EasyBbk Documentation:

data_center Unit Test

Author: Chen YiBai

Version: 1.0

Date: 2014 / 10 / 6

Introduction

This is a documentation to perform unit test of the unit “data_center” in the software “Easy BBK”. The test is used to check the validation of the unit “data_center” by examining the main functions and the outputs under different inputs in the unit. The examining covers all of the four main parts of the package and the service if provides.

"Easy BBK" is a trial to provide a standardized, visualized and user friendly access to bio-brick system by simplify and interassociate the searching, comparing, designing and uploading of bio-bricks. These four main functions constitute the "Easy BBK" client-side which is realized by pure java and supported by a remote server through the internet.

“data_center” is the connector between the GUI and the remote server, it connect to the online database and program in server, as well as downloading, storing and providing the data to the GUI part of “Easy BBK”. Having access to the source code or .jar file of the “data_center”, one can use most of the main functions of “Easy BBK” about searching, comparing and uploading through the API provided.

Test Overview

Here are the functions and their inputs and outputs of the test. The test is aimed at the unit “data_center”, which has no GUI, thus the test is proceeded under the cmd interface.

Function	Input	Output
Keyword searching and detail inquiring	Any string for keyword. Any string as the part_name of a biobrick.	The search method always return a list contains 0~n biobrick information wrapped into a class named BbkOutline in a list, the list is associated with the keyword input. The detail inquiring, however, returns a null if the input string in not a valid part_name in the database by SJTU-software. When the part_name is correct, the function will return the detail of a biobrick wrapped in a class named BbkDetail.
Search result filter and sort	Filtering accepts the constant string defined in class SearchResultList. Sorting requires no input. The sorting condition is determined in the name of the sorting functions.	Filtering should return a new list contains the biobrick that fits the filter condition of one of them if there are multiple filter condition. Sorting take place in situ, it has no return and change the list.
Blasting search	The sequence specified in the stdin or contains in a file. Another parameter to hint the function whether the first input is a sequence of a file path.	The search result list same as keyword searching, except that the blasting field in BbkOutline is filled for distinguishing and used in sorting.
Searching history managing and inquiring	Using API provided by “data_center” to inquire searching history.	The previous list or following list in the history. After a new searching, the item after the current should be dismissed.
Assign biobrick detail to compare slot	Any string as the part_name of a biobrick.	Return null if the part_name specified is not in the database, otherwise, returns the detail of the biobrick.
Sketch project operations: new project, close project	Newing a project needs no input, closing a project can specify a project name of pass null to close current project.	When newing a project, a project name will be auto generated, the name should not be the same as any of the current opened project. When closing the project, the current project should not change if not closing the current project, otherwise point to the first project as the current project.
Sketch project read and write XML file	The file path to save or load.	For saving, generate or update the XML file that contains all the component in the

		project. And for loading, reappear the contains of the project into the class named SketchProject.
Upload and reappear the uploaded biobrick	Upload pass the attributes packed in class BbkUpload, reappear uses the part_name and the part_id to find the uploaded biobrick.	After uploading a biobrick, a string represents the odd num will be returned. When reappearing the biobrick, if the biobrick is found, a BbkUpload instance will be returned, or null will be returned if the biobrick specified is not found.
Uploading part_name and sequence token validation check	Any string as the part_name. Any string as the sequence token.	Returns true if the part_name is not occupied, false if not. Returns true if the input sequence token is only consist of “a”, “t”, “c”, or “g”. False if some other character exists.
Uploading subpart and subscar validation check	Any string as the part_name of subpart and the scar_name of the subscar.	Returns the corresponding subpart and subscar if found, or null if not.

Test detail

Keyword searching and detail inquiring

Testing code:

```
SearchResultList list = dataCenter.searchCenter.search("GFP");
list.display();
System.out.println("List size: " + list.size());
for (int i = 0; i < list.size(); i += 10)
{   BbkDetail detail =
        dataCenter.searchCenter.getDetail(list.get(i).name);
    detail.display();
}
```

We used the common keyword “GFP” to perform the search. After that, get the detail in every 10 outlines.

Expected output:

A filled list with a size of positive number. When getting detail, all details can be found(no nulls).

Output:

The output list size is 2149 > 0, and the details are filled.

Search result filter and sort

Testing code:

```
BbkOutline bbkOutline =
    DatabaseConnector.getOutlineByName("BBa_B0034");
bbkOutline.displayFilteringConditions();

SearchResultList rawList =
    dataCenter.searchCenter.search("BBa_B0034");
System.out.println("\n\nFilter by the conditions that fits BBa_B0034: ");
rawList.filterByDeletedOrNot(false)
    .filterByDNAStatus(SearchResultList.Filter.DNAStatus.AVAILABLE)
    .filterByEnterYear(new int[]{2003, 2013})
    .filterByRelaseStatus(SearchResultList.Filter.ReleaseStatus.RELEA
SED).displayFilteringConditions();
```

```
System.out.println("\n\nSort by enter date: ");
rawList.sortByEnterDate(true);  rawList.displaySortingConditions();
System.out.println("\n\nSort by google items: ");
rawList.sortByGoogleQuoteNum(true);
rawList.displaySortingConditions();
System.out.println("\n\nSort by star num: ");
rawList.sortByAverageStars(true); rawList.displaySortingConditions();
System.out.println("\n\nSort by confirm: ");
rawList.sortByConfrimedTimes(true);
rawList.displaySortingConditions();
System.out.println("\n\nSort by total score: ");
rawList.sortByTotalScore(true); rawList.displaySortingConditions();
```

We use all the filter conditions that fits the biobrick previously get, and all the sort conditions to test these functions.

