SKETCH OF SOME FEATURES OF THE CONTEXT DESIGN

ADAM WALSH

1. Prototype shortcuts

- Data handling still done via the old system, no internal caching.
- No caching or nice handling of the user or server custom defined commands.
- The implementation of the prefix checking gives the bot a lot of work to do when people type for instance the global prefix in a server with a different custom prefix set. Not sure if this can be avoided.

2. General Notes

- To handle user defined commands, we shall use a possibly temporary method, prefixing the userid or serverid in front of the command name to give it a unique identification key. If a server defines the command "hello" for instance, the bot writes the name of the command as "server1235432:hello". Use a separate command cache for the server and user defined commands, which gets checked after the global command cache.

3. Bot

3.1. **Initialisation of Bot.**

3.2. **onmessage event handling.**

3.2.1. *Implementation (prototype).*

- Iterate through prefix list, for each prefix check whether message starts with that prefix.
- Note that unless exists in the prefix list, we want to explicitly ignore the message if it contains . This should save some work for the bot.
- If we find a prefix in the cache, instantiate a message context and ask it what the valid prefixes are, then make sure that we match those.
- If this fails, and if the mention prefix flag is on for the bot, check whether the message starts by mentioning the bot.
- If the message starts by mentioning the bot, then again instantiate a message context, and set mentioned to True.
- If a valid prefix is found, set the prefix variable.
- Now strip out the prefix from the message and grab the first word, which should be the command
- Check the command against the user defined commands and the server defined commands disctionaries, making sure we include a prefix of the user or serverid, in that order. Then check the global command dictionary. If the command does not appear in any of them, if the mentioned flag is on, reply to the message with a meangingless ack, otherwise terminate.
- If we do find a command, then grab the actual command from the dictionary, instantiate a commandcontext, and execute the commands run method in a try/catch block.

3.3. **Attributes.**

- Prefix_list:
  Cached list of valid prefixes. Load them in the bot initialisation in on_ready.
-

¡++¿

4. Context

5. Message Context

5.1. **Methods.**

- *char** get_prefixes():*
  Returns a list of current valid prefixes for the message. Takes into account the server prefix, the global prefix if the server prefix isn't set, and the user prefix.

6. Server configuration

6.1. **Config settings:**

- Server local prefix(s) (overrides global prefix). Server prefix is treated as global prefix if not set.

7. User configuration

7.1. **Config settings.**

- User bot prefix (does not override server prefix)

8. Data storage

8.1. **Notes.**

- Most data storage will be handled by the database interface.
- Let the database interface handle the caching of common things? For instance, when we retrieve some data, have a cache=False setting which stores the data in memory, next time that request is made, it retrieves it from memory instead. When the data is updated it updates both the cached copy and the actual copy. To do this, we will want some sort of structure held in memory with the same form as the stored data, but obviously a lot smaller. When we request the server prefix for a server for the first time, it gets cached into this data structure. Next time we request it, it is loaded from the structure instead of from the probably slower database.