（middleName）
		漢字	middleName;lang-ja
		ローマ字	middleName;lang-en
		カタカナ	middleName;lang-ja;phonetic
	名	（ベース属性）	（givenName）
		漢字	givenName;lang-ja
		ローマ字	givenName;lang-en
		カタカナ	givenName;lang-ja;phonetic
	姓名	（ベース属性）	fullName
		漢字	fullName;lang-ja
		ローマ字	fullName;lang-en
		カタカナ	fullName;lang-ja;phonetic
	プライマリの所属部局	（ベース属性）	（department）
		漢字	department;lang-ja
		コード	departmentNumber
	プライマリの所属学科(専攻)，掛	（ベース属性）	（section）
		漢字	section;lang-ja
		コード	sectionNumber
	学生番号，職員番号，あるいは学務システム用番号		employeeNumber
	在学・在職中か否かを表すフラグ		Enrollment
学生のみ	入学年度		nagAdmissionYear
	学年		nagGrade
職員のみ	プライマリの職名	（ベース属性）	（title）
		漢字	title;lang-ja
		コード	titleCode
	プライマリの職種	（ベース属性）	（employeeType）
		漢字	employeeType;lang-ja
		コード	employeeTypeCode
	性別		Gender
学生・職員共通	正規学生ロール		roleStudentFulltime
	非正規学生ロール		roleStudentParttime
	常勤教員ロール		roleProfFulltime
	非常勤教員ロール		roleProfParttime
	常勤職員ロール		roleStaffFulltime
	非常勤職員ロール		roleStaffParttime
	常勤役員ロール		roleExecutiveFulltime
	非常勤役員ロール		roleExecutiveParttime
	教諭ロール		roleTeacher
	名誉教授ロール		roleProfEmeritus

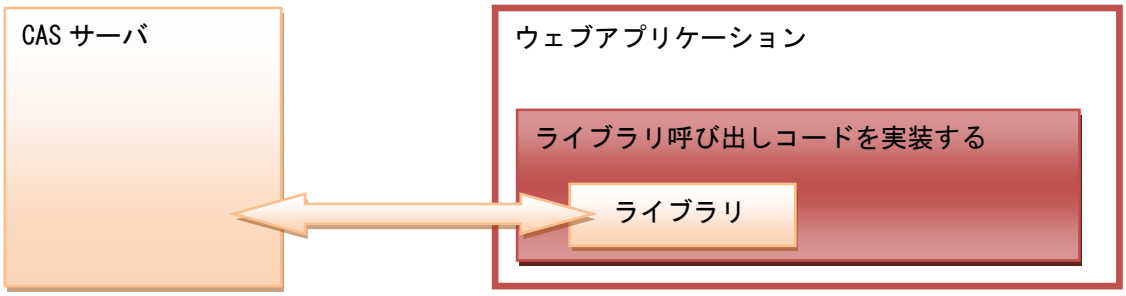
※属性名に括弧がついたベース属性は取得できません。

3. CAS 認証の実装方法

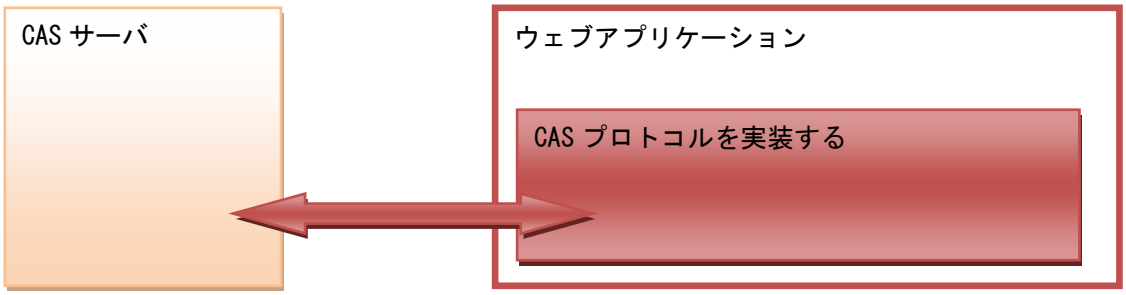
ウェブアプリケーションに CAS 認証を実装する場合、クライアント用ライブラリを利用する方法と、CAS プロトコルを実装する方法があります。

次に実装する範囲の概要を示します。

クライアント用ライブラリを利用する場合



CAS プロトコルを実装する場合



3.1. Java 用のライブラリを使用する場合

3.1.1. 概要

Java 用のライブラリは、サーブレットのフィルターとして使用します。

必要なライブラリをプロジェクトに入れ、web.xml ファイルにフィルターを設定します。

以下に JAVA での多要素認証 CAS クライアント構築の例を示します。詳細に関してはライブラリの公式ページ「GitHub - apereo/java-cas-client: Apereo Java CAS Client」をご覧ください。

3.1.1. 必要なライブラリ

CAS のクライアント用公式ライブラリ「https://github.com/apereo/java-cas-client」を使用します。ライブラリで使用するのは「cas-client-core」の jar ファイルで、上記ライブラリの git より clone してビルドするか、以下リンクの MavenRepository から取得します。

