

The title is framed by a dashed white line. A dashed arrow on the right side points upwards, and a dashed arrow on the bottom left points to the right.

Developing Extensions for JupyterLab

Meet the instructors



Piyush Jain

AWS

Piyush is a software engineer working on JupyterLab



Alex Bozarth

IBM

Alex is a software engineer working on Elyra and JupyterLab



Martha Cryan

IBM

Martha is a software engineer working on Elyra and JupyterLab

Exploring Extensions

- What are extensions?
- Examples
 - DrawIO
 - Latex
 - Git
 - Spellchecker
 - Themes
- Installing prebuilt vs source



L^AT_EX



DrawIO

Anatomy of an Extension

- Extensions, plugins and widgets
- Code Walkthrough...
 - We will be using an example created from <https://github.com/jupyterlab/extension-cookiecutter-ts>
 - The example is loosely based on the in-depth example found in the jupyterlab documentation: https://jupyterlab.readthedocs.io/en/stable/extension/extension_tutorial.html

Code Exercise

1. Open `examples/tutorial_extension` dir in your IDE
2. Try adding a toolbar button that refreshes the image using the following hints:
 - The toolbar can be accessed from `MainAreaWidget.toolbar`
 - A `ToolBarButton` class can be found in `@jupyterlab/apputils`
3. An example answer can be found on the next slide if you get stuck

Code Exercise Example Answer

```
const button = new ToolbarButton({  
  icon: refreshIcon,  
  onClick: () => widget.load_image()  
});  
main.toolbar.addItem('refresh', button);
```


Debugging JupyterLab Extensions

- When is debugging useful
- Setting up for debugging
- Launching JupyterLab for debugging
- Setting breakpoints
- Other ways to debug

When is debugging useful

- To find errors in code
- Investigating unexpected results
- Understanding the code path
- Learning internals of other extensions

How to know something has gone wrong

- UI elements are missing
- Errors appear in the server log
- Errors appear in the browser console

Debugging in Visual Studio Code

- Instructions are in DEBUGGING.md
- Install the cookiecutter package
`pip install cookiecutter`
- Use the debug-config-cookiecutter
`cookiecutter ../debug-config-cookiecutter`
- Install the debug dependencies
`jlpmp install`
- Build the extensions
`jlpmp build`

Setting breakpoints

The screenshot shows the JupyterLab interface with a dark theme. The top bar includes a 'RUN ...' button, a 'jlab debug' dropdown, and a settings icon. The left sidebar contains icons for file explorer, search, recent files (9), debugger, console, variables, watch, and call stack. The main editor displays the file 'handlers.py' with the following code:

```
tutorial_extension > handlers.py > RouteHandler > get
1  import json
2
3  from jupyter_server.base.handlers import APIHandler
4  from jupyter_server.utils import url_path_join
5  import tornado
6
7  class RouteHandler(APIHandler):
8      # The following decorator should be present on all verb methods (head, get, post,
9      # patch, put, delete, options) to ensure only authorized user can request the
10     # Jupyter server
11     @tornado.web.authenticated
12     def get(self):
13         self.finish(json.dumps({
14             "data": "This is /tutorial-extension/get_example endpoint!"
15         }))
16
17
18 def setup_handlers(web_app):
19     host_pattern = ".*$"
20
21     base_url = web_app.settings["base_url"]
22     route_pattern = url_path_join(base_url, "tutorial-extension", "get_example")
23     handlers = [(route_pattern, RouteHandler)]
24     web_app.add_handlers(host_pattern, handlers)
25
```

A red dot indicates a breakpoint is set on line 13, at the start of the `self.finish` call. The breadcrumb at the top of the editor shows the navigation path: `tutorial_extension > handlers.py > RouteHandler > get`.

Launching JupyterLab for debugging

The screenshot displays the JupyterLab interface with a dark theme. The top bar shows the 'RUN' button and a dropdown menu with 'jlab debug' selected. The file explorer on the left shows a project structure with a '9' icon. The main editor area displays a Python file named 'handlers.py' with the following code:

```
1 import json
2
3 from jupyter_server.base.handlers import APIHandler
4 from jupyter_server.utils import url_path_join
5 import tornado
6
7 class RouteHandler(APIHandler):
8     # The following decorator should be present on all verb methods (head, get, post,
9     # patch, put, delete, options) to ensure only authorized user can request the
10     # Jupyter server
11     @tornado.web.authenticated
12     def get(self):
13         self.finish(json.dumps({
14             "data": "This is /tutorial-extension/get_example endpoint!"
15         })))
```

The left sidebar contains several panels: 'WATCH' (empty), 'CALL STACK' (showing 'jlab backend' and 'jlab frontend: localh...' as 'RUNNING'), and 'BREAKPOINTS' (showing 'Caught Exceptions' and 'Uncaught Exceptions' as 'RUNNING'). The bottom panel shows the 'TERMINAL' output with the following log entries:

```
[W 2022-07-08 09:54:18.084 ServerApp] 404 GET /static/lab/3496.ecb0e7fcc54191234ae6.js.map?v=ec
[W 2022-07-08 09:54:18.101 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
[W 2022-07-08 09:54:18.115 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
[W 2022-07-08 09:54:18.139 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
[W 2022-07-08 09:54:18.151 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
[W 2022-07-08 09:54:18.192 ServerApp] 404 GET /static/lab/714.d38baae8faccca175d4b.js.map?v=d38
[W 2022-07-08 09:54:18.203 ServerApp] 404 GET /static/lab/714.d38baae8faccca175d4b.js.map?v=d38
[I 2022-07-08 09:54:24.703 LabApp] Build is up to date
```


