Domain Spatial Service Extension (DSSE)

SuperMap iServer classifies spatial services into two categories: 1. Generic Spatial Services (GSS), e.g. map services, spatial data services, spatial analysis services, network analysis services, etc., and SuperMap iServer provides this type of services by default; 2. Domain Spatial Services (DSS), which are spatial information related services constructed by the user according to domain-specific professional logics.

SuperMap iServer provides a flexible mechanism for constructing and publishing services that applies to both GSS and DSS.

Why providing DSSE

When the general GIS services cannot satisfy specific needs of a certain industry, the GIS service platform needs to provide a flexible mechanism for extending the services. Developers can incorporate these specific function requirements into the GIS service platform by using this extension mechanism, to provide the industry with a service system for processing spatial information.

Currently, there are a number of approaches to extending GIS services in the GIS platform. Each approach has its own advantages and disadvantages, and is suitable under certain situations.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed extension</td>
<td>Custom interfaces are provided in a service, based on which developers can implement specifics of</td>
<td>Easy development. It is only needed to implement method overloading.</td>
<td>The interface is not transparent. The interface user and provider need to agree on the format of passed</td>
</tr>
<tr>
<td>Building function modules</td>
<td>Implements a set of workflow-based GIS services through building function modules. The GIS service platform provides a mechanism for building models. Developers can build models for GIS function modules, in order to realize GIS services that have certain workflow for processing business.</td>
<td>The modeling is simple and easy to use.</td>
<td>It can build models for existing function modules, but cannot create or integrate extensions for new business.</td>
</tr>
</tbody>
</table>
This extension mechanism provides flexible and easy-to-extend framework for developing service components. In this framework, developers can quickly develop domain spatial information services with characteristics suitable for various domains using the basic GIS modules provided in SuperMap iServer. Domain spatial information services can be integrated to the SuperMap iServer service system using simple and flexible configuration tools. People who need services can then obtain more professional spatial service.

Flexible and easy to extend.
What is DSS

SuperMap iServer categorizes spatial services into two types. One is general spatial services (GSS) provided by SuperMap iServer itself, including 3D, etc. The other type is domain spatial information services (DSS) which users build by themselves for processing spatial information based on special business logics of their domains, such as a map service for wind direction symbols applied in the meteorology industry. Although users need to develop the services themselves, SuperMap iServer provides a flexible and agile framework for developing service components as well as a mechanism for operating service components. When developing service components, users can incorporate their business logics with the built-in GIS service modules, and easily integrate their DSS in the service framework in SuperMap iServer.

What applications need DSS

Basic map services, spatial analysis services, realspace services, and data services provided in SuperMap iServer cover most of the spatial processing functions. But spatial information service alone may not satisfy business requirements of an industry. It may be needed to incorporate the business features or existing business data of an industry into spatial information processing, in order to build a set of spatial information services proper for the specific business logics of this industry. To achieve this, custom business service components can be built with the domain spatial information service mode provided in SuperMap iServer. Then the services can be published by using the domain spatial information service extension mechanism of SuperMap iServer.

- Domain spatial information services built from integrating GIS and business models required by the system

Business rules in different industries can be different. Some businesses have logics requirements on spatial information processing that built-in GIS processing modules in SuperMap iServer cannot realize. Therefore users can build custom domain spatial information services that incorporate business logic relationships in their industry to process spatial information. To achieve this, users can employ the GIS processing modules in SuperMap iServer. For example, a stochastic hydrological modeling analysis service can be built for
the water conservancy industry. The built-in GIS analysis functions in SuperMap iServer cannot implement the algorithms for stochastic hydrological modeling, whereas the hydrological analyses in the water conservancy industry do not consider the characteristics of spatial variability. Therefore, users can integrate the stochastic hydrological modeling algorithms in the industry and the spatial information processing functions in GIS, to build a spatial information-oriented stochastic hydrological modeling analysis service for the water conservancy industry. This service can be used to analyze drought characteristics and stochastic variability of storm and flood processes in a watershed, to design retention systems, and to support decision making on planning a hydrological and water resource system.

- Domain spatial information services with multiple GIS functions combined

Usually every interaction between a client and a SuperMap iServer spatial service is based on a single GIS function request, such as finding the query result and map location corresponding to a keyword. If a business request needs a combination of two or more GIS functions to obtain a result, the user can combine multiple GIS functions through building a domain spatial information service. This type of domain spatial information services only requires the user to arrange a set of GIS functions. The specific GIS operations can be implemented using the service provider modules in the SuperMap iServer framework.

