OpenbizFrameworkService
describe openbiz service
Phase-Implementation

Updated Aug 11, 2011 by rockys...@gmail.com

Openbiz Service

Openbiz allows developers to write their special logic by implementing Plug-in Service. Openbiz services are also metadata-driven objects. Service code is under bin/service and metadata is under /metadata/service. Service metadata can specify the service name, implementation class as well as other service specific configuration data. Service metadata doesn't have fixed schema, this is because different services can have very different configuration.

Configure a service

Openbiz service can be configured in the following ways:

1. Change the original service metadata. For example, you can write different accessService.xml to control the view access rule, but the implementing class is accessService.php in core library.

```
<PluginService Name="accessService" Class="accessService">
  application specific view access definition
</PluginService>
```

2. Specify the service implementing class. For example, different applications have different profile service. The profileService.xml is like

```
<PluginService Name="profileService" Class="your_own_service_class">
</PluginService>
```

3. Combine the above 2 methods.

Write a service

It is straightforward to write your own Openbiz service - simply make a service metadata service_name.xml and service_name.php by copying an existing service.

In the service code, define a class that has the same name as the service name. If the service needs to read settings from metadata file, it should inherit from MetaObject class and process xml array from its contructor. See the following code snippet.

```
class emailService extends MetaObject
{
   public function __construct(&$xmlArr)
   {
       $this->readMetadata($xmlArr);
       ...
   }

   public function sentEmail($to, ......) {
       // send email logic here
   }
}
```

Openbiz BizSystem has "GetService" method to get an instance of the service. Then the service instance is used to invoke its public method.

```
$svcobj = BizSystem::GetService($service_name);
$svcobj->$method($param1, ...);
```

Openbiz Core Service

Openbiz core library includes services under openbiz/bin/service/ and their metadata files under openbiz/metadata/service/.

Log Service

Openbiz log service writes log message to log files. Openbiz log service is written upon Zend Log. Openbiz log service can be configured with 3 options:

- Log format
- Log level
- Log organization

Log Format

Logging format options include CSV, XML and HTML

Log Level

Log Levels are the highest level of logging to record. By default, Log Service logs all entries up to the selected level. If this is set to FALSE, logging is disabled. The log level can be chosen from

- EMERG = 0; // Emergency: system is unusable
- ALERT = 1; // Alert: action must be taken immediately
- CRIT = 2; // Critical: critical conditions
- ERR = 3; // Error: error conditions
- WARN = 4; // Warning: warning conditions
- NOTICE = 5; // Notice: normal but significant condition
- INFO = 6; // Informational: informational messages
- DEBUG = 7; // Debug: debug messages

Log Orgnization

Log files can be organized in different ways:

- profile- store log files by user id
- level - store log files groups by their level
- date - write log entries into a different file for each day

Sample Service Metadata

```
<?xml version="1.0" standalone="no"?>
<PluginService Name="logService" Package="service" Class="logService">
 <Log_Config Format="CSV" Level="7" Org="level"/>
</PluginService>
```

Email Service

Openbiz email service integrated with Zend_Mail to send emails through sendmail or smtp mail server. Mail transportation method and predefined email accounts can be specified in the email service metadata.

Mail Transportation

Openbiz email service accepts two mail transportation methods

- Send email by sendmail. This is deault if “isSMTP” is empty or set to “N”.
- Send email through STMP server“isSMTP” is set to “Y”.

Mail Accounts

In the email service metadata, one can specify more than one accounts as sender accounts. The following attributes are be set per account:

- Name. The identifier of the account element
- FromName. The name used as “from”.
- FromEmail. The email address used in “from”
- STMP setting
  - Host. SMTP host address.
  - SMTPAuth. If “Y”, username and password need to be given.
  - Username. SMTP server username
  - Password. SMTP server password.

Mail Log

Log can be written into text file or database table per each mail delivery. To enable email log, one can set.

- Type. Type can be either “DB” or “File”
- Object. This is the DO name if log type is “DB”.
- Enabled. If “Y”, mail log is enabled.

Sample Service Metadata

```
<?xml version="1.0" standalone="no"?>
<PluginService Name="emailService" Package="service" Class="emailService">
 <Accounts>
   <Account Name="System2" Host="localhost" FromName="admin" FromEmail="admin@yourcompany.com" IsSMTP="Y" SMTPAuth="Y" Use
   <Account Name="SystemNotifier" Host="smtp.qq.com" FromName="System Notification" FromEmail="jixian2003@qq.com" IsSMTP="
 </Accounts>
 <Logging Type="DB" Object="email.do.EmailLogDO" Enabled="Y" />
</PluginService>
```

DO Trigger Service

Upon data objects create/update/delete operations, Openbiz DO Trigger Service provides hooks to trigger custom defined actions. Once DO trigger service is set to a DO, it will be executed right after the corresponding DO action is completed successfully.

