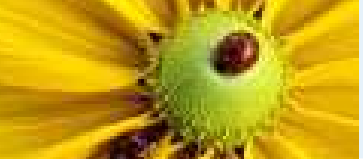
KKPortal Presentation

January 6, 2011

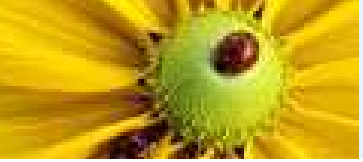


Simple Module Code

We create the most simple module by extending from AbstractModule, and implementing one function.

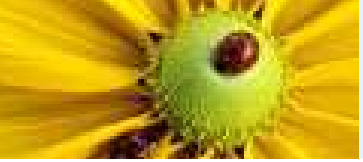
```
public class MotD extends AbstractModule {
  HTML widget = new HTML("Fetching␣MotD");

  @Override
  public Widget asWidget() {
    return widget;
  }
}
```



Simple Module Result

The module is available after running devmode and going to http://localhost:8888/KKPortal.html
<insert image of portal with MotD module>

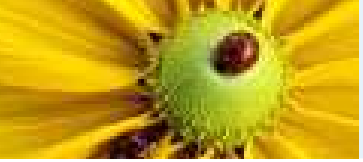


Backend Interface

Now we want to contact the backend

```
public interface MotDService extends RemoteService{
  public void getMessageOfTheDay(AsyncCallback<String>
      callback);
}
```

Similar to the way GWT does it.
The return value is defined based on the callback and enables asynchronous communication.



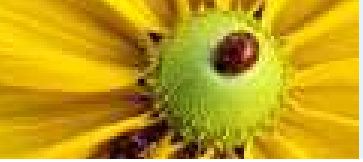
Changes to the client code

To use the interface we add two new functions to the MotD class

```
public class MotD extends AbstractModule {
  /* appended to the class: */

  @Inject
  public MotD(MotDService service){
    service.getMessageOfTheDay(this);
  }
  public void onSuccess(String result) {
    widget.setText(result);
  }
}
```

You do not have to worry about how the MotDService is implemented, but you can make easy mock implementations, that facilitates testing.



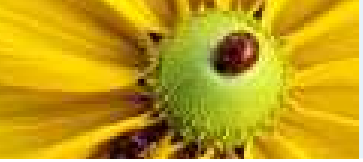
Backend implementation

We create a python file called MotDService.py and place it in server/w2b/rpc/

```
import w2b.settings

def getMessageOfTheDay(context):
    motd = w2b.settings.getGlobalSetting('LoginMessage
        ');
    return motd or 'No message'
```

Now the framework automatically connects the interface to backend implementation.

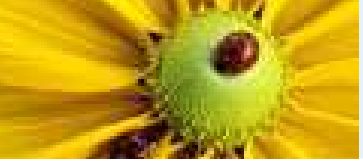


Alternative backend

Now suppose we annotate MotDService interface with @Dispatch(PHPDispatcher.class) and and place a file like this on the server:

```
<?php
class MotDService{
  function getMessageOfTheDay(){
    return "This is the message of the day";
  }
}
?>
```

Note that since the default server we run doesn't have an php interprenter we proxy http://localhost/php/... to another server, but if all the serverinterface classes is implemented with this, you do *not* need to use the supplied server, and it's usuable even without the Portal.

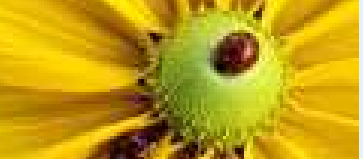


Dispatcher implementation

Excluding error handling and some details (see full code in repository)

```
public class PHPDispatcher implements Dispatcher {
  public <T> void execute() {
    String url = "php/dispatch.php?i="+serverinterface.getName()+"&m="
        +method;
    ...
    builder.sendRequest(encodedParameters,callback);
    });
  }
  public void onResponseReceived() {
    callback.onSuccess(decodedResult);
  }
}
```

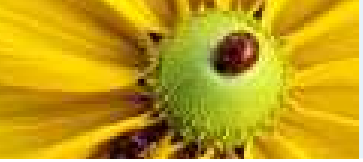
Note that since the default server we run doesn't have an php interprenter we proxy http://localhost/php/... to another server, but if all the serverinterface classes is implemented with this, you do *not* need to use the supplied server, and it's usuable even without the Portal.



The default dispatcher

The default dispatcher does this:

- Use bundling to get around the HTTP Connection limit.
- Receive bundled answers to requests on a different connection
- Use the receive connection to get server events
- Use json-rpc
- Uses python as backend



Summary

- Custom dispatchers for different calling conventions