Google App Engine
Plus
Spring

Ken Pritchard
16 May 2013

Overview

- Google App Engine Descrption
- Creating An App Engine Project
- Dependency Injection and Spring
- Spring Application Demo
- Port Spring Application to App Engine
- Google App Engine Demo
- Performance Considerations
- Dependency Injection Revisited

Google App Engine

- Highly elastic application servlet container hosting
- Additional APIs available for use, like authentication, persistence, data caching, etc
- Java, Python, and Go environments are available, PHP is on the way

Eclipse Projects

- Install the Google Plugin for Eclipse.
- Easiest way to create a demo app.
- Can create with sample code as stubs.
- The app has a unique directory structure that is required for deployment.

Maven Projects

- Google has provided a Maven plugin and several archetypes for creating and building Maven applications.
- Run the archetype:generate goal and search for:
  - `Group ID: com.google.appengine.archetypes`
  - `Artifact ID: skeleton-archetype or guestbook-archetype`
- Can also create a Maven project in Eclipse using the same information, although you may have to add the archetypes.

Dependency Injection

- Dependency Injection is essentially a design pattern that promotes loose coupling of large