Servers

Kahuaランタイムシステムは、協調して動作する、複数のサーバープロセスにより構成される。

・supervisor process (`kahua-spvr`) は常に立ち上がっている。

・他のアプリケーションサーバ(app server)はすべて、`kahua-spvr`の子プロセスとなる。

・`kahua-spvr`はサーバーソケットをよく知られた箇所に開いておく (現在はunixソケットで、`/tmp/kahua/kahua`)

・app serverのサーバソケット名は立ち上げる度に変わる。 `kahua-spvr`が全てのapp serverのソケット名を把握している。

・異なる種類のapp serverを並行して立ち上げることができる。

・`kahua-spvr`はどの種類のserverがどのソケットでlistenしているかを把握している

・開発バージョンと安定バージョンのapp serverをあげておくとか。

・異なる種類のサービスを別のapp serverにやらせるとか。

・将来的にはapp serverを別マシンに振って負荷分散なんかも。

クライアントとの通信

ここでの「クライアント」とは、cgiブリッジとかサーバ管理用のプログラムのこと。

・クライアントが初めて接続する際 (session initiating request) は、ソケット名がわかっている`kahua-spvr`に接続する。

1.`kahua-spvr`はapp serverを選んで、リクエストをフォワードする。

・クライアントはx-kahua-workerヘッダを用いて通信したいアプリケーションサーバのタイプを指定することができる。同タイプのアプリケーションサーバが複数立ち上がっている場合は `kahua-spvr` はその中から適当に選ぶ。

2.app serverは`kahua-spvr`にリプライを返す。リプライにはセッションキー?が含まれる。

3.`kahua-spvr`はクライアントにリプライをフォワードする。

4.クライアントがcgiブリッジの場合は、さらにセッションキー?を form

のパラメータやcookieにエンコードして、httpd経由でWebクライアントへと返す。

・ひとたびセッションが開始されたら…

1.クライアントはセッションキー?から、どのapp serverと通信しているかを知り、直接app serverへとリクエストを投げる。

2.app serverはクライアントに、リプライおよび更新されたセッションキー?を返す。

3.セッションが終了した場合(つまり、クライアントの状態を保持する必要がなくなった場合)は、app serverはセッションキー無しでリプライを返してもよい。その場合、次にクライアントがアクセスしてくると、セッションキーが無いので kahua-spvrに接続することになる。

この方式のメリットは、セッション開始後はkahua-spvrをバイパスするのでkahua-spvrがボトルネックにならないこと。ただアーキテクチャ的には、セッションキーをapp serverによって使われる opaqueな部分と、クライアントがapp serverを見付けられるようにする部分とを持たせる必要があって、ちょっと複雑になる。