OSPP 项目申请书

项目名称：为 Solon 框架增加配套的轻量级监控工具（ID：23f4f0086）

项目导师：[数据删除]（[数据删除]）

申请人：[数据删除]（[数据删除]）

日期：2023/5/15

项目背景

Solon 背景

Solon 是一个类似 Spring 的后端应用开发框架，以“**更快、更小、更简单**”为目标，对比相似竞品可做到*启动快 5～10 倍；qps 高 2～3 倍；运行时内存节省 1/3 ~ 1/2；打包可以缩到 1/2 ~ 1/10*[1]。

需求项目背景

issue地址：**任务发放：实现 spring boot admin 类似的工具 · Issue #96 · noear/solon (github.com)**

为了满足 Solon 应用程序的性能监测需求，希望参照 **Spring Boot Admin** 项目，**为 Solon 框架增加配套的轻量级监控工具**，并提供简单的服务发现和信息查询功能[2]。

技术方法及可行性

Java 相关

作为我的第一编程语言，我对 Java 了解甚多，并在某公司有半年的 Java 程序开发实习经历。在**我的 GitHub**中有非常多的基于 Java 的应用程序，因此我相信我能够胜任相关开发。

企业级框架开发相关

我接触过与 Solon 类似的 Spring Boot 框架，并基于该框架开发过**一些程序**，大致了解 AOP 和 IOC 的思想，相信可以很快适应 Solon 应用程序的开发。

分布式服务相关

toktik 是一个基于 Go 的短视频后端程序，应用了分布式服务的相关内容，也同时支持服务发现功能，我参与了该程序的开发，并对分布式服务发现有所了解。

Solon 相关

我详细阅读了 Solon 在其官网上的 **入门教程**，对于 Solon 应用程序的开发已有所了解。

项目实现细节

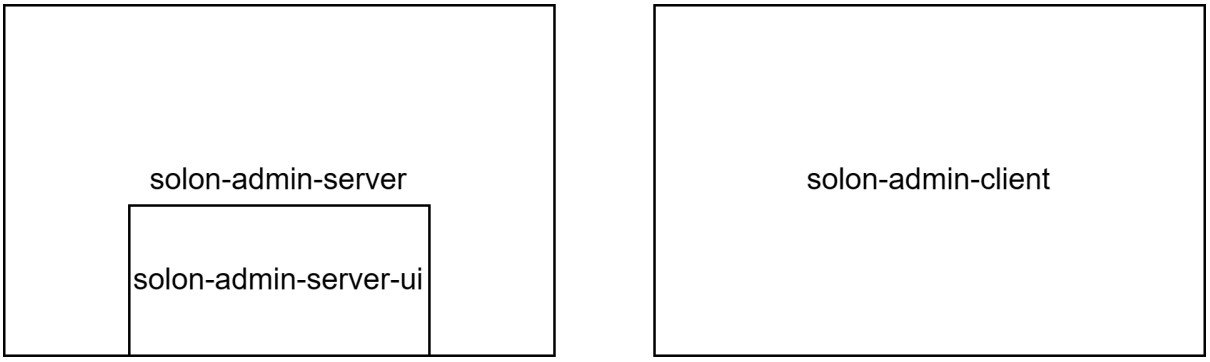
模块结构

Solon Admin 将被分为三个模块进行开发：

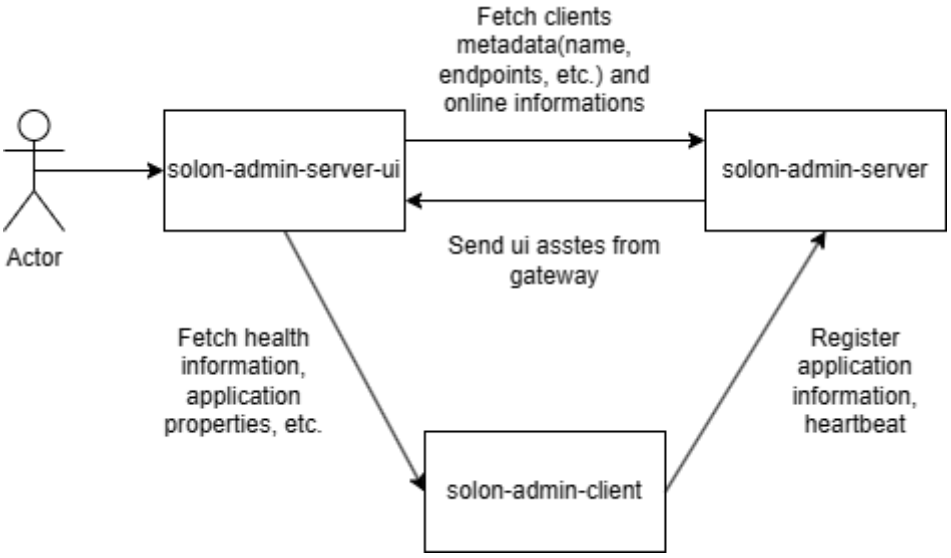
- 客户端（`solon-admin-client`），负责向服务端上报在线信息，收集客户端状态信息，同时提供客户端元数据给服务端，方便网页前端调取数据展示给用户。
- 服务端（`solon-admin-server`），负责服务发现，下发配置文件，将客户端元数据通过 Restful API 暴露给前端，同时作为 Gateway 发送前端资源文件。
- 网页前端（`solon-admin-server-ui`），负责展示服务端提供的数据。

