Spring - boot

Feature 1: Spring Command Line Interface

Spring-boot provides a user-friendly interface. To figure out how to generate the structure and the prompt information, we first walk into the main function `SpringCli.java`. This function calls `CommandRunner` as an entry to the command operations. It uses `addCommands()` to add commands to the interface. This function takes commands as the parameter, and if the command is `null`, it shows the prompt that " Commands can not be null". What's more, this function has been overloaded so that user could add an Iterable object to add commands. The command is the instance of `Command` class. `Command` class is an interface, and implemented by `AbstractCommand` which is then extended by `HelpCommand`, `HintCommand`, `VersionCommand` and etc.

In the `SpringCli.java`, `addServiceLoader()` is called to load `CommandFactory` as service. This class is implemented by `DefaultCommandFactory` and only have one method, `getCommands()` helps return all default commands. By far, all commands are running as service. There are other types of commands listed and all of them are extended from `AbstractCommand.java` . After we looked at these definition, we held the idea that these are different types of commands and we don't need to totally understand the implementation of these command to understand the command line interface.

```
static {
    List<Command> defaultCommands = new ArrayList<>();
    defaultCommands.add(new VersionCommand());
    defaultCommands.add(new RunCommand());
    defaultCommands.add(new GrabCommand());
    defaultCommands.add(new JarCommand());
    defaultCommands.add(new WarCommand());
    defaultCommands.add(new InstallCommand());
    defaultCommands.add(new UninstallCommand());
    defaultCommands.add(new InitCommand());
    defaultCommands.add(new EncodePasswordCommand());
    DEFAULT_COMMANDS = Collections.unmodifiableList(defaultCommands);
  }
```

And the `runner` mentioned before has other functions to setup the command line interface. `setOptionCommands()` and `setHiddenCommands()` set two variables' value in the `CommandRunner.java` .

What's more, `runner` could deal with exceptions as well. `runAndHandleErrors()` takes command input as parameter and return 0 if something went wrong or return 1 otherwise. To further understand the exception handle mechanics, we dive into this part of codes. The interface allows users to input a debug flag, so the function call `removeDebugFlags()` to remove this flag and pass the new String array to `argsWithoutDebugFlags`. If the length is changed, we could know that there is a debug info in the input and we set `isDebug = true`. Then it starts to deal user's input command by `run()` . Depends on commands, `run()` could set different `ExitStatus`.

`ExitStatus` is a class defined in command/status, and when a command is run, that command would return an `ExitStatus` object. `result` has an attribute `isHangup`, and if that is marked and program could not deal with it, it would return 0 , and back in `SpringCli.java`, the program will not end normally, but pop up some error message.

```
try {
        ExitStatus result = run(argsWithoutDebugFlags);
        // The caller will hang up if it gets a non-zero status
        if (result != null && result.isHangup()) {
            return (result.getCode() > 0) ? result.getCode() : 0;
        }
        return 0;
    }
    catch (NoArgumentsException ex) {
        showUsage();
        return 1;
    }
    catch (Exception ex) {
        return handleError(debug, ex);
    }
```

Folder	File	Method	Why?	Priority	Notes
command/util	SpringCli.java	Constructor	main entry of the function	5	
command/status	CommandRunner.java	addCommands()	Used to add prompt	5	
command/status	CommandRunner.java	addServiceLoader()	Run as service	3	
command/status	Command.java	getName()	interface	3	
command/status	AbstractCommand.java	getName()	interface	3	
command/status	CommandService.java	getCommands()	interface	2	
command/util	DefaultCommandFactory.java	getCommands()	implement	3	
command/status	CommandRunner.java	setOptionCommands()	set option commands	4	
command/status	CommandRunner.java	setHiddenCommands()	set hidden commands	3	
command/status	CommandRunner.java	runAndHandleErrors()	deal with errors	5	
command/status	ExitStatus.java	isHangup()	set status	5	

