Gears properties manager for jira

本页内容

- Function
- Scenario
- Parameters
  - Prefix
  - Key-Value
- Usage
- REST API
- API

Function

The main goal of Gears Properties Manager is to manage system level and global parameters;

In some enterprises with strong development ability, it is inconvenient to use some global variables in various app For example, in ScriptRunner, some customized scripts are often written into the workflow to obtain business data and send it to the third-party system. The third-party information may include some URLs, account names and passwords, or other parameters. When these parameters are written more, the workload of modifying these workflows will be a challenge for changing them.

Therefore, the use of this app is to define global parameters, which can be referenced to any required address. Just remember the key and value, and it will have the effect of one modification then all modifications.

At the same time, because of the global visual management of these parameters, it is more convenient for operation and maintenance. You only need to query in the system without asking the specific developers how to define them.

Scenario

For example, we have written a large number of scripts in the scheduled tasks, fields and postfusion of ScriptRunner. This script uses a user-defined field (development leader) and uses its value for many business processing.

In this case, we write multiple scripts and put them in different places. Usually, we will write the obtained value of this custom field in the script

```
customFieldManager.getCustomFieldObjectByName ("development leader")
```

In this scenario, if the development leader says that it is necessary to change a name to: "Dev leader", then we need to find all the scripts in the system that use this code to modify its value.
Or we can write it this way,

```
customFieldManager.getCustomFieldObjec (123456L)
```

In this case, it is also possible that this field is accidentally deleted and re created; Then its ID value will also change. You still need to find all the scripts in the system that use this code to modify its value.

Or in the following cases, the ID value or name is inconsistent in our production environment or test environment, which will also lead to the script debugged in the test environment, and the script obtained from the production environment can only be run by changing this value one by one.

For the above scenarios, using this plug-in, we can define a fixed variable in it. The value of this variable can be quickly managed visually. After modifying this value, all the changes in the system will take effect, which makes it convenient for us to manage these variables in a unified way, so as to achieve visual and global management of variable scripts used everywhere in our system.

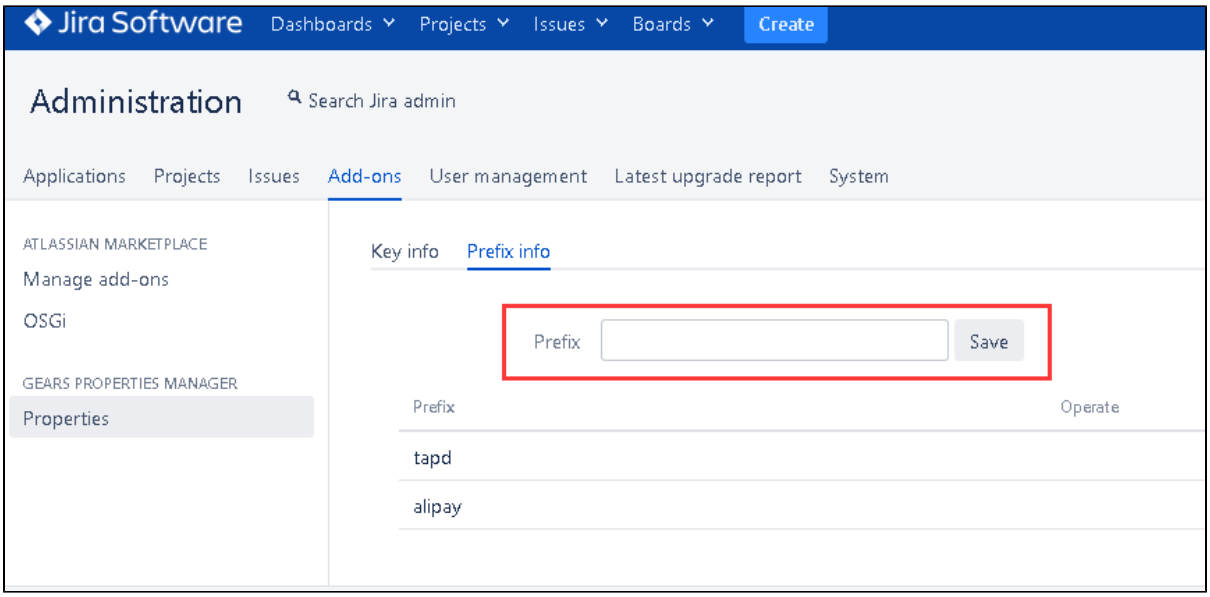
Parameters

In order to manage global parameters and facilitate the maintenance of managers, for example, if an administrator needs to write multiple parameters for interfacing with a third-party system, he can group by these parameters;

Therefore, the global grouping will be distinguished by prefix in this app; When managing parameters, you first need to determine which prefix for them .