其中，`solon-admin-server` 模块依赖于 `solon-admin-server-ui` 模块，前后端分离。

从模块结构角度，三个模块大致关系如下：



从用户角度，三个模块大致关系如下：



技术选型

三个项目采用如下技术选型进行开发：

`solon-admin-client`：

- Java 8
- Maven
- solon-web
- solon-cloud
- solon-api
- solon.boot.websocket
- solon.logging.simple
- Lombok

`solon-admin-server`：

- Java 8
- Maven
- solon-web
- solon-cloud
- solon-health-detector
- solon.logging-simple
- Lombok
- okhttp

`solon-admin-server-ui`：

- Vue 3（SPA Mode）
- Vite
- TypeScript
- HTML & CSS
- Maven (for submodule import only)
- arco-design
- yarn
- vue-i18n
- vue-router
- pinia

打包与构建

以上三个项目通过如下方式进行打包构建：

1. 通过 Maven 调用 `yarn build`，构建前端资源文件；构建 `solon-admin-server-ui`，将前端资源文件（`dist`）目录打包到 JAR 的 `META-INF` 目录中；
2. 构建 `solon-admin-server`，此时 `solon-admin-server-ui` 模块构件随 `solon-admin-server` 被一起打包；
3. 构建 `solon-admin-client`。

项目自动配置（AutoConfiguration）

首先，`solon-admin-server` 和 `solon-admin-client` 分别引入 `@EnableAdminServer` 和 `@EnableAdminClient` 注解，并标识 `@org.noear.solon.admin.server.config.Import` 注解，对于在主类上标识以上两个注解的 Solon 应用程序，指示 Solon 装配 Solon Admin 配置类：

```
/**
 * 指示此服务作为 solon-admin 客户端，此注解一般用在启动类上。
 */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Import(scanPackages = "org.noear.solon.admin.client")
public @interface EnableAdminClient {
}
```

其次，籍由 `AdminClientBootstrapConfiguration` 和 `AdminServerBootstrapConfiguration` 两个自动配置类，对客户端和服务端配置文件，系统环境进行检测，以判断是否应该拉起 Solon Admin Client 和 Solon Admin Server 服务。对于可以拉起的，向 Solon 装配 `MarkedClientEnabled` 和 `MarkedServerEnabled` 标记类。

```
@Bean
    public MarkedClientEnabled markedClientEnabled(@Inject(required = false)
IClientProperties clientProperties) {
        if (clientProperties == null || !clientProperties.isEnabled()) {
            log.error("Failed to enable Solon Admin client.", new
IllegalStateException("Could not enable Solon Admin client because none of the
properties has been configured correctly."));
            return null;
        }
        return new MarkedClientEnabled(clientProperties.getMode());
    }

    @Value
    public static class MarkedClientEnabled {
        String mode;

        public MarkedClientEnabled(String mode) {
            this.mode = mode;

            log.info("Solon Admin client has been successfully enabled in {}
mode.", this.mode);
        }
    }
```

最后，通过读取 `MarkedClientEnabled` 和 `MarkedServerEnabled` 标记类的注入状态，拉起其他关键服务（Conroller，Service 等）。

前端资源打包和路由

首先，从 `solon-admin-server-ui` 模块的 Maven 中引入 `com.github.eirslett:frontend-maven-plugin` 插件，自动安装 Node 和 yarn 环境并执行构建操作：

```
        <plugins>
                    <plugin>
                <groupId>com.github.eirslett</groupId>
                <artifactId>frontend-maven-plugin</artifactId>
                <version>1.12.1</version>

                <configuration>
                    <installDirectory>target</installDirectory>
                    <workingDirectory>src/main/vue</workingDirectory>
                </configuration>

                <executions>

                    <execution>
                        <id>install node and yarn</id>
                        <goals>
```

```
                            <goal>install-node-and-yarn</goal>
                        </goals>
                        <configuration>
                            <nodeVersion>v16.15.1</nodeVersion>
                            <yarnVersion>v1.22.19</yarnVersion>
                        </configuration>
                    </execution>

                    <execution>
                        <id>yarn install</id>
                        <goals>
                            <goal>yarn</goal>
                        </goals>
                        <configuration>
                            <arguments>install</arguments>
                        </configuration>
                    </execution>

                    <execution>
                        <id>yarn build</id>
                        <goals>
                            <goal>yarn</goal>
                        </goals>
                        <configuration>

 <yarnInheritsProxyConfigFromMaven>false</yarnInheritsProxyConfigFromMaven>
                            <arguments>build</arguments>
                            <environmentVariables>
                                <PROJECT_VERSION>${project.version}
</PROJECT_VERSION>
                            </environmentVariables>
                        </configuration>
                    </execution>

                </executions>
            </plugin>
        </plugins>
```