- Domain spatial information services integrated with original business systems

For businesses that have built information platforms, integrating GIS technology’s capacity to process spatial information can certainly help with decision making and information management in the businesses. Users can build domain spatial information services that integrate the original business systems, information management, and decision systems, by using the extension mechanism of SuperMap iServer spatial information services. Interfaces that can be interacted with the original systems can be published in the domain spatial information services. In this way, the original business data can be effectively presented with GIS functions, and the information representation as well as business decision making and management can be fully supported. Domain spatial information services in SuperMap iServer provide some special industries with excellent solutions to the integration between business and spatial information.
DSSE features

- An open framework of a service system allows easy DSS extension

The framework of a service system in SuperMap iServer is composed of layers and modules. Services are configured and deployed in a unified fashion. A service framework in SuperMap iServer has three layers. Each layer contains relatively independent function modules in which specific operation contents are implemented. Users can build function modules that conform to the business logics of their industries on any level of the framework, or develop domain spatial information service components by extending a module on a layer. The modules on the layers of services published by SuperMap iServer are organized by the service configuration and management component. Users can deploy their custom DSSs to the SuperMap iServer service system using the service configuration and management component. By doing this, the DSSs are published.

- Supporting abundant and multi-level GIS service modules

As mentioned above, relatively independent function modules are built in each level of the SuperMap iServer service framework. For example, the basic map service provider module—UGCMAPProvider, the 3D service provider module—3DProvider, the RESTful interface module—REST Servlet, etc. These modules communicate with other modules via interfaces. When designing and developing a DSS, the user can implement the GIS processing part by using built-in GIS service modules in SuperMap iServer, and then combine GIS and the business logics of the industry. The built-in GIS service modules in SuperMap iServer are tools for a user to implement the GIS part of a DSS, so that the user can quickly develop a spatial-oriented domain service component.

- Flexible mechanism for customizing business services

The extension mechanism of domain spatial information services in SuperMap iServer provider users with a flexible construction mechanism for industry application-based spatial operations. All the modules in the framework of a SuperMap iServer service system are organized and published with the service configuration and management component, which operates by reading the service assembly information and module metadata. This kind of configuration mechanism makes it possible for users to freely combine function modules from the
layers. To integrate a custom DSS to a SuperMap iServer system and publish the DSS, the user only needs to describe the metadata of the DSS, and describe its modules and its relationships between the modules on each layer.

DSSE features

- An open framework of a service system allows easy DSS extension

The framework of a service system in SuperMap iServer is composed of layers and modules. Services are configured and deployed in a unified fashion. A service framework in SuperMap iServer has three layers. Each layer contains relatively independent function modules in which specific operation contents are implemented. Users can build function modules that conform to the business logics of their industries on any level of the framework, or develop domain spatial information service components by extending a module on a layer. The modules on the layers of services published by SuperMap iServer are organized by the service configuration and management component. Users can deploy their custom DSSs to the SuperMap iServer service system using the service configuration and management component. By doing this, the DSSs are published.

- Supporting abundant and multi-level GIS service modules

As mentioned above, relatively independent function modules are built in each level of the SuperMap iServer service framework. For example, the basic map service
provider module--UGCMAPProvider, the 3D service provider module--3DProvider, the RESTful interface module--REST Servlet, etc. These modules communicate with other modules via interfaces. When designing and developing a DSS, the user can implement the GIS processing part by using built-in GIS service modules in SuperMap iServer, and then combine GIS and the business logics of the industry. The built-in GIS service modules in SuperMap iServer are tools for a user to implement the GIS part of a DSS, so that the user can quickly develop a spatial-oriented domain service component.

- Flexible mechanism for customizing business services

The extension mechanism of domain spatial information services in SuperMap iServer provider users with a flexible construction mechanism for industry application-based spatial operations. All the modules in the framework of a SuperMap iServer service system are organized and published with the service configuration and management component, which operates by reading the service assembly information and module metadata. This kind of configuration mechanism makes it possible for users to freely combine function modules from the layers. To integrate a custom DSS to a SuperMap iServer system and publish the DSS, the user only needs to describe the metadata of the DSS, and describe its modules and its relationships between the modules on each layer.
Server service framework is a three-layer architecture, that is, GIS service provider layer, GIS service component layer, and GIS service interface layer. For more details, please refer to Architecture. DSSE can be performed on any of the three layers.