「https://mvnrepository.com/artifact/org.jasig.cas.client/cas-client-core」

また、「cas-client-core」で使用する機能の依存関係等、必要に応じて「commons-logging」などその他のライブラリも取得します。

3.1.2. フィルターとして使用するクラス

フィルターとして使用するクラスを次に示します。

これらは、フィルターとして web.xml で使う他、各ページにインポートする必要がありまっす。

順序	クラス名	内容
1	org.jasig.cas.client.authentication .AuthenticationFilter	CAS 認証を行う。
2	org.jasig.cas.client.validation .Cas20ProxyReceivingTicketValidationFilter	検証を行う。
3	org.jasig.cas.client.util .HttpServletRequestWrapperFilter	属性情報を request 情報から取得できるようにする。
4	org.jasig.cas.client.util .AssertionThreadLocalFilter	検証結果を ThreadLocal に保持する。
	org.jasig.cas.client.authentication .AttributePrincipal	ユーザ属性参照を行う web.xml では使われない。

3.1.1. web.xml の記述

web.xml に追加する記述を次に示します。また、ウェブアプリケーション毎に変更する箇所を赤字で示します。下記では申請したウェブアプリケーションの URL が「http://webapplication.jp/login」の場合を例にしています。

```
<filter>
  <filter-name>CAS Authentication Filter</filter-name>
  <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>
  <init-param>
    <param-name>casServerLoginUrl</param-name>
    <param-value>https://auth-mfa.nagoya-u.ac.jp/cas/login</param-value>
  </init-param>
  <init-param>
    <param-name>serverName</param-name>
    <param-value>http://webapplication.jp/</param-value>
  </init-param>
</filter>

<filter>
```

CAS サーバのログイン URL

ウェブアプリケーションの URL

```
  <filter-name>CAS Validation Filter</filter-name>

<filter-class>org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter</filter-class>
  <init-param>
    <param-name>casServerUrlPrefix</param-name>
    <param-value>https://auth-mfa.nagoya-u.ac.jp/cas</param-value>
  </init-param>
  <init-param>
    <param-name>serverName</param-name>
    <param-value>http://webapplication.jp/</param-value>
  </init-param>
  <init-param>
    <param-name>redirectAfterValidation</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>useSession</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>authn_method</param-name>
    <param-value>mfa-duo</param-value>
  </init-param>
</filter>

<filter>
    <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
    <filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</filter-class>
</filter>

<filter>
  <filter-name>CAS Assertion Thread Local Filter</filter-name>
  <filter-class>org.jasig.cas.client.util.AssertionThreadLocalFilter</filter-class>
</filter>

  <filter-name>CAS Authentication Filter</filter-name>
  <url-pattern>/login</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>CAS Validation Filter</filter-name>
  <url-pattern>/login</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
  <url-pattern>/login</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>CAS Assertion Thread Local Filter</filter-name>
  <url-pattern>/login</url-pattern>
</filter-mapping>
<filter-mapping>
```

CAS サーバの URL

ウェブアプリケーションの URL

フィルターを適用する URL パターン

フィルターを適用する URL パターン

フィルターを適用する URL パターン

フィルターを適用する URL パターン

3.1.1. ソースコードの記述

フィルターを通過した時点で、CAS 認証が成功しています。認証で異常が発生した場合は、Exception となります。

多要素認証 CAS サーバが返す情報をウェブアプリケーションで利用する場合、CAS Assertion Thread Local Filter を使って取得する場合と、CAS HttpServletRequest Wrapper Filter を使って取得する場合があります。

CAS Assertion Thread Local Filter を用いた場合では、web 層セッションにアクセスできないリソースからも、参照が可能な Thread Local に属性が配置されます。

ログインユーザの属性情報を取得する（CAS Assertion Thread Local Filter）

```
Assertion assertion=AssertionHolder.getAssertion();
Map<String, Object> attributes = assertion.getPrincipal().getAttributes();
attributes.get("NagoyaUnivID");
```

ログインユーザの属性情報を取得する（CAS HttpServletRequest Wrapper Filter）

```
AttributePrincipal principal = (AttributePrincipal) request.getUserPrincipal();
final Map attributes = principal.getAttributes();
attributes.get("NagoyaUnivID");
```

3.1.2. ログアウト

ウェブアプリケーションのログアウト処理を行った後、多要素認証 CAS サーバのログアウト URL「https://auth-mfa.nagoya-u.ac.jp/cas/logout」にリダイレクトします。

多要素認証 CAS サーバのログアウト後、ウェブアプリケーションの画面を再度表示させたい場合、「https://auth-mfa.nagoya-u.ac.jp/cas/logout?service=http://test.jp/logout」の様に、service パラメータで URL を指定できます。

なお、リダイレクト可能な URL は、多要素認証 CAS サービスを使用している URL のみです。

3.2. PHP 用のライブラリを使用する場合

3.2.1. 概要

PHP 用のライブラリは、API として使用します。必要なライブラリをディレクトリに入れ、認証 API をコールします。

以下に PHP での多要素認証 CAS クライアント構築の例を示します。詳細に関してはライブラリの公式ページ「GitHub - apereo/phpCAS: Apereo PHP CAS Client」をご覧ください。

