Multitenancy Support

Multitenancy Support

Status: DRAFT - **see more recent discussions at Multitenancy scenarios and use cases**
Created: 2. April 2009
Author: fmeschbe
JIRA: SLING-2656
References: -
Updated: -

Problem Scope

Consider a hosting service provider, who wants to use Sling as the server to run the sites for different customers in the same Sling instance. Here you might want to separate the resolution of resources for each client.

For example some client C1 will have its components below `/apps/c1`, client C2 will have its components below `/apps/c2` and the service provider has shared components below `/apps/shared`. To shield the client components from each other, being able to have per-client search paths – as returned from `ResourceResolver.getSearchPath()` – would be a nice feature. This way, requests to client C1's site would have a search path of `[ "/apps/c1", "/apps/shared" ]` and requests to client C2's site would have search path `[ "/apps/c2", "/apps/shared" ]`.

Proposed Solution

The Sling API defines a new `Tenant` interface to represant tenants and a `TenantProvider` service interface which provides access to tenants. The tenant applicable to a given request is available through the new `SlingHttpServletRequest.getTenant()` method.

Basically a tenant has just three defined properties which are mainly intended to identify the tenant and list it for human consumption:

- A globally unique identifier
- A (short) name, which is not required to be unique
- A (longer) description

In addition a tenant may provide a number of application specific properties. The properties are name value pairs, where the names are strings and the values may be of any type.

One application of the tenant and its property will be the `JcrResourceResolverFactory`. A new `getResourceResolver(Session, Tenant)` method is defined, where the tenant is used to inject configuration for the `ResourceResolver` returned. In a first step the `resource.resolver.searchpath` tenant property is used to enhance the globally configured search path. If the property is set and can be converted to a `String[]` the entries are merged with the global search path as follows:

```
String[] globalPath = // globally configured search path
String[] tenantPath = // per-tenant search path
List<String> searchPath = // search path merged from globalPath and tenantPath

// add absolute tenantPath entries
for each entry in tenantPath
   if entry is absolute then searchPath.add(entry);
done

// add relative tenantPath entries
for each globalEntry in globalPath
   for each entry in tenantPath
      if entry is relative searchPath.add(globalEntry + "/" + entry);
   done
   searchPath.add(globalEntry);
done
```

Example: For a globalPath of `[ "/apps", "/libs" ]` and a tenantPath of `[ "/tenant", "c1", "shared" ]` the merged search path will be `[ "/tenant", "/apps/c1", "/apps/shared", "/apps", "/libs/cq", "/libs/shared", "/libs"]`.

If the `resource.resolver.searchpath` property of the tenant does not exist or if no tenant is available to the `JcrResourceResolverFactory` the global search path configuration is used unmodified.

In the future more properties might be used to augment the configuration of a resource resolver returned for a given tenant.

Proposed API

Tenant

The `Tenant` interface is defined in the `org.apache.sling.api.tenant.Tenant` package of the Sling API bundle `org.apache.sling.api`.