- Extension for Provider Layer

Service providers in iServer indicates the sources GIS capabilities. By extending service providers, the sources of GIS capabilities can be expanded. For instance, a map service provider can be extended to read iServer 2008 map caches, therefore publishing map caches from iServer 2008 as different types of services, like REST services, WMS services, etc., in iServer. Also, a map service provider can be extended to realize the capability of getting GoogleMaps map services, therefore publishing map services from GoogleMaps as different types of services, like REST services, WMS services, etc., in iServer.

Meanwhile, new types of service providers can be defined to realize capabilities for certain domains. Domain service providers can be combined and encapsulated on the component layer with other service providers as more coarse-weight module, realizing the integration of domain capabilities and GIS capabilities.

**Extension for Provider Layer**

The implementation of the service provider is as follows:

```java
public class SampleProvider implements ProviderContextAware {
    ...
}
```
ProviderContextAware is used to get the service provider context, through which the configuration information of the service provider can be got.

For the configuration of the three-layer structure, see iServer Configuration File

- **Service Provider interface**
- **Service Provider context**
- **Extending RemoteTiledMapProviderBase**
  - Introduction
  - Sample
- **Extending LocalTileProviderBase**
  - Introduction
  - Sample

### Service Provider interface

The types of service providers already provided by SuperMap iServer include: MapProvider, DataProvider, RealspaceProvider, SpatialAnalystProvider, and TransportationAnalystProvider. SuperMap iServer has implemented these service providers to ensure the sources of GIS capabilities (Refer to Service Providers).

Users can extend service providers to expand the sources of GIS capabilities provided by SuperMap iServer.

#### Table 1 Interfaces or classes for extending service provider

<table>
<thead>
<tr>
<th>Type</th>
<th>Classes/Interfaces</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapProvider</td>
<td>com.supermap.services.compoennts.spi.MapProvider</td>
<td>All map providers need to implement this interface. The service provider that implements the interface can be used by the map component.</td>
</tr>
<tr>
<td></td>
<td>com.supermap.services.component</td>
<td>Encapsulated the basic functions needed to</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>s.spi.AbstractRestMapProvider</td>
<td>access REST map service, all the service providers using REST map service can use this class to extend.</td>
<td></td>
</tr>
<tr>
<td>com.supermap.services.component.s.spi.RemoteTiledMapProviderBase</td>
<td>RemoteTiledMapProviderBase is an abstract class that used to aggregate the third-party map services. It implements ProviderContextAware and MapProvider. (See: RemoteTiledMapProviderBase_Extension)</td>
<td></td>
</tr>
<tr>
<td>com.supermap.services.component.s.spi.LocalTiledMapProviderBase</td>
<td>LocalTileProviderBase is an abstract class for extending the local map tile service, and the local map tiles are derived from ArcGIS. (See: LocalTiledMapProviderBase_Extension)</td>
<td></td>
</tr>
<tr>
<td>Other implementation classes (See: Service Provider Introduction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com.supermap.services.component.s.spi.DataProvider</td>
<td>All data providers need to implement this interface. The service provider that implements the interface can be used by the data component.</td>
<td></td>
</tr>
<tr>
<td>Other implementation classes (See: Service Provider Introduction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com.supermap.services.component.s.spi.SpatialAnalystProvider</td>
<td>All spatial analyst providers need to implement this interface. The service provider that implements the interface can be used by the spatial analyst component.</td>
<td></td>
</tr>
<tr>
<td>Other implementation classes (See: Service Provider Introduction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com.supermap.services.providers.s.pi.TransportationAnalystProvider</td>
<td>All transportation analyst providers need to implement this interface. The service provider that implements the interface can be used by the transportation analyst component.</td>
<td></td>
</tr>
<tr>
<td>Other implementation classes (See: Service Provider Introduction)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The restrictions above are not applicable to defining new service provider types to realize functions in a certain domain.

### Service Provider context

In the service provider implementation class, the service provider context can be obtained through the ProviderContextAware interface, and the service provider configuration information can be obtained through the service provider context.