This is how it works. DO trigger has two parts - triggering events and to-be-triggered actions. These information are defined in DO Trigger service

metadata files. At runtime when a record is created, updated or deleted, Openbiz searches for this DO trigger by looking for its trigger metadata file with name DOName_trigger.xml under the same directory. For example, demo/EventDO's trigger metadata file is demo/BOEvent_trigger.xml.

Configure DO Trigger Service

The syntax of the DO trigger metadata is listed below:

```
<PluginService Name="DOName_Trigger" Class="service.doTriggerService" DataObjectName="DOName">
<DOTrigger TriggerType="CREATE|UPDATE|DELETE"> *
 <TriggerCondition Expression="" ExtraSearchRule="" />
 <TriggerActions>
    <TriggerAction Action="Method_Name">
      <ActionArgument Name="" Value="" /> *
    </TriggerAction>
 </TriggerActions>
</DOTrigger>
</PluginService>
```

Trigger Condition

- Expression. It is the Openbiz Simple Expression. For example, {Expense}>100. checks if current record's Expense field > 100.
- ExtraSearchRule. It can apply additional search rule upon the default DO search rule. For example, {AlertFlag}='Y' checks if there's at least one record whose AlertFlag is 'Y'.

Trigger Action

Trigger Action defines what action to be invoked when the trigger condition is met. Action attribute accepts the following options:

ExecuteSQL

ExecuteSQL executes SQL statement given in the ActionArgument elements. Sample:

```
<TriggerAction Action="ExecuteSQL">
  <ActionArgument Name="DBName" Value="Default" />
  <ActionArgument Name="SQL" Value="insert into alert values ('Record {[Id]} is updated', now())" />
</TriggerAction>
```

SendEmail

SendEmail sends outbound emails. It invokes email service sendEmail method. Sample:

```
<TriggerAction Action="SendEmail">
  <ActionArgument Name="EmailService" Value="service.emailService" />
  <ActionArgument Name="Account" Value="MyOpenbizAccount" />
  <ActionArgument Name="TOs" Value="tom@mail.com" />
  <ActionArgument Name="CCs" Value="" />
  <ActionArgument Name="BCCs" Value="" />
  <ActionArgument Name="Subjects" Value="Alert message" />
  <ActionArgument Name="Body" Value="This is an alert message. \nPlease notice that the record with {[Id], [name]} was de
</TriggerAction>
```

AuditTrail

AuditTrail records field changes. It invokes audit service Audit method. Sample:

```
<TriggerAction Action="AuditTrail">
  <ActionArgument Name="AuditService" Value="service.auditService" />
  <ActionArgument Name="DataObjectName" Value="{@:Name}" />
</TriggerAction>
```

Method Name

Method Name can be any method defined in DO Trigger Service. The method will be invoked. Sample:

```
<TriggerAction Action="MethodName">
  <ActionArgument Name="arg1 name" Value="arg1 value" />
  <ActionArgument Name="arg2 name" Value="arg2 value" />
</TriggerAction>
```

ID Generation Service

ID generation service is used by DO to generate the primary key (PK) of a new record. The table below lists supported method for generating ID.

IdGeneration	how id (PK) is generated	Supported database types
Identity	Primary key column is generated by database engine. Id is called "Identity" column in SQL Server, also called "auto-increment" column in MySQL.	MySQL, SQL Server, Sybase, DB2
Sequence:seqname	Primary key column is generated by database sequence. "seqname" is the sequence name used to generate PK	Oracle, PostgreSQL, DB2
GUID	Primary key column is database-generated GUID	MySQL, SQL Server, Oracle

UUID	Primary key column is generated with php uuid function	Apply to all database types
Other	Developers can write customer specific Id generation algorithm by modifying **function GetNewID(...)** in genIdService class under openbiz/bin/service/genIdService.php	decided by customer

Cache Service

Cache Service is used to cache data like DO query results, Form rendering data, View html data... Zend Cache class is used in its implementation. In cache service metadata, general cache setting and cache engine can be both specified.