然后，引入 `org.apache.maven.plugins:maven-resources-plugin`，将以上构建好的资源文件由 `target/dist` 打包至 `META-INF/solon-admin-server-ui` 目录：

```
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.3.1</version>
                <configuration>
                    <nonFilteredFileExtensions>
```

```
 <nonFilteredFileExtension>woff</nonFilteredFileExtension>

 <nonFilteredFileExtension>ttf</nonFilteredFileExtension>

 <nonFilteredFileExtension>woff2</nonFilteredFileExtension>

 <nonFilteredFileExtension>eot</nonFilteredFileExtension>

 <nonFilteredFileExtension>swf</nonFilteredFileExtension>

 <nonFilteredFileExtension>ico</nonFilteredFileExtension>

 <nonFilteredFileExtension>png</nonFilteredFileExtension>
                    </nonFilteredFileExtensions>
                    <includeEmptyDirs>true</includeEmptyDirs>
                </configuration>
            </plugin>
        </plugins>
        <resource>
                <directory>target/dist</directory>
                <targetPath>META-INF/solon-admin-server-ui</targetPath>
                <filtering>false</filtering>
        </resource>
```

最后，由 `solon-admin-server` 模块创建 Controller，从 classpath 中拉取对应目录打包好的前端资源文件并发送：

```
@Controller
public class UIController {

    @Get
    @Mapping("/")
    @Produces(MimeType.TEXT_HTML_VALUE)
    public InputStream index() throws IOException {
        InputStream stream = this.getClass().getResourceAsStream("/META-
INF/solon-admin-server-ui/index.html");
        if (stream == null)
            throw new IOException("Could not find index.html from solon-admin-
server-ui, please check if the solon-admin-server-ui dependency is added");
        return stream;
    }

    @Get
    @Mapping("/**")
    public void resources(Context ctx) throws IOException {
        if (ctx.path().equals("/index.html")) {
            ctx.redirect("/");
            return;
```

```
        }

        InputStream stream = this.getClass().getResourceAsStream("/META-
INF/solon-admin-server-ui" + ctx.path());
        if (stream == null) {
            ctx.status(404);
            return;
        }


 Optional.ofNullable(StaticMimes.findByFileName(ctx.path())).ifPresent(ctx::co
ntentType);

        ctx.output(stream);
    }

}
```

国际化

国际化支持由 `solon-admin-server-ui` 模块从前端直接提供，使用 `vue-i18n` 库进行支持，通过 Hooks 的方式引入组件，并合并 `arco-design` 组件库的本地化模块：

```
export default function useLocale() {
    const i18n = useI18n();
    const allLocales = i18n.messages;
    const currentLocale = computed(() => {
        const customLocale = allLocales.value[i18n.locale.value];
        switch (i18n.locale.value) {
            case 'en_US':
                return {
                    ...enUS,
                    ...customLocale
                };
            case 'zh_CN':
            default:
                return {
                    ...zhCN,
                    ...customLocale
                };
        }
    });
    const changeLocale = (value: string) => {
        if (i18n.locale.value === value) return;
        i18n.locale.value = value;
        localStorage.setItem('solon-admin-locale', value);
        Message.success(i18n.t('header.action.locale', {language:
i18n.t('language')}));
```

```
    };
    return {
        currentLocale,
        changeLocale,
        allLocales,
    };
}
```

项目开发时间计划

第一阶段（23/7/1 - 23/7/15）

前后端基础代码架构编写和实现。

第二阶段（23/7/16 - 23/8/5）

最小可行 MVP 实现，包括前端项目打包和路由，后端应用程序注册和反注册等功能。

第三阶段（23/8/6 - 23/9/30）

完整项目产出，包括分布式服务发现和配置文件拉取的实现，完整的 UI 功能，完整的监控报表和应用程序元数据显示，应用程序事件流。

---

1. 更快、更小、更简单 (noear.org) ↩

2. OSPP 2023 (summer-ospp.ac.cn) ↩