The configuration information of the service components is in services.xml. Below is the configuration for a service component: The configuration of service provider is as follows:

```xml
<provider class="com.supermap.sample.SampleProvider"
    name="sampleProvider">
    <config class="com.supermap.sample.SampleProviderSetting">
        <param1>default</param1>
        ...
    </config>
</provider>
```

Where, SampleProviderSetting is the configuration class of SampleProvider. param1 is the property of SampleProviderSetting. Through SampleProviderSetting, the service providers with different types and sources can be built together in iServer.

In the implementation class of the service provider, the service provider context which enables to get configuration (services.xml) of the service provider can be obtained from the ProviderContextAware interface.

```java
public class SampleProvider implements ProviderContextAware{
    ...
}
```
public void setProviderContext(ProviderContext context) {
    SampleProviderSetting sampleProviderSetting =
    context.getConfig(SampleProviderSetting.class);
    ...
}

Where sampleProviderSetting is the configuration item <config/> of
<provider/> in services.xml.

Extending RemoteTiledMapProviderBase

Introduction

RemoteTiledMapProviderBase is an abstract class that used to
aggregate the third-party map services and build online map
providers. It implements ProviderContextAware and MapProvider.

The main output method of this class is
getTileImageUrl(TileImageParameter tileImageParameter).

1. How to develop extension

You can aggregate a third-party map services by inheriting from the abstract
class of RemoteTileProviderBase.

The methods are shown as follows:

• initDefaultMapParameter(), initialize the default map parameters;
• cacheEnabled(). Whether to set caches.
• getOrigin(String paramString), return the origin coordinates of the map tile;
• initSupportedMapNames(), initialize the list of maps supported.
• getSupportImageFormat(String paramString), return the output map format.
• getSupportResolutions(String paramString), return the set of resolutions supported.
  If all resolutions are supported, then return null;
• getTileImageUrl(TileImageParameter tileImageParameter), construct the url by setting
  the current Resolution and the row and column index number of the tile at the
current resolution. Users call this method to get url, send request to the third-party services and get images. You can test url by browser.

**Note:**

- If it is a protected method, the subclass of the extension and this class should be both defined in the same package. Then it can be called.
- Return a blank map if the geographic extent you request is beyond your defined extent.
- When the request scale doesn't match the fixed scale, the scale will be changed automatically.
- If you want to configure the third-party map services that you access, you can inherit the MapProviderSetting class.

2. **Configuration and Deployment**

Place the compiled classes files to %SuperMap iServer_HOME%/webap\iserver\WEB-INF\(classes folder does not exist by default.), or the jar package to %SuperMap iServer_HOME%/webap\iserver\WEB-INF\lib\. iServer can use extensions of the service provider classes. For configuration, see [Configure service provider by XML](#).

3. **About cache**

`cacheEnabled()` returns true or false that can be used to enable or disable using cache.

When `MapParameter.cacheEnabled` is true, the system uses image cache. The catalog is `MapProviderSetting.getOutputStream()`, and its path is “cache\MapName\ImageWidth\ImageHeight\ScaleReciprocal\IndexImage\Suffix”. Meanwhile, this class also enables to cache the third-party map services, and the path is “cache\http\MapName\ImageWidth\ImageHeight\ScaleReciprocal\IndexImage\Suffix”.

**Sample**

Here you are allowed to extend a map service provider. `GoogleMapsMapProvider` is inherited from `TiledMapProviderBase`. You can find the sample code in %SuperMap iServer_HOME%/samples/code/DSSE/GoogleMapsProviderSample.
The code for the implementation of GoogleMapsProvider is as follows:

GoogleMapsMapProvider.java

GoogleMapLayer in GoogleMapsMapProvider class represents that the current map uses the Google layer (When extend other provider, you can refer to GoogleMapLayer to define corresponding Layer). The codes of GoogleMapLayer are as follows:

GoogleMapLayer.java

Configuration class uses MapProviderSetting, as follows:

```xml
<provider
class="com.supermap.sample.serviceprovider.GoogleMapsMapProvider"
name="mapProvider-GoogleMaps">
  <config
class="com.supermap.services.components.spi.MapProviderSetting"/>
</provider>
```

Place the Jar package in %SuperMap iServer_HOME%/webapps/iserver/WEB-INF/lib after compiling.