3.2.2. 必要なライブラリ

CAS 認証サービス用ライブラリ https://github.com/apereo/phpCAS を使用します。

認証を行いたいアプリケーションに、ライブラリから「CAS」「CAS.php」をインストールし配置します。

3.2.3. 認証の仕方

CAS ではクライアント-サーバ間での SSL 通信が強く推奨されており、クライアント側では CAS サーバの証明書を設定する必要があります。

JAVA の一般的な構成等では、サーバ内部の openssl 等の証明書が参照され、証明書を手動で入れる必要は無いようですが、本 PHP 用ライブラリでは、証明書を手動でインストールする必要な形式のため、以下のように設定します。

CAS への SSL 接続のため root サーバ証明書をインポートしてライブラリで読み込める形に変換します。

cd /path/to/app（設定が必要なアプリケーション上で作業を行います）

wget https://repository.secomtrust.net/SC-Root2/SCRoot2ca.cer

openssl x509 -inform der -in SCRoot2ca.cer -out SCRoot2ca.pem

この時、chmod または chown で実行ユーザが参照可能なよう権限設定が必要な場合があります。

次に、以下のように config.php を作成します。

```
<?php
// Full Hostname of your CAS Server
$cas_host = 'auth-mfa.nagoya-u.ac.jp';
// Context of the CAS Server
```

```
$cas_context = '/cas';
// Port of your CAS server. Normally for a https server it's 443
$cas_port = 443;
// Path to the ca chain that issued the cas server certificate
$cas_server_ca_cert_path = '/path/to/SCRoot2ca.pem';//証明書のフルパス
?>
```

各 php ファイルの上部に、以下のような API の呼び出しを記述します

```
<?php
//設定ファイルのロード
include_once('config.php');
//CAS ライブラリのロード
include_once('CAS.php');

// デバック用
//phpCAS::setDebug();
// CAS 初期設定
phpCAS::client(CAS_VERSION_2_0, $cas_host, $cas_port, $cas_context);

// CAS のサーバ証明書確認
phpCAS::setCasServerCACert($cas_server_ca_cert_path);
// CAS のサーバ証明書を確認しない,本番環境では用いられるべきではない
//phpCAS::setNoCasServerValidation();

//CAS 認証の実行
phpCAS::forceAuthentication();
?>
```

3.2.4. ソースコードの記述

認証 API を通過した時点で、CAS 認証が成功しています。

CAS サーバが返す情報をウェブアプリケーションで利用する場合、API を使って取得します。

ログインユーザの属性情報を取得する場合

```
$attributes = phpCAS::getAttributes();
$attributes["NagoyaUnivID"];
```

3.2.5. ログアウト

ウェブアプリケーションのログアウト処理を行った後、多要素認証 CAS サーバのログアウトをするために API を実行します。。

多要素認証 CAS サーバのログアウト後、ウェブアプリケーションの画面を再度表示させたい場合、service パラメータで URL を指定できます。

なお、リダイレクト可能な URL は、多要素認証 CAS サービスを使用している URL のみです。

```
// CAS サーバからのログアウト
phpCAS::logout();
// CAS サーバからログアウト後に、特定 URL へリダイレクトする場合
//phpCAS::logout(array('service'=>'http://mysite/'));
```

3.3. Apache 用のライブラリを使用する場合

3.3.1. 概要

Apache の基本認証に多要素認証 CAS サービスが使用できます。

使い方は「https://wiki.jasig.org/display/CASC/mod_auth_cas」に記載されていますので、本マニュアルでは実装例を紹介します。

なお、CAS サーバから返る属性情報を使用したい場合は、モジュールのソースコードを変更する必要がありますが、これは基盤課ではサポートされません。

3.3.2. モジュールの作成

ソースファイルをウェブからダウンロードし、コンパイルします。

```
$ apxs -i -lssl -lcurl -c mod_auth_cas.c
```

3.3.3. モジュールの配置と設定

作成した mod_auth_cas.so を modules に配置します。例えば/etc/httpd/modules です。

httpd.conf に追加します。

```
LoadModule auth_cas_module modules/mod_auth_cas.so
CASDebug                On
CASCertificatePath      /usr/share/purple/ca-certs/
CASValidateServer       On
CASLoginURL             https://auth-mfa.nagoya-u.ac.jp/cas/login
CASValidateURL          https://auth-mfa.nagoya-u.ac.jp/cas/serviceValidate
```

```
CASCookiePath           /tmp/
CASAllowWildcardCert    On
CASValidateDepth        3
```

認証をかける場所を指定します。

```
Alias /mod_cas /var/www/html/mod_cas
<Directory "/var/www/html/mod_cas">
    AuthType CAS
    Require valid-user
</Directory>
```

3.4. その他の言語の場合

「2 CAS 認証メカニズム」に示すプロトコルを実装すると、多要素認証 CAS サービスを利用できます。