Expected output:

The biobrick get by name “BBa_B0034” survived through the filtering.

After sorting, the lists respectively contain newest, most google items, most star number, most confirmed times, highest total score in the head of the list.

Output:

“BBa_B0034” survived through the filtering.

After sorting, the lists respectively contain newest, most google items, most star number, most confirmed times, highest total score in the head of the list.

Blasting search

Testing code:

```
SearchResultList list;
list = dataCenter.searchCenter.blast
       ("blastInput", BlastingSearcher.MODE_INPUT_FILE_PATH);
list.sortByBlastResult(true);   list.displaySortingConditions();

list = dataCenter.searchCenter.blast
       ("tccaaagcttacgttaaacacccggctgacatcccggactacctgaaactgtccttccc"
                 + "ggaaggtttcaaatgggaacgtgttatgaacttcgaa",
       BlastingSearcher.MODE_INPUT_SEQUENCE);
list.sortByBlastResult(true);   list.displaySortingConditions();
BlastingSearcher.deleteLocalCacheFiles();
```

We test both two input methods: file path and sequence.

Expected oupput:

Both two lists have contents sorted by blasting score.

Output:

Output two lists, the item with the lowest eValue is on the top of the list.

Searching history managing and inquiring

Testing code:

```
System.out.println("The search history can roll back: "
              + dataCenter.searchCenter.canRollBack());
System.out.println("\t\t can go forward: "
              + dataCenter.searchCenter.canGoForward());

dataCenter.searchCenter.search("BBa_B0012");
dataCenter.searchCenter.search("BBa_B0011");
dataCenter.searchCenter.search("GFP BBa_B");
System.out.println("Searched 3 times, current list... ");
dataCenter.searchCenter.getCurrentRawSearchResultList().display();

System.out.println("The search history can roll back: "
              + dataCenter.searchCenter.canRollBack());
System.out.println("\t\t can go forward: "
              + dataCenter.searchCenter.canGoForward());
System.out.println("Roll back... ");
dataCenter.searchCenter.rollBack().display();
System.out.println("Roll back... ");
dataCenter.searchCenter.rollBack().display();
System.out.println("Go forward... ");
dataCenter.searchCenter.goForward().display();

System.out.println("Search again... ");
dataCenter.searchCenter.search("BBa_B0034");
dataCenter.searchCenter.getCurrentRawSearchResultList().display();
System.out.println("The search history can go forward: "
              + dataCenter.searchCenter.canGoForward());
```

We examined the movability of the cursor of the current page in the first time, after perform search for 3 times, (then roll back and go forward) and after a new search.

Expected output:

If the current page is in the begin / end of the list, the cursor cannot roll back / go forward. The previous list will be printed after roll back, and the following list after go forward. After a new search, the cursor cannot go forward.

Output:

The cursor cannot move at the beginning, the roll back / go forward operation can return its previous / following item. After new search, the cursor cannot go forward.

Assign biobrick detail to compare slot

Testing code:

```
dataCenter.compareCenter.assignDetail("BBa_I13545", 2).display();
```

We used the “BBa_I13545” and the slot No.2 to test.

Expected output:

The detail of the biobrick “BBa_I13545”, including the number of the properties.

Output:

The detail of the biobrick “BBa_I13545”.

Sketch project operations: new project, close project

Testing code:

```
dataCenter.sketchCenter.newProject();
dataCenter.sketchCenter.newProject();
dataCenter.sketchCenter.newProject();
dataCenter.sketchCenter.newProject();

String[] projectNames =
        dataCenter.sketchCenter.getAllProjectNames();
for (String name : projectNames)
    System.out.println("Project name: " + name);

System.out.println("Current: " +
        dataCenter.sketchCenter.currentProject.name);

System.out.println("Closing project1... ");
dataCenter.sketchCenter.closeProject(projectNames[0]);
```

```
System.out.println("Current: " +
        dataCenter.sketchCenter.currentProject.name);
System.out.println("Closing current... ");
dataCenter.sketchCenter.closeProject(null);
System.out.println("Current: " +
        dataCenter.sketchCenter.currentProject.name);
System.out.println("Closing current... ");
dataCenter.sketchCenter.closeProject(null);
System.out.println("Current: " +
        dataCenter.sketchCenter.currentProject.name);
dataCenter.sketchCenter.closeProject(null);
```

We created 4 projects and get all the project names, respectively close current project and non-current project to check the change of current project.

