Streamlining Contributions With EclipseTheia

paul.marechal@ericsson.com
EclipseCon Germany 2018
Talk

- Developing today
- Cloud Development
  - Eclipse Theia
- Theia Development
- Deployment
  - Workspace managers
Developing Today
Developing Today

- Checkout the project
- Install the tools / Configure the tools
- Install dependencies
- Start contributing
- Rinse and repeat (× projects × machines/users)
Developing Today - More

- Use the correct Operating System
- Dependency collisions between projects?
- Not your habitual workstation? Have fun recreating something of an environment.
What we would like to do

- Checkout the project
- Start contributing
What we would like to do

- Removing contribution overhead
- It is about changing habits and the ways of working
  - Develop anywhere
  - No compromises
Cloud Development
Not a new concept

- vi over ssh (remote development)
- Cloud9 (proprietary)
- GitLab IDE (minimalist)
- JDT/CDT over remote desktop (lag)
- ...
Theia

TypeFox

ERICSSON

redhat

arm

IBM
Theia Framework

- Written in TypeScript
- Runs in the browser
- Server-side
  - Node.js
- Runs on your desktop
  - Electron
Theia Features

• Code navigation
  – Code completion, go to definition, ...
  – LSP based

• Debugging
  – Breakpoints, step over/in/out
  – DAP based
LSP/DAP?

- **Language Server Protocol**
  - Deep language understanding factorization
- **Debug Adapter Protocol**
  - Generic protocol to drive a debug session
protected readonly resourceProvider: ResourceProvider;

@inject(KeybindingRegistry)
protected readonly keyBindingRegistry: KeybindingRegistry;

@inject(MessageService)
protected readonly messageService: MessageService;

@inject(OpenerService)
protected readonly opener: OpenerService;

export const OpenerService = Symbol('OpenerService');

/**
 * 'OpenerService' provide an access to existing openers.
 */
export interface OpenerService {
    /**
     * Return all registered openers.
     * Never reject.
     */
    getOpeners(): Promise<OpenHandler[]>;

    /**
     * Return all openers able to open the given URI for given options
     * ordered according their priority.
     * Never reject.
     */
}

@inject(KeymapsParser)
protected readonly parser: KeymapsParser;

protected readonly changeKeymapEmitter = new Emitter<void>();
onDidChangeKeymaps = this.changeKeymapEmitter.event;

protected resource: Resource;

@postConstruct()
protected async init() {
    await this.resourceProvider落地UI/Keymaps (src) withSchema(UserStoragesAll, SCHEM)
```typescript
import * as jsonparser from 'json-parser';
import { Emitter } from '@theia/core/lib/common/';

export interface KeybindingJson {
    command: string,
    keybinding: string,
}

@injectable()
export class KeymapsService {

    @inject(ResourceProvider)
    protected readonly resourceProvider: ResourceProvider;

    @inject(KeybindingRegistry)
    protected readonly keyBindingRegistry: KeybindingRegistry;

    @inject(MessageService)
    protected readonly messageService: MessageService;

    @inject(OpenerService)
    protected readonly openerService: OpenerService;
}
```
Theia Features

- Git support
- Terminal
- Tasks
- ...

16/40
Theia Framework

- Extensibility
  - Extensions
  - Plugins
Theia Application

- Composition of extensions
  - `$ npm install <theia-extension>`

- Two targets
  - "browser" (browser + Node.js server application)
  - "electron" (desktop application)
Theia Development
Theia Development

- Extensions (application build time)
  - Total control, rebinding of components, ...
- Plugins (runtime)
  - More rigid API via specific contribution points
Theia Extensions

• Published as npm packages
  - $ npm search @theia/*

• Dependency injection
  - Add/Override any Theia component
  - InversifyJs
```typescript
import { CommandContribution, MenuContribution } from '@theia/core/lib/common';
import { KeybindingContribution } from '@theia/core/lib/browser/keybinding';
import { KeymapsParser } from './keymaps-parser';
import './monaco-contribution';
import { WidgetFactory } from '@theia/core/lib/browser';
import { KeybindingWidget } from './keybindings-widget';
import './src/browser/style/index.css';

export default new ContainerModule(bind => {
  bind(KeymapsParser).toSelf().inSingletonScope();
  bind(KeymapsService).toSelf().inSingletonScope();
  bind(KeymapsFrontendContribution).toSelf().inSingletonScope();
  bind(CommandContribution).toService(KeymapsFrontendContribution);
  bind(KeybindingContribution).toService(KeymapsFrontendContribution);
  bind(MenuContribution).toService(KeymapsFrontendContribution);
  bind(KeybindingWidget).toSelf();
  bind(WidgetFactory).toDynamicValue(context => ({
    id: KeybindingWidget.ID,
    createWidget: () => context.container.get<KeybindingWidget>(KeybindingWidget).inSingletonScope();
  }));
});
```
```typescript
@injectable()
export class KeymapsService {

  @inject(ResourceProvider)
  protected readonly resourceProvider: ResourceProvider;

  @inject(KeybindingRegistry)
  protected readonly keyBindingRegistry: KeybindingRegistry;

  @inject(MessageService)
  protected readonly messageService: MessageService;

  @inject(OpenerService)
  protected readonly opener: OpenerService;

  @inject(KeymapsParser)
  protected readonly parser: KeymapsParser;

  protected readonly changeKeymapEmitter = new Emitter<void>();

  onChangeKeymaps = this.changeKeymapEmitter.event;

  protected resource: Resource;

  @postConstruct()
  protected async init() {
    this.resource = await this.resourceProvider(new URI('keymaps.json').withScheme('file').toString());
    this.reconcile();
    if (this.resource.onDidChangeContents) {
      this.resource.onDidChangeContents(() => this.reconcile());
    }
  }

  protected async reconcile(): Promise<void> {
    const keybindings = await this.parser.parseKeybindings();
    this.keyBindingRegistry.setKeymap(KeybindingScope.USER, keybindings);
    this.changeKeymapEmitter.fire(undefined);
  }
}
```
Theia Extensions

- Various contribution points for extenders
  - Menus/Commands/Openers
  - Views/Widgets
  - Language Clients
  - TextMate Grammars
  - And more...
- Everything in the framework can be made as an extension
Theia Plugins

- Self-contained *(no dependency resolution)*
- Runtime isolation
- Simple API
Theia Plugins

- API can be extended by Extensions
- VS Code extensions compatibility underway
Deployment
Cloud?

- Cloud development requires infrastructure
  - Networked machines
  - Administration (docker/kubernetes, ...)
  - Teams to operate
Docker

• Basic docker images available
  - github.com/theia-ide/theia-apps
Gitpod

- TypeFox cloud development platform
- Checkout PR branch and review it
- Theia-based

[gitpod.io](http://gitpod.io)
Eclipse Che

- RedHat cloud workspace manager
- Heavy-duty
- Theia-able

www.eclipse.org/che
Personal Use

- Remote
  - Workspace manager
  - Vanilla Theia application instance

- Local
  - Electron application
Conclusion
What does Theia change?

- Allows anyone to have its own IDE accessible from a browser
- Workspace managers can expose the IDE to work in the space
Contributing now

- Contribute from (almost) anywhere
- Avoid setup overhead
- Focus on task at hand
Contacts

• Presenter
  – Paul Maréchal: paul.marechal@ericsson.com

• Social
  – gadget.im/theia-ide/theia
  – github.com/theia-ide/theia/issues
References

- github.com/theia-ide/theia
- eclipse.org/che
- gitpod.io
Cue and Hay
(Q&A)
Evaluate the Sessions

Sign in and vote at eclipsecon.org

-1  0  +1