Working with Results

Downloading and viewing your data from WebUI is convenient, but may not suitable for computer.

Working with ResultDB

Although resultdb is only designed for result preview, not suitable for large scale storage. But if you want to grab data from resultdb, there are some simple snippets using database API that can help you to connect and select the data.

```
from pyspider.database import connect_database
resultdb = connect_database("<your resutldb connection url>")
for project in resultdb.projects:
    for result in resultdb.select(project):
        assert result['taskid']
        assert result['url']
        assert result['result']
```

The `result['result']` is the object submitted by `return` statement from your script.

Working with ResultWorker

In product environment, you may want to connect pyspider to your system / post-processing pipeline, rather than store it into resultdb. It's highly recommended to override ResultWorker.

```
from pyspider.result import ResultWorker

class MyResultWorker(ResultWorker):
    def on_result(self, task, result):
        assert task['taskid']
        assert task['project']
        assert task['url']
        assert result
        # your processing code goes here
```

`result` is the object submitted by `return` statement from your script.

You can put this script (e.g., `my_result_worker.py`) at the folder where you launch pyspider. Add argument for `result_worker` subcommand:

`pyspider result_worker --result-cls=my_result_worker.MyResultWorker`

Or

```
{
  ...
  "result_worker": {
    "result_cls": "my_result_worker.MyResultWorker"
  }
  ...
}
```

if you are using config file. Please refer to Deployment

Design Your Own Database Schema

The results stored in database is encoded as JSON for compatibility. It's highly recommended to design your own database, and override the ResultWorker described above.

TIPS about Results

Want to return more than one result in callback?

As resultdb de-duplicate results by taskid(url), the latest will overwrite previous results.

One workaround is using `send_message` API to make a `fake` taskid for each result.

```
def detail_page(self, response):
    for li in response.doc('li').items():
        self.send_message(self.project_name, {
            ...
        }, url=response.url+"#"+li('a.product-sku').text())

def on_message(self, project, msg):
    return msg
```

See Also: apis/self.send_message