Folder	File	Method	Relevant?	Relevant how?	Confidence	Notes
command/util	SpringCli.java	Constructor	Yes	Entrance of the code	5	
command/status	CommandRunner.java	addCommands()	Yes	add default commands	5	
command/status	CommandRunner.java	addServiceLoader()	Yes	run as services	3	
command/status	Command.java	getName()	Yes	return command name	3	
command/status	AbstractCommand.java	getName()	Yes	return command name	3	
command/status	CommandService.java	getCommands()	Yes	this is the interface	3	
command/util	DefaultCommandFactory.java	getCommands()	Yes	return all of the default commands	2	
command/status	CommandRunner.java	setOptionCommands()	Yes	show option commands	2	
command/status	CommandRunner.java	setHiddenCommands()	Yes	show hidden commands	5	
command/status	CommandRunner.java	runAndHandleErrors()	Yes	handle errors	5	
command/status	ExitStatus.java	isHangup()	Yes	get command status	5	

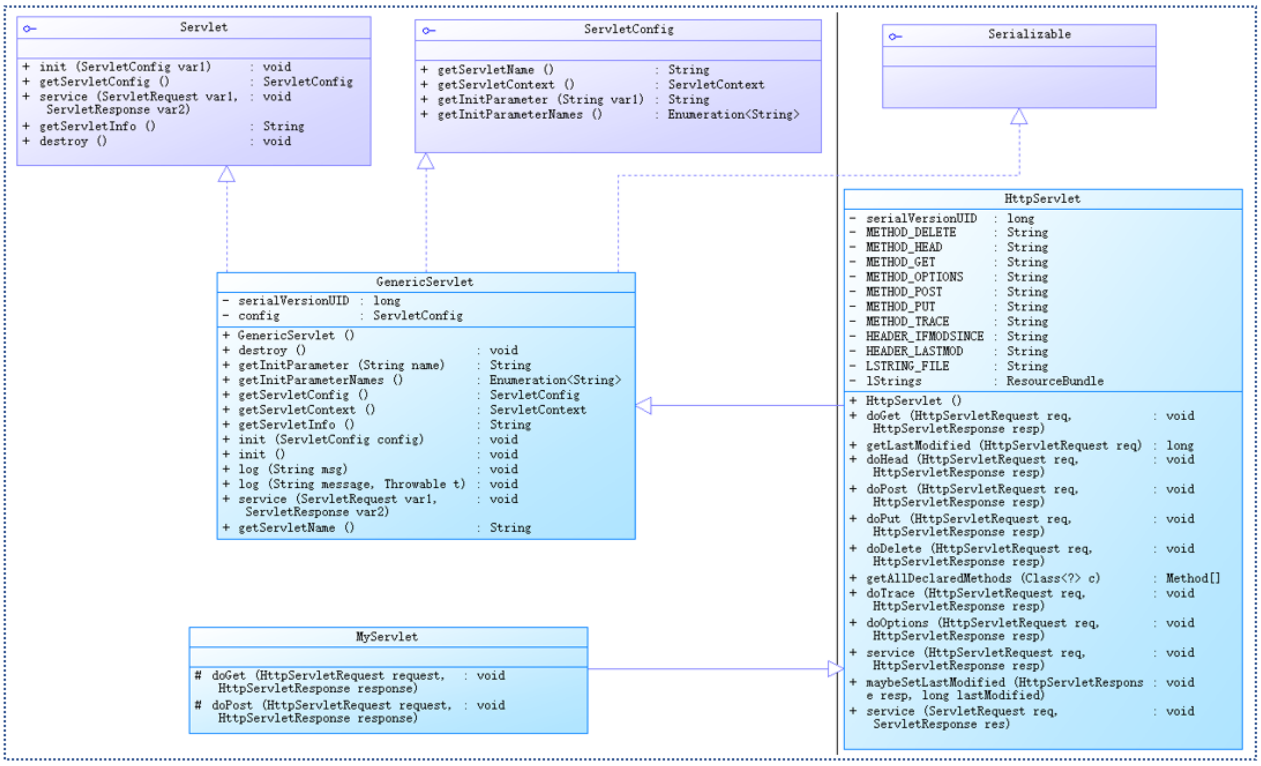
Feature 2: How Http request is proceed? ---Servlet

Spring-boot internally assembled servlet which provide the function to proceed the http request such as "GET", "POST", "PUT" , etc. I will analyze a standard flow of how a http request was proceeded in the Spring-boot.