Debugging Front End Extension

The screenshot displays the Visual Studio Code interface with the following components:

- Editor:** Shows the file `TS index.ts` with the following code:

```
40 console.log(data);
41 }
42 .catch(reason => {
43   console.error(
44     `The tutorial_extension server extension appears to be missing.\n${reason}`
45   );
46 });
47
48 app.commands.addCommand('tutorial:open', {
49   // code to run when this command is executed
50   execute: () => {
51     const widget = new TutorialWidget();
52     const main = new MainAreaWidget({ content: widget });
53     main.title.label = 'Tutorial Widget';
54     main.title.icon = jupyterIcon;
55     main.title.caption = widget.title.label;
56
57     app.shell.add(main, 'main');
58   },
59 });
```
- Left Sidebar:**
 - LOCAL: activate** (selected):

```
> app: W {_started: true, _plugi...
> launcher: E {stateChanged: e, ...
> settingRegistry: g {schema: {...
  this: undefined
```
 - WATCH**: Empty.
 - CALL STACK**:
 - `_run_code` `runpy.py` 87:1
 - `_run_module_as_main` `run...`
 - `ThreadPoolExecutor-...` **PAUSED**
 - `jlab fro...` **PAUSED ON BREAKPOINT**
 - `activate` `src/index.ts` 48:5
 - `<anonymous>` `localhost:999...`
 - LOADED SCRIPTS**: Empty.
 - BREAKPOINTS**:
 - ☐ Caught Exceptions
 - ☐ Uncaught Exceptions
 - ☒ `handlers.py` `tutorial_extensi...` 13
 - ☒ `index.ts` `src` 48
 - BROWSER BREAKPOINTS**: Empty.
- Bottom Panel:**
 - PROBLEMS**: Empty.
 - OUTPUT**: Empty.
 - DEBUG CONSOLE**: Empty.
 - TERMINAL**:

```
.0.0.1) 6.72ms referer=None
[W 2022-07-08 10:03:19.862 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
.0.0.1) 5.70ms referer=None
[W 2022-07-08 10:03:19.878 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
.0.0.1) 7.19ms referer=None
[W 2022-07-08 10:03:19.893 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
.0.0.1) 6.21ms referer=None
[W 2022-07-08 10:03:20.066 ServerApp] 404 GET /static/lab/jlab_core.fc632a0f38747f007842.js.map
(127.0.0.1) 6.90ms referer=None
[W 2022-07-08 10:03:20.077 ServerApp] 404 GET /static/lab/jlab_core.fc632a0f38747f007842.js.map
(127.0.0.1) 6.37ms referer=None
```
 - JUPYTER**: Empty.

Debugging Server Extension

The screenshot shows the JupyterLab IDE interface. The top bar indicates the application is running in 'jlab debug' mode. The left sidebar contains navigation icons and a 'Locals' pane showing the current object: `> self: <tutorial_extension.handlers.RouteHandler>`. The 'WATCH' pane is empty. The 'CALL STACK' pane shows the current call stack, with the top frame being `get` in `handlers.py` at line 13:1. The 'BREAKPOINTS' pane shows two breakpoints: one in `handlers.py` at line 13 (caught exceptions) and one in `widact.ts` at line 8 (uncaught exceptions). The main editor displays the `handlers.py` file, which defines a `RouteHandler` class and a `setup_handlers` function. The `get` method is highlighted, and a breakpoint is set at line 13. The terminal window at the bottom shows the output of the application, including several HTTP requests and responses.

```
1 import json
2
3 from jupyter_server.base.handlers import APIHandler
4 from jupyter_server.utils import url_path_join
5 import tornado
6
7 class RouteHandler(APIHandler):
8     # The following decorator should be present on all verb methods (head, get, post,
9     # patch, put, delete, options) to ensure only authorized user can request the
10     # Jupyter server
11     @tornado.web.authenticated
12     def get(self):
13         self.finish(json.dumps({
14             "data": "This is /tutorial-extension/get_example endpoint!"
15         }))
16
17 def setup_handlers(web_app):
18     host_pattern = ".*$"
19
```

TERMINAL

```
.0.0.1) 7.39ms referer=None
[W 2022-07-08 10:02:08.469 ServerApp] 404 GET /static/lab/4429.c4f083ef6b6e29345fd4.js.map?v=c4
.0.0.1) 7.72ms referer=None
[W 2022-07-08 10:02:08.488 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
.0.0.1) 7.98ms referer=None
[W 2022-07-08 10:02:08.502 ServerApp] 404 GET /static/lab/7796.53c158c42e7f9697953b.js.map?v=53
.0.0.1) 7.64ms referer=None
[W 2022-07-08 10:02:08.663 ServerApp] 404 GET /static/lab/jlab_core.fc632a0f38747f007842.js.map
(127.0.0.1) 5.43ms referer=None
[W 2022-07-08 10:02:08.674 ServerApp] 404 GET /static/lab/jlab_core.fc632a0f38747f007842.js.map
(127.0.0.1) 5.38ms referer=None
```


Other ways to debug

- Front end extension
 - Use the browser directly to debug
- Server extensions
 - Python's command line debugger (pdb)

```
import pdb; pdb.set_trace()
```

- IPython pdb, a better alternative to pdb (pip install ipdb)

```
import ipdb; ipdb.set_trace()
```


Working on Your Own Extension

- Jupyter Server extension
- Theme extension
- Whatever you wanted to start on - or pick from [here](#)