Multitenancy Support Integration

Tenant Integration

See more recent discussions at Multitenancy scenarios and use cases

Introduction

Like with tenants in an apparment complex tenants in Sling are *users* that have their own space but share common amnenities like an elevator, laundry room and so on.
Now for Sling **Tenants** mean severals things and this document wants to document them, their use cases and look for an implementation of the requirements in Sling.

This page is based on the Multitenancy Support page that describes the Tenant feature as well as the current Tenant module.

Currently there is a **Tenant** module in the **contrib/extensions** package which provides the following:

- Tenant
- Tenant Manager and Provider interface
- TenantProviderImpl that implements the Tenant Manager and Provider Interface
- TenantAdapterFactory which can adapt a Resource or Resource Resolver to a Tenant
- Tenant Customizer interface that lets a client add a service to further customize a Tenant during creation and clean up during removal

This package provides the basics but does not provide any implementation a multi-teant aware server. There are also parts of the proposal missing like providing the Tenant on the Sling Http Servlet Request.

Tenant Definition

A full fledged Tenant implementation would allow the Sling host to create a tenant, create group(s) / users(s) to login as **tenant developer** on the site and provide a tenant specific view to the public. This includes the following features:

1. **Visibility**: A tenant developer and the public tenant viewer only see the tenant space and don't have access to other tenants
2. **Customization**: The Sling host can provide basic frameworks which then tenant then can overlay with its own customization which includes:
    a. **Code**: ESPs, JSPs and Servlets can be overlaid by a tenant to provide its own view
    b. **I18N**: Tenants must be able to overlay the Internationalization so that they can provide and use their own translations
    c. **Content**: Most tenants will provide their own content but the host might provide some common content so there must be a way to fall back onto common content.
    d. **Discovery**: A tenant must be discovered based on various data like the logged in user, resource path, sub domain name, cookie (a service outside of Sling is setting a tenant cookie) and others. This is especially important for the public view but also for administrators which might want to see a particular tenant view without impersonating
3. **Configuration**: Sling should provide a way to setup a tenant in a simple step. For that Sling should provide its own Tenant Customizer which creates the necessary paths, groups / users and content folders.

Implementation

Visibility can be accomplished by access permission for the **developer** and through Path Mapping for the public view.

Customization needs a per-call extension of the Resource Resolver's Search Path. Currently the Search Path is provided from the Resource Resolver Factory which is a system-wide setting delivering the same resource. Tenants require an independent view of other tenants and therefore needs to have their own Search Path so that the Resource Resolver can provide different, tenant specific resources which is based on properties from the request. This is not per-se a Tenant specific requirements and maybe others would like to use that feature but Tenants requires it to obtain the Code, Internationalization and Content for a specific Tenant and avoiding to access resources from other Tenants. Because the Servlet Resolver uses an Administrative Resource Resolver it cannot know of the per-call Search Path and it also uses a Cache that stores the first found Servlet in its cache. This requires a way to enhance the Administrative Resource Resolver's Search Path (like an one-off Resource Resolver wrapper) for that call with the Search Path from the incoming Resource Resolver and it would require to manage the cache differently so that an overlay of a Tenant is cached per Tenant and not globally otherwise only the first overlay of a Tenant is used through the server.

Discovery: The Tenant Adaptor Factory is assuming that tenants are either indified through a logged in user or a given path but that is not always true. For example the host administrator might want to edit some of the Tenants code or wants to review changes maybe to shared code / content like a login page. Maybe a Service interface which a client could implement would give the discovery more flexibility to that process. For example the tenant might be specified in a subdomain (like tenan1t.myhost.com) and does request a shared page from the host. That page then imports other pages (ESPs, JSPs, Servlets) like the header, footer or components of that page which could might be customized by the Tenant through overlays. The resource path would not indicate a Tenant and hence the overlays would be ignored.

Configuration: A sample project might be best to illustrate on how to handle tenants. This could server to purposed. First to showcase the Tenant managment and can be used by clients as template to implement their template handling.

Use Cases and Proposed Solutions

1. Host provides a Framework, Tenant can Customize It

Use Case: a Host provides a general framework to its tenants and gives the tenant to opportunity to customize it through overlayings code, translations and content.

Solution: A per-call Search Path (first entries are the tenant specific search paths followed by the Resource Resolver Factory Search Path) is set on the Resource Resolver of the call and on the Resource Resolver of the Servlet Resolver. In the Servlet Resolver the cache handling must be changed because we can now have multiple servlets, one for each tenant.

2. Host wants to limit what Parts are Customizable

Use Case: a Host wants to prevent the Tenant developer from customizing key parts of its framework like the login page etc.

Solution: the Resource Resolver or Servlet Resolver needs to check the other resource candiates if they are looked.

3. Tenants are Isolated

Use Case: a Tenant Deverloper and Viewer should not see the parts of other Tenants.

Solution: The current security model should be enough to prevent access to other tenatns resources. That said a Tenant Customizer could be used to make sure that ACLs are created and put into place.