

Mini-talk Turbinado (Summary)

Bodo Naumann

Advanced Functional Programming

Turbinado claims to be an easy-to-use, fricking fast web application framework for Haskell. And in deed compared to other web application frameworks, turbinado performs very well. This applies to static as to dynamic content. On the website turbinados performance has been compared to rails and apache. With dynamic content turbinado needs only one sixth of the time rails needs to serve dynamic content. With static content it can't outperform the primary rock apache, but it needs less then one third of the time a rails server would need. The good performance with dynamic content is based on the fact that turbinado precompiles all pages. We encounter the same behaviour as of we would run an asp.net or java web application. On the first request a page is compiled, this is why the first request can take a bit longer, but all following requests are served ultra fast, because the page has not to be compiled again. This is a major benefit to web programming languages like php that are basically interpreters that have to interpret everything on each request.

Turbinado uses the model view controller approach and adds additionally Components and Layouts. The controller receives requests and models then it runs computations on the data. To make data access easier turbinado provides an Object-Relational Mapper that creates data types and functions corresponding to the data model used in the database. We can simply create those functions and types by running a script that is provided with the web server.

The view component produces XML or HTML and it can provide other extensions as well. We can use formats if we want to use other extensions. Formats translate requests based on provided MIME file type extension.

Components are reusable groups of controllers and views that can be used by other controllers and views. In Components we can integrate related functionality for later reuse. A component has a single controller and a set of views. They can be called via a tag inside of a view.

The server uses on big environment to store all runtime information. It stores server, application and request specific data such as data base connections requests and responses. The server components are glued together by routes. It is possible to define dynamic or static routes, or to combine dynamic and static routes. Dynamic routes translate urls into function calls. Folder and subfolder structures are translated into function calls and arguments. With static calls we can predefine function calls.

The server allows the usage of Layouts that provide wrappers for views. They are views them selves. In layouts the common layout can be defined and on a predefined place we can add the content that is actually provided by a view via a tag. For some outputs we do not want to use layouts and we can disable them inside of the controller. If we want to use a specific layout, we can define this inside of the controller as well.

The server provides support for cookies and cookie sessions, this means that we can store information that is relevant for a user during his stay at the website during the time he browses the page. Without sessions we would only have information in the scope of each request and all information that we can use has to be gathered by the current request. With sessions we can use previously gathered information as well.