Create a new map component—map-google to publish rest, WMS 1.1.1 services using mapProvider-GoogleMaps.

```xml
<component class="com.supermap.services.components.impl.MapImpl"
interfaceNames="rest,wms111" name="map-google"
providers="mapProvider-GoogleMaps">
  <config class="com.supermap.services.components.MapConfig"/>
</component>
```

Access the map REST service with the root URI: http://<host>:<port>/iserver/services/map-google/rest. And access the map with the name googlemaps, which is declared in the implementation class of GoogleMapsMapProvider, through http://<host>:<port>/iserver/services/map-google/rest/maps/googlemaps.ijs. with the result shown as below:
Extending LocalTileProviderBase

Introduction

LocalTileProviderBase is used to extend the local map tile service. It implements the ProviderContextAware and MapProvider.

The method used to output map is
getTileImage(TiledMapProviderBase.TileImageParameterparameter).

1. How to develop

Users can inherit LocalTileProviderBase to extend the local map tile service.

Methods to use:

- initDefaultMapParameters(), initialize the default map parameters;
- cacheEnabled(). Whether to set caches.
- getOrigin(String paramString), return the origin coordinates of the map tile;
- initSupportedMapNames(), initialize the list of maps supported.
- getSupportImageFormat(String paramString), return the output map format.
- getSupportResolutions(String paramString), return the set of resolutions supported. If all resolutions are supported, then return null;
- `getTileImage(TiledMapProviderBase.TileImageParameter paramTileImageParameter)`, construct the url by setting the current Resolution and the row and column index number of the tile at the current resolution. Get the tile image and assign it to the `BufferedImage` object, and return the `BufferedImage` object.

**Note:**
- If it is a protected method, the subclass of the extension and this class should be both defined in the same package.
- If you want to configure the third-party map services that you access, you can inherit the `MapProviderSetting` class.

2. **Configure and Deploy**

Place the compiled classes files to `%SuperMap iServer_HOME%/webap\iserver\WEB-INF\`, or the jar package to `%SuperMap iServer_HOME%/webap\iserver\WEB-INF\lib`. iServer can use extensions of the service provider classes. For configuration, see Configure service provider by XML.

3. **About Cache**

`cacheEnabled()` returns true or false which is used to set using cache or not.

True means to use cache. The image cache directory, namely, the directory of `MapProviderSetting.getOutputPath()`, is “cache\MapName\ImageWidth x ImageHeight\ScaleReciprocal\Image x Index\Image y Index\ImageSuffix”. Meanwhile, this class also caches for the third-party map services, and the path is “cache\http\MapName\ImageWidth x ImageHeight\ScaleReciprocal\Image x Index\Image y Index\ImageSuffix”.

**Sample**

The sample takes ArcGISExplodedCacheMapProvider as an example which inherits from the class LocalTileProviderBase to demonstrate how to publish your local ArcGIS Server map tiles to SuperMap services. You can find the sample code in `%SuperMap iServer_HOME%/samples/code/DSSE/ArcGISExplodedCacheMapProviderSample`.

**Notes:** When generating ArcGIS Server tiles, choose "Exploded" as the format, and choose from NG8, PNG24, PNG32 as the image format.
ArcGISExplodedCacheMapProvider Sample Code:

ArcGISExplodedCacheMapProvider.java

For the class ArcGISExplodedCacheMapProvider, ArcGISExplodedCacheMapLayer means ArcGIS layer is used currently (to extend other map provider, please refer to ArcGISExplodedCacheMapLayer to define the Layer).

ArcGISExplodedCacheMapLayer Sample Code:

ArcGISExplodedCacheMapLayer.java

ArcGISXMLParseTool is used to get the conf.xml file of the ArcGIS Server tile directory and some node values in the conf.cdi file.

cnf.xml file is as follows:

```xml
<CacheSource id="123" name="ArcGIS" type="MapSource">
  <CacheSourceType id="ArcGIS" name="ArcGIS"/>
  <Projection id="123" name="ArcGIS"/>
  <ServiceDescription id="123" name="ArcGIS"/>
  <Server id="123" name="ArcGIS"/>
</CacheSource>
```
conf.cdi file is as follows:

```xml
<provider
class="com.supermap.sample.serviceprovider.ArcGISExplodedCacheMapProvider"
name="mapProvider-ArcGISExplodedCacheMap">
  <config>
    <cachePath>C:/arcgisserver/arcgiscache/China400/China400</cachePath>
    <mapName></mapName>
  </config>
</provider>
```

ArcGISXMLParseTool Sample Code:

ArcGISXMLParseTool.java

ArcGISExplodedCacheMapProviderSetting inherits from MapProviderSetting. It is used to extend MapProviderSetting to set some specific map parameters. cachePath is the path to store the map caches and mapName is the map name.