Prefix

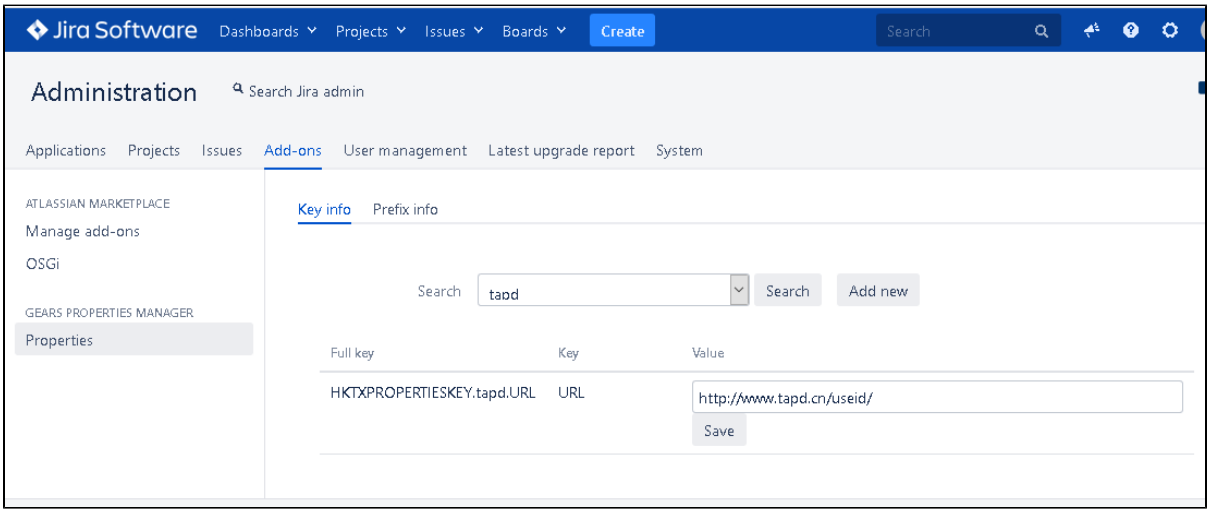
Prefix information is a definition of grouping global parameters, so prefix management is required.



You can navigate to the prefix information to add and manage prefixes. After saving, the prefix information defined in the system will be displayed.

Key-Value

Key-value is effective information provided for business use. It is allowed to add a key value pair to the corresponding prefix to obtain information through full key in the system.



You can select the prefix defined in the system to query all key value pairs included in this prefix. Of course, you can also modify a key value pair.

If you need to add a new key value pair to a prefix, you can add one. At this time, click "add" to pop up the add interface

Usage

When the key value pair is defined and newly added, its information can be obtained where needed for business processing.

Yes, you can use this kind of **com.atlassian.jira.config.properties. Applicationproperties** and use its **getText()** method.

键值对使用
String value = applicationProperties.getString($fullkey) ;

REST API

Get all key-values information for a prefix

- REST URL

/rest/gearsproperties/1.0/properties/prefix/{prefixkey}

- Method

get

- Example

/rest/gearsproperties/1.0/properties/prefix/ITDESK

```
{
"fullkey": "HKTXPROPERTIESKEY.ITDESK.StaffId",
"prefix": "ITDESK",
"key": "StaffId",
"value": "1123"
},
{
"fullkey": "HKTXPROPERTIESKEY.ITDESK.key1",
"prefix": "ITDESK",
"key": "key1",
"value": "value11"
},
{
"fullkey": "HKTXPROPERTIESKEY.ITDESK.key2",
"prefix": "ITDESK",
"key": "key2",
"value": "value22"
}
]
```


Get a value for a specific full key

- REST URL

/rest/gearsproperties/1.0/properties/fullkey/{fullkey}

- Method

get

- Example

/rest/gearsproperties/1.0/properties/fullkey/HKTXPROPERTIESKEY.ITDESK.StaffId

[


```
{
"fullkey": "HKTXPROPERTIESKEY.ITDESK.StaffId",
"value": "1123"
}
```


API

First, we define a variable value by this app (prefix is: SCRITRUNNER, KEY is : DEV_OWNER) then its fullkey is: HKTXPROPERTIESKEY.SCRITRUNNER.DEV_OWNER , value is: 123456

Your code may be like this

```
import com.atlassian.jira.config.properties.ApplicationProperties

String devOwnerCustomfieldId = applicationProperties.getString("HKTXPROPERTIESKEY.SCRITRUNNER.DEV_OWNER") ;

CustomField devOwnerCustomfield =  customFieldManager.getCustomFieldObject(Long.parseLong(devOwnerCustomfieldId));

ApplicationUser devOwnerUser = issue.getCustomFieldValue(devOwnerCustomfield);
```