First, we can find a `Servlet.class` in the source code, which is an interface and provides some abstract functions. Then we can find `GenericServlet.class` implements the interface, but it is still an abstract class which exposes some abstract functions. Continuously, we find `HttpServlet.class` which is a sub-class of `GenericServlet`.

In the `HttpServlet.class`, we can find two important method `doGet` and `doPost`, which are the functions to handle the "Get" request and "Post" request.

The class diagram is as below:


Servlet
+ init (ServletConfig var1) : void
+ getServletConfig () : ServletConfig
+ service (ServletRequest var1, ServletResponse var2) : void
+ getServletInfo () : String
+ destroy () : void
ServletConfig
+ getServletName () : String
+ getServletContext () : ServletContext
+ getInitParameter (String var1) : String
+ getInitParameterNames () : Enumeration<String>
Serializable
GenericServlet
- serialVersionUID : long
- config : ServletConfig
+ GenericServlet ()
+ destroy () : void
+ getInitParameter (String name) : String
+ getInitParameterNames () : Enumeration<String>
+ getServletConfig () : ServletConfig
+ getServletContext () : ServletContext
+ getServletInfo () : String
+ init (ServletConfig config) : void
+ init () : void
+ log (String msg) : void
+ log (String message, Throwable t) : void
+ service (ServletRequest var1, ServletResponse var2) : void
+ getServletName () : String
HttpServlet
- serialVersionUID : long
- METHOD_DELETE : String
- METHOD_HEAD : String
- METHOD_GET : String
- METHOD_OPTIONS : String
- METHOD_POST : String
- METHOD_PUT : String
- METHOD_TRACE : String
- HEADER_IFMODSINCE : String
- HEADER_LASTMOD : String
- LSTRING_FILE : String
- lStrings : ResourceBundle
+ HttpServlet ()
+ doGet (HttpServletRequest req, HttpServletResponse resp) : void
+ getLastModified (HttpServletRequest req) : long
+ doHead (HttpServletRequest req, HttpServletResponse resp) : void
+ doPost (HttpServletRequest req, HttpServletResponse resp) : void
+ doPut (HttpServletRequest req, HttpServletResponse resp) : void
+ doDelete (HttpServletRequest req, HttpServletResponse resp) : void
+ getAllDeclaredMethods (Class<?> c) : Method[]
+ doTrace (HttpServletRequest req, HttpServletResponse resp) : void
+ doOptions (HttpServletRequest req, HttpServletResponse resp) : void
+ service (HttpServletRequest req, HttpServletResponse resp) : void
+ maybeSetLastModified (HttpServletResponse resp, long lastModified) : void
+ service (ServletRequest req, ServletResponse res) : void
MyServlet
# doGet (HttpServletRequest request, HttpServletResponse response) : void
# doPost (HttpServletRequest request, HttpServletResponse response) : void

- ***Q1: When these two functions `doGet` and `doPost` are called?***

> So we need to continue to find where these two functions are called. Then we find the `service` function.

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        String method = req.getMethod();

        if (method.equals(METHOD_GET)) {
            long lastModified = getLastModified(req);
            if (lastModified == -1) {
                // servlet doesn't support if-modified-since, no reason
                // to go through further expensive logic
                doGet(req, resp);
            } else {
                long ifModifiedSince = req.getDateHeader(HEADER_IFMODSINCE);
                if (ifModifiedSince < lastModified) {
                    // If the servlet mod time is later, call doGet()
                    // Round down to the nearest second for a proper compare
                    // A ifModifiedSince of -1 will always be less
                    maybeSetLastModified(resp, lastModified);
                    doGet(req, resp);
                } else {
                    resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
                }
            }

        } else if (method.equals(METHOD_HEAD)) {
            long lastModified = getLastModified(req);
            maybeSetLastModified(resp, lastModified);
            doHead(req, resp);

        } else if (method.equals(METHOD_POST)) {
            doPost(req, resp);

        } else if (method.equals(METHOD_PUT)) {
            doPut(req, resp);

        } else if (method.equals(METHOD_DELETE)) {
            doDelete(req, resp);

        } else if (method.equals(METHOD_OPTIONS)) {
            doOptions(req,resp);

        } else if (method.equals(METHOD_TRACE)) {
            doTrace(req,resp);