ArcGISExplodedCacheMapProviderSetting Sample Code:

ArcGISExplodedCacheMapProviderSetting.java

Use ArcGISExplodedCacheMapProviderSetting that herits from MapProviderSetting to do the settings. You must set the cache path by <cachePath>. Optionally set the map name by <mapName>. If the map name is not set, the default will be used. Here shows how to do the settings:

Place the compiled Jar to %SuperMap iServer_HOME%/webapps/iserver/WEB-INF/lib.

Create a new “map service component” map-arcgis that uses mapProvider-ArcGISMaps, and publish the service as rest, WMS 1.1.1:
Access the map REST functions via URI: http://<host>:<port>/iserver/services/map-arcgis/rest. Access the map via http://<host>:<port>/iserver/services/map-arcgis/rest/maps/arcgismaps.ijs. Arcgismaps is the map name that is declared in ArcGISExplodedCacheMapProvider. The result is as follows:

- **Extension for Component Layer**

  The service component layer is responsible for the combination and encapsulation of capabilities provided by service providers.

  When the capabilities of the current service components cannot satisfy the needs, we can extend the existing service components to encapsulate more capabilities.
Meanwhile, the extension of the service component layer can be combined with the service interface layer extension. The newly extended service component can be published as Web service through the extended service interface, which is got from defining new resource in the JAX-RS mechanism, customizing new service interface, etc.

For instance, for the applications in the Meteorological field, to integrate weather forecast and weather alerting services into iServer, the service component layer needs to be extended accordingly.

**Extension for Component Layer**

Below demonstrates the implementation of the component layer:

```java
@Component(providerTypes={MapProvider.class},optional=false, type = "PushpinMap")
public class PushpinComponent implements ComponentContextAware{
    ...
}
```

@Component annotates the meta information of the service component, with type indicating the type the service component, providerTypes indicating the service providers associated with the service component, and nameMapping indicating an implementation class of the NameMapping class, which is used to provide service interface information when clustering and is required when clustering is employed (SuperMap iServer supports for clustering extended services, please refer to Building DSS Cluster).

**Service Component context**

**Extension example**
Service Component context

In the implementation class of the service interface, ComponentContextAware can be used to get the service component context, through which the configuration information of the service providers and service components is provided.

Note: To improve the flexibility and make the structure of the domain service clearer, it is suggested that the business logic should not be placed on the business component layer. It is suggested that the business logic should be placed on the service provider layer, with the business capabilities implemented through the service providers by the service components.

Getting service component configuration

The configuration information of the service components is in iserver-services.xml (please refer to iServer configuration file for more details). Below is the configuration for a service component:

```xml
<component class="com.supermap.sample.SampleComponent"
    interfacenames="rest" name="samplecomponent"
    providers="ugcMapProvider-China400">
    <config class="com.supermap.sample.SampleComponentConfig">
        <param1>default</param1>
        ...
    </config>
</component>
```

SampleComponentConfig is the configuration class for SampleComponent, with param1 corresponding to the property of the SampleComponentConfig class. "rest" is the name of the interface to publish the Web service, "ugcMapProvider-China400" is the service provider used by the service component (Please refer to Service configuration file Structure).

In the implementation class of the service component, ComponentContextAware can be used to get the service component context, through which the configuration information of the service component can be got. The code is as follows:
@Component(providerTypes={MapProvider.class},optional=false, type = "SampleType")
public class SampleComponent implements ComponentContextAware{
    ...
    public void setComponentContext(ComponentContext context) {
        SampleComponentConfig
        config=context.getConfig(SampleComponentConfig.class);
        ...
    }
}

The SampleComponentConfig object corresponds to the <config
class="com.supermap.sample.SampleComponentConfig"/> configuration
item.

Getting service components

In SuperMap iServer, the service component can get service providers,
therefore getting the corresponding GIS capabilities. The code is as
follows:

@Component(providerTypes={MapProvider.class},optional=false, type = "SampleType")
public class SampleComponent implements ComponentContextAware{
    ...
    public void setComponentContext(ComponentContext context) {
        List<MapProvider> mapproviders=context.getProviders(MapProvider.class);
        ...
    }
}

Extension example

By extending and adding new service components, existing service
providers can be reused to implement richer capabilities.