Tenant.java

```
/**
 * The <code>Tenant</code> interface represents a tenant which may be used to
 * further customize request and other processing.
 * <p>
 * This interface is intended to be implemented by the implementor of the
 * {@link TenantProvider} interface to be returned for it tenant accessor
 * methods.
 */
public interface Tenant {

    /**
     * The name of the {@link #getProperty(String) property} whose string
     * representation is used as this tenant's {@link #getName() name} (value is
     * "sling.tenant.name").
     *
     * @see #getName()
     * @see #getProperty(String)
     */
    static final String PROP_NAME = "sling.tenant.name";

    /**
     * The name of the {@link #getProperty(String) property} whose string
     * representation is used as this tenant's {@link #getDescription()
     * description} (value is "sling.tenant.description").
     *
     * @see #getDescription()
     * @see #getProperty(String)
     */
    static final String PROP_DESCRIPTION = "sling.tenant.description";

    /**
     * Returns the unique identifier of this tenant.
     * <p>
     * The tenant identifier has not predefined mapping to a property and may be
     * generated automatically by the TenantProvider.
     */
    String getId();

    /**
     * Returns the name of the tenant. This is a short name for quickly
     * identifying this tenant. This name is not required to be globally unique.
     * <p>
     * The name of the tenant is the string representation of the
     * {@link #PROP_NAME} property.
     */
    String getName();

    /**
     * Returns a human readable description of this tenant.
     * <p>
     * The description of the tenant is the string representation of the
     * {@link #PROP_DESCRIPTION} property.
     */
    String getDescription();
    /**
     * Returns the named property or <code>null</code> if no such property
     * exists or if the property value itself is <code>null</code>.
     */
    Object getProperty(String name);

    /**
     * Returns the named property converted to the requested type or
     * <code>null</code> if the property does not exist, the property value
     * itself is <code>null</code> or cannot be converted to the requested type.
     */
    <Type> Type getProperty(String name, Type type);

    /**
     * Returns an iterator or String values representing the names of defined
     * properties of this tenant.
     */
    Iterator<String> getPropertyNames();
}
```

TenantProvider

The `TenantProvider` interface is defined in the `org.apache.sling.api.tenant.Tenant` package of the Sling API bundle `org.apache.sling.api`.

TenantProvider.java

```
/**
 * The <code>TenantProvider</code> defines the service interface of for a sevice
 * which may be asked for {@link Tenant tenant instances}.
 * <p>
 * For now this provider interface provides access to a tenant applying to a
 * particular request as well as to all tenants known to this provider.
 */
public interface TenantProvider {

    /**
     * Returns the {@link Tenant} with the given <code>tenantId</code> or
     * <code>null</code> if no such tenant exists.
     */
    Tenant getTenantById(String tenantId);

    /**
     * Returns an iterator of all {@link Tenant tenants} known to this provider.
     * If no tenants are known the iterator is empty.
     */
    Iterator<Tenant> getTenants();

    /**
     * Returns an iterator of {@link Tenant tenants} whose names match the given
     * <code>tenantName</code>. Since the name of a tenant is not required to be
     * globally unique, this method may return more than one tenant.
     * <p>
     * If no tenants are known with the requested name the iterator is empty.
     */
    Iterator<Tenant> getTenantsByName(String tenantName);

    /**
     * Returns an iterator of {@link Tenant tenants} matching the given
     * <code>tenantFilter</code>.
     * <p>
     * The <code>tenantFilter</code> is a valid OSGi filter string as defined in
     * Section 3.2.6, Filter Syntax, of the OSGi Core Specification, Release 4.
     * <p>
     * If no tenants match the <code>tenantFilter</code> or the
     * <code>tenantFilter</code> is not a valid filter string the iterator is
     * empty.
     */
    Iterator<Tenant> getTenantsByFilter(String tenantFilter);
}
```

AdapterFactory

The Tenant provider bundle should also register an `AdapterFactory` to adapt `ResourceResolver` and other objects to `Tenant` instances.

SlingHttpServletRequest

The `SlingHttpServletRequest` interface is extended to return the `Tenant` applicable for the request.

SlingHttpServletRequest.java

```
public interface SlingHttpServletRequest extends HttpServletRequest {
    ...
    /**
     * Returns the {@link Tenant} applicable to this request or
     * <code>null</code> if no tenant can be resolved for this request.
     */
    Tenant getTenant();
    ...
}
```

SlingHttpServletRequestWrapper

The `SlingHttpServletRequestWrapper` interface is extended to return the `Tenant` applicable for the request.

SlingHttpServletRequestWrapper.java

```
public class SlingHttpServletRequestWrapper extends HttpServletRequestWrapper
        implements SlingHttpServletRequest {
    ...
    public Tenant getTenant() {
        return getSlingRequest().getTenant();
    }
    ...
}
```

JcrResourceResolverFactory.java

```
public interface JcrResourceResolverFactory {
    ...
    ResourceResolver getResourceResolver(Session session, Tenant tenant);
    ...
}
```