Cache Setting

CacheSetting controls the general setting of cache. It mainly takes the Zend Cache frontend parameters.For details, please refer to http://framework.zend.com/manual/1.11/en/zend.cache.frontends.html Sample:

```
<CacheSetting Mode="Enable">
 <Config Name="lifetime" Value="3600" />
</CacheSetting>
```

Cache Engine

Cache backend settings are configured in CacheEngine element. CacheEngine cache backend type supported by Zend Cache backend and corresponding parameters. Sample:

```
<CacheEngine Type="File">
 <File>
   <Config Name="cache_dir" Value="data/" />
   <Config Name="file_locking" Value="Y" />
   <Config Name="read_control" Value="Y" />
   <Config Name="read_control_type" Value="crc32" />
   <Config Name="hashed_directory_level" Value="0" />
   <Config Name="hashed_directory_umask" Value="0700" />
   <Config Name="file_name_prefix" Value="openbiz" />
 </File>
<!--
 <Sqlite>
   <Config Name="cache_db_complete_path" Value="" />
   <Config Name="automatic_vacuum_factor" Value="" />
 </Sqlite>
 <Memcached>
   <Config Name="server" Value="localhost:11211" />
   <Config Name="compression" Value="N" />
   <Config Name="compatibility" Value="N" />
 </Memcached>
 <Xcache>
   <Config Name="user" Value="" />
   <Config Name="password" Value="" />
 </Xcache>
 <APC />
 <ZendPlatform />-->
</CacheEngine>
```

Security Service

Security Service provides application level security check on incoming requests. It can apply the following filters on requests.

- URL Filter
- Domain Filter
- IP Filter
- User Agent Filter
- Post Filter
- Get Filter

Each Filter has common metadata which defines the filter matching rule and action. The Rule has following attributes

- Name, the name of the rule.
- Action, can either “Allow” or “Deny”.
- Match, a regular expression used in matching the target string.
- EffectiveTime, the start and end time in format of “minute””second” between when the rule applies.

Sample

```
<UrlFilter Mode="Enabled" >
   <Rule Name="url_filter_1" Action="Deny" Match="\.\." EffectiveTime="0000-2359" />
   <Rule Name="url_filter_2" Action="Deny" Match="'" EffectiveTime="0000-2359" />
</UrlFilter>
```

Query Service

Query Service provides an universal interface to query DO and get results without initializing and accessing DO. It has 3 public methods to query a DO.

- fetchAll. Fetch all records based on search rule. Example:

```
$recordSet = $queryService->fetchAll(“ContactDO”, “[age]<10”);
```

- fetchRecord. Fetch one record based on search rule. Example:

```
$dataRecord = $queryService->fetchRecord("ContactDO", "[Id]=1011");
```

- fetchField. Fetch one record field based on search rule. Example:

```
$lastName = $queryService->fetchAll("ContactDO", "[Id]=1011", "last_name");
```

Validation Service

Validation Service extends Zend Validation class to provide the following types of data validation.

- String length.
  - shorterThan. Check if a value is shorter than the $max length
  - betweenLength. Check if a value is between a $min and $max length
  - longerThan. Check if a value is longer than the $min length
- Number value.
  - lessThan. Returns true if and only if $value is less than the minimum boundary.
  - greaterThan. Returns true if and only if $value is greater than the minimum boundary.
  - between. Returns true if and only if $value is between the minimum and maximum boundary values.
- strongPassword. Strong Password checks if the string is "strong", i.e. the password must be at least 8 character * and must contain at least one lower case letter, one upper case letter and one digit.
- email. Check if a string is a valid email address.
- date. Check if a string is a valid date format.
- phone. Check for US format ###-###-#### or (###)###-####
- zip. US Zip check in ##### or #####-####
- social. Social Security check for US format ###-###-#### or (###)###-####
- credit. Credit Card check for US format VISA/AMEX/DISC/MC
- address. Street Address check for US format

Excel Service

Excel Service provides several options to ouptut Openbiz Form data to excel readable files.

- renderCSV. Render the excel output with CSV (comma separated value) format
- renderTSV. Render the excel output with TSV (tab separated value) format

Excel Service also allows import CSV file to DO through its importCSV method. This method must be called from a popup form where a file is uploaded. The parent form of the popup form is the target to import. Of course, the above condition can be changed based on different implementation of the import method (developers can override or write their own one).

View Access Service

Openbiz provides simple control on view access via its view access service. The full-features ACL is supported by Openbiz Cubi which is discussed in Cubi document.

View access service metadata is the central place to control the access of views. The metadata has format as following.

```
<?xml version="1.0" standalone="no"?>
<PluginService Name="accessService" Class="accessService">
 <access-constraint>
   <view-collection>
   <!--
     <view name="view pattern support regular expression">
       <role name="role name 1"/>
       <role name="role name 2"/>
       <role name=""/>
     </view>
   -->
     <!-- system views only allows uses whose role is "admin" to access -->
     <view name="system*">
       <role name="admin"/>
     </view>
   </view-collection>
 </access-constraint>
</PluginService>
```

Other Services

Openbiz core includes other services like Authentication Service, User Profile Service and etc. These service must be rewritten at the application because difference applications will have different way to authenticate users and get their profiles. If an Openbiz application declares its own service with the same name of the core service, Openbiz will use the service in application.