In this example, a new service component PushpinComponent is
implemented using the MapProvider to provide the labeling capability
on the map.

Below is the implementation code of PushpinComponent:
PushpinComponent.java

public String GetImage(double pushpinX, double pushpinY) returns the URI of a map image. The position of (pushpinX, pushpinY) on the map is labeled with a red flag.

Note: The URI of the image contains the server IP and port number. In this example, the IP and port need to be processed in the service component. Here we replace the IP and port with localhost and 8090, as shown below:

```java
    String imageURI=imageURI.replace("{ip}", "localhost");
    imageURI=imageURI.replace("{port}", "8090");
```

Below is the implementation of the configuration class for the service component:

PushpinConfig.java

Place the Jar package after compiling in Java_HOME%/webapps/iserver/WEB-INF/lib.

As an example, PushpinComponent uses "China" in the China.smwu workspace for output, with ugcProvider-China400 being the service provider. The configuration in iserver-services.xml is as follows:

```xml
<component class="com.supermap.sample.component.PushpinComponent" interfaceNames="rest" name="map-pushpin" providers="ugcMapProvider-China400">
    <config class="com.supermap.sample.component.PushpinConfig">
        <mapName>China</mapName>
    </config>
</component>
```

interfaceNames="rest" indicates that the service component will be published as REST service. SuperMap iServer can publish any service component as REST service through the Restlet mechanism (refer to Publishing Domain Service Component as REST Service).

In the http://localhost:8090/iserver/services/map-pushpin/rest/domainComponents/PushpinComponent page, we can see resource list mapped by the PushpinComponent method. Click the
The service interface layer is the layer for the display of capabilities. By extending service interfaces, the formats in which services are provided by iServer can be enriched (please refer to ), therefore better satisfying the needs of users.

- Extension for Interface Layer

The response representation in XML format is as follows:

```xml
<string>
http://localhost:8090/iServer/output/temp/China_1f4xfa/30236220/41x31/16487x12495_1497621698.png?_t=1333697117750
</string>
```

We can get the map access URI, by accessing which the following map image will be got:
Besides, by extending existing service interfaces, such as JaxrsServletForJersey and RestServlet, to customize new REST resource implementations, the capabilities of the domain components can better match the service interfaces.

**Extension for Interface Layer**

The service interface layer is the layer for the display of capabilities, including the extensions of existing interfaces and new interfaces.

- Extending existing interface

The extension of existing interfaces in SuperMap iServer mainly focuses on REST service interfaces. The JaxrsServletForJersey and RestServlet service interface implementation classes correspond the Restlet and JAX-RS mechanisms respectively. The extensions include:

  - Defining REST resource Restlet and JAX-RS mechanisms
  - Extending encoder
  - Extending decoder

SuperMap iServer provides two implementation mechanisms for REST services: Restlet and JAX-RS. When creating domain resources, extensions based on JaxrsServletForJersey and RestServlet interfaces are different. Users need to choose the property mechanism.

  - Restlet-based domain resource extension
  - JAX-RS-based domain resource extension

- Defining new Service Interface

Below demonstrates the implementation of the service interface:

```java
@Interface(componentTypes =
{ com.supermap.services.components.Map.class }, optional = false, multiple =
false)
@Protocol(names={"BingMaps"})
public class BingMapsServlet extends HttpServlet implements
InterfaceContextAware {
```
Where

- The service interface implementation class needs to be a Servlet, which is implemented by being inherited from HttpServlet. InterfaceContextAware, which is the service interface context interface, is used to get the configuration information of the service components and service interfaces.

**Service Interface context**

**Getting Service Interface configuration**

**Extension example**

**Service Interface context**

In the implementation class of the service interface, InterfaceContextAware can be used to get the service interface context, through which the service components and the configuration information of the service interfaces can be got.