Expected output:

Four different project names printed, After first closing, the current project will not change, but it will change in the second and third closing.

Output:

Project names: “SketchProject1” ~ “SketchProject4”, After first closing, the current project does not change, but it changes in the second and third closing.

Sketch project read and write XML file

Testing code:

```
ArrayList<Point> curve = new ArrayList<Point>();
curve.add(new Point(11, 11));   curve.add(new Point(22, 22));
curve.add(new Point(33, 33));   curve.add(new Point(44, 44));
ArrayList<Integer> backBoneChildren = new ArrayList<Integer>();
backBoneChildren.add(1); backBoneChildren.add(6);

SketchProject project = dataCenter.sketchCenter.newProject();
System.out.println("Auto generated project name: " + project.name);

project.addComponent(new Label(0, "Lable text",
    new Point(5, 5), new Font("Times Roman", 10, 3), new Color(0, 0, 0)));
project.addComponent(new BioBrick(1, "Bba_B0034",
    BbkType.Sketch.BioBrick.PROMOTER, new Point(10, 10), null));
project.addComponent(new BioBrick(1, "Bba_B0012",
    BbkType.Sketch.BioBrick.PROMOTER, new Point(10, 10), null));
project.addComponent(new Protein(2, BbkType.Sketch.Protein.FACTOR,
```

```
    new Point(20, 20), Color.BLUE));
project.addComponent(new
    BackBone(3, new Point(50, 50), 50, backBoneChildren));
project.addComponent(new Relation(4, BbkType.Sketch.Relation.SUPPRESS,
    curve, new Color(50, 50, 50), 10));
project.addComponent(new BioVector(5,
    BbkType.Sketch.BioVector.BACTERIA, new Point(300, 300), 3));

project.saveIntoFile("testXML.xml");
project.loadFromFile("testXML.xml");

project.displayComponents();
```

We test all the components, add them into the project, save into the XML file and then reappear the list.

Expected output:

File “testXML.xml” generated in the current working directory, the list printed is the same as the components specified.

Ouptut:

File “testXML.xml” generated in the position, the list printed is the same as the components specified.

Upload biobrick into database by SJTU-software and reappear

Test code:

```
BbkUpload bbkUpload = new BbkUpload();
bbkUpload.setName("K1479001");
bbkUpload.setID();
dataCenter.uploadCenter.uploadAndGetOddNum(bbkUpload);
DatabaseConnector.displayTable(DBConsts.Table.MAIN_UPLOAD, 2);
dataCenter.uploadCenter.getBbkUploadByNameAndOddNum
       ("BBa_K1479001_EasyBbk", "201410115566901").display();
```

We manually newed a BbkUpload instance for the test, fill the part_name, upload and download it from the database.

Expected output:

Print a biobrick which has a name of “BBa_K1479001_EasyBbk” and an ID of “201410115566901”.

Output:

Print a biobrick which has a name of “BBa_K1479001_EasyBbk” and an ID of “201410115566901”. Other attributes are null.

Uploading part_name and sequence token validation check

Test code:

```
System.out.println("Name validation(K1479001): " +
    dataCenter.uploadCenter.isBbkNameNotOccupied("K1479001"));
System.out.println("Name validation(K1479010): " +
    dataCenter.uploadCenter.isBbkNameNotOccupied("K1479010"));
System.out.println("Sequence validation(atctgctagctgafacgt): " +
    dataCenter.uploadCenter.isSequanceValid("atctgctagctgafacgt"));
System.out.println("Sequence validation(atctgctagctgacacgt): " +
    dataCenter.uploadCenter.isSequanceValid("atctgctagctgacacgt"));
```

We use both validate and invalidate subpart and subscar input to test validation check.

Expected output:

false, true, false, true.

Output:

false, true, false, true.

Uploading subpart and subscar validation check

Test code:

```
System.out.println("Subpart validation(BBa_I13545): " +
    (dataCenter.uploadCenter.getSubpartForSequenceToken("BBa_I13545")
     != null));
System.out.println("Subpart validation(BBa_K1479010): " +
    (dataCenter.uploadCenter.getSubpartForSequenceToken("BBa_K1479010
    ") != null));
System.out.println("Subpart validation(RFC[10]): " +
    (dataCenter.uploadCenter.getSubscarForSequenceToken("RFC[10]")  !=
    null));
System.out.println("Subpart validation(RFC[1000]): " +
    (dataCenter.uploadCenter.getSubscarForSequenceToken("RFC[1000]")
    != null));
```

Expected output:

true, false, true, false.

Output:

true, false, true, false.

Evaluation

The unit test has proved that the backstage unit “data_center” can fulfill all the main service it should provide to the GUI and can get connect to the online server by the SJTU-software well. Some bugs did appear in the process of unit test, which had been debugged after some effort. The “data_center” appears to have no obvious bug for now.