        } else {
            String errMsg = lStrings.getString("http.method_not_implemented");
            Object[] errArgs = new Object[1];
            errArgs[0] = method;
            errMsg = MessageFormat.format(errMsg, errArgs);

            resp.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED, errMsg);
        }
    }
```

In this function, we can see it will first parse the type of Http request in the function of `req.getMethod()` , and distribute it to the handling method like `doGet`, `doPost` , etc.

- ***Q2: When the `service` function is called?***

> We can find a class called `ServiceHandler.class` and there is a lifecycle function called `doHandler`

```
@Override
    public void doHandle(String target, Request baseRequest,
HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        ServletHolder servletHolder =
(ServletHolder)baseRequest.getUserIdentityScope();
        FilterChain chain = null;

        // find the servlet
        if (target.startsWith("/"))
        {
            if (servletHolder != null && _filterMappings != null &&
_filterMappings.length > 0)
                chain = getFilterChain(baseRequest, target, servletHolder);
        }
        else
        {
            if (servletHolder != null)
            {
                if (_filterMappings != null && _filterMappings.length > 0)
                {
                    chain = getFilterChain(baseRequest, null, servletHolder);
                }
            }
        }

        try
        {
            if (servletHolder == null)
                notFound(baseRequest, request, response);
            else
            {
                // unwrap any tunnelling of base Servlet request/responses
                ServletRequest req = request;
                if (req instanceof ServletRequestHttpWrapper)
                    req = ((ServletRequestHttpWrapper)req).getRequest();
                ServletResponse res = response;
                if (res instanceof ServletResponseHttpWrapper)
                    res = ((ServletResponseHttpWrapper)res).getResponse();

                // Do the filter/handling thang
                servletHolder.prepare(baseRequest, req, res);

                if (chain != null)
                    chain.doFilter(req, res);
                else
                    servletHolder.handle(baseRequest, req, res);
            }
        }
        finally
        {
            if (servletHolder != null)
                baseRequest.setHandled(true);
        }
    }
```

In this function:

1. Firstly, it uses `target` to match the corresponding `servletHolder`

2. Secondly, it checks whether there are filters need to handle. If true, it needs to handle the filter first, otherwise it can proceed the http request.
3. Thirdly, we see `servletHolder.handle(baseRequest, req, res)`, and we can go into it.
4. Finally, we see the `service` function is called here.

```
public void handle(Request baseRequest,
                       ServletRequest request,
                       ServletResponse response)
        throws ServletException,
        UnavailableException,
        IOException
    {
        try
        {
            Servlet servlet = getServlet();
            if (servlet == null)
                throw new UnavailableException("Servlet Not Initialized");
            servlet.service(request, response);
        }
        catch (UnavailableException e)
        {
            makeUnavailable(e).service(request, response);
        }
    }
```

- ***Q3: What does it do in the function of `doGet` in `HttpServlet`?***

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        String protocol = req.getProtocol();
        String msg = lStrings.getString("http.method_get_not_supported");
        if (protocol.endsWith("1.1")) {
            resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, msg);
        } else {
            resp.sendError(HttpServletResponse.SC_BAD_REQUEST, msg);
        }
    }
```

- ***Q4: Why here send an error response in `doGet`?***

  We can see `HttpServlet` is still an abstract class with a lot of abstract functions. So it only provides some default implementations to the external. When developers needs to handle a http request, they need to extend the `HttpServlet` and provide their own implementations. So here it only return an error response. After developer implement their own Servlet, this function will be overridden.

Folder	File	Method	Why	Priority	Notes
servlet.http	HttpServlet.java	doGet	Proceed "Get" request	5	
servlet.http	HttpServlet.java	service	Find where `doGet` is called	5	
servlet	ServletHandler.java	doHandle	Find where `service` is called	5	
servlet	ServletHolder.java	handle	Find where `service` is called	3	

Folder	File	Method	Relevant	Relevant how	Confidence	Notes
servlet.http	HttpServlet.java	doGet	True	Proceed "Get" request	5	
servlet.http	HttpServlet.java	service	True	`doGet` is called	5	
servlet	ServletHandler.java	doHandle	True	`service` is called	5	
servlet	ServletHolder.java	handle	True	`service` is called	5	