**Getting Service Interface configuration**

The configuration information of the service interfaces is in iserver-services-interfaces.xml. (refer to Structure of Service Configure File) Below is the configuration for one service interface:

```xml
<interface class="com.supermap.sample.SampleServlet" name="sampleinterface">
    <config class="com.supermap.sample.SampleServletSetting">
        <param1>default</param1>
        ...
    </config>
</interface>
```

SampleServletSetting is the configuration class of SampleServlet, with param1 corresponding to the property of the SampleServletSetting class.
In the implementation class of the service interface, InterfaceContextAware can be used to get the service interface context, through which the service components and the configuration information of the service interfaces in services interface configuration file can be got. The code is as follows:

```java
@Interface(componentTypes =
{ com.supermap.services.components.Map.class }, optional =
false, multiple = false)
@Protocol(names={"SampleInterface"})
public class SampleServlet extends HttpServlet implements
InterfaceContextAware {
    ...
    public void setInterfaceContext(InterfaceContext context) {
        //Get service interface configuration from the
        service interface context
        SampleInterfaceConfig sampleConfig =
        context.getConfig(SampleInterfaceConfig.class);
    }
}
```

SampleInterfaceConfig object corresponds to the `<config
class="com.supermap.sample.SampleServletSetting"/>` in the
interface.

If additional configuration is not needed for the service
interface, it is not necessary to define the service interface
configuration class.

**Getting service components**

SuperMap iServer publishes the service components as Web
services through the corresponding service interfaces. In the
implementation class of the resource, GIS capabilities of the
service components obtained from the service interface context.

```java
@Interface(componentTypes =
{ com.supermap.services.components.Map.class }, optional =
false, multiple = false)
@Protocol(names={"SampleInterface"})
```
public class SampleServlet extends HttpServlet implements InterfaceContextAware {
    ...
    public void setInterfaceContext(InterfaceContext context) {
        //Get service components from the service interface context (here it is MapComponent)
        List components = context.getComponents(Map.class);
        ...
    }
}

Extension example

The service interface determines how GIS capabilities are provided. By extending service interfaces, the formats in which the services are provided can be enriched, therefore better satisfying the needs of users.

In this example, a BingMaps image output interface is implemented. You can find the sample code in %SuperMap iServer_HOME%/samples/DSSE/BingMapsInterfaceSample.

According to the REST interface of BingMaps, we can get a map by accessing the URI below:

http://dev.virtualearth.net/REST/v1/Imagery/Map/Road?mapArea=30.0,100.0,50.0,120.0&mapSize=512,512&key=BingMapsKey

"Road" is the map name provided by BingMap, mapArea (latitude top, longitude left, latitude bottom, and longitude right) is the geographical bounds for map image output, mapSize is the map image size (width, height), and BingMapsKey is the key to access BingMaps, please refer to BingMaps Key (http://msdn.microsoft.com/en-us/library/ff428642.aspx).

Actually, the REST interface of BingMap has severeral parameters. As an demonstration, here we only adopt 3 parameters--imagerySet, mapArea, and mapSize.
Below is the BingMapServlet implementation class, which implements the handling of URI and calling of the Map component to get the map image:

BingMapsServlet.java

Place the Jar package after compiling in %SuperMap iServer_HOME%/webapps/iserver/WEB-INF/lib.

Add the BingMapsREST service interface, i.e., the <interface /> node in services.xml (refer to Structure of Service Configure File), as shown below:

```
<interface
class="com.supermap.sample.serviceinterface.BingMapsServlet"
name="bingmapsrest">
</interface>
```

Modify the configuration for the map-world component by adding bingmapsrest to the service interfaces bound with map-world, as shown below:

```
<component
class="com.supermap.services.components.impl.MapImpl"
interfaceNames="rest,wms111,wms130,wmts100,bingmaps"
name="map-world" providers="ugcMapProvider-World">
    <config class="com.supermap.services.components.MapConfig">
    </config>
</component>
```

Implement the GET request on http://localhost:8090/iserver/services/map-world/bingmapsrest/WorldMap?mapArea=10.0,100.0,50.0,180.0&mapSize=512,512 to get the map in png format, as shown below:
Besides, domain modules extended based on iServer can be integrated into iServer WebManager (Please refer to Integrating Domain Module into WebManager (To view: Home > Developer guide > Extending iServer > Domain Spatial Service Extension(DSSE) > Integrating domain module into iServer Manager)). Also, clusters can be built based on domain domain spatial services (please refer to Building DSS Cluster (To view: Home > Developer guide > Extending iServer > Domain Spatial Service Extension(DSSE) > Building DSS cluster)).
Other overall sample about domain service expansion please refer to: [3 layers](structure expansion overall sample). (To view: Home > Developer guide > Extending iServer > Domain Spatial Service Extension(DSSE) > 3-layer DSS extension)