Episode 14: GenServer

Summary

- Introduction to OTP
- Manually creating a `GenServer`
- `GenServer` API
- When to use `GenServer`

See the mix project in `code/e14-genserver` for some fun examples!

GenServer API

Starting the process

```
# Starts a non-linked process. Errors won't crash the current process.
{:ok, pid} = GenServer.start(CallbackModule, [arg1, arg2], opts)

# Starts a linked process. Errors will crash the current process.
{:ok, pid} = GenServer.start_link(CallbackModule, [arg1, arg2], opts)
```

Sending messages

- `GenServer.cast` - sends asynchronous message to a process without waiting for a response (non-blocking)
  - `handle_cast/2` will be called on the server to handle the request
  - A response is optional; can return `{:noreply, ...}` or `{:reply, ...}`
  - `:ok` is always returned, regardless of whether destination `server` received a response
- `GenServer.call/2` - sends synchronous message and waits for a response (blocking)
  - `handle_call/3` will be called on the server to handle the request

Reasons to Hide GenServer Callbacks

- A client API makes refactoring easier

- "Naming dynamic processes with atoms is a terrible idea! If we use atoms, we would need to convert the bucket name (often received from an external client) to atoms, and we should never convert user input to atoms. This is because atoms are not garbage collected. Once an atom is created, it is never reclaimed. Generating atoms from user input would mean the user can inject enough different names to exhaust our system memory!" - GenServer tutorial

When to Use GenServer

- To **distribute work across cores** for asynchronous tasks or for speed
  - *Also consider `Task` and `Agent`*
- To **synchronize** multiple users of a given piece of data
  - *Sequential reading can be a bottleneck with many messages*
- For **error recovery**
  - *Processes can be supervised and restarted*
- To **update** your data format without disconnecting users
- For a **client/server** system

If none of these cases apply, try using a struct or other data structure.

> A `GenServer`, or a process in general, must be used to model runtime characteristics of your system. A `GenServer` must never be used for code organization purposes.
>
> This is an anti-pattern not only because it convolutes the logic [and] puts [it] behind a single process that will potentially become a bottleneck in your system, especially as the number of calls grow.
>
> If you don't need a process, then you don't need a process. Use processes only to model runtime properties, such as mutable state, concurrency and failures, never for code organization.
>
> *source: When (not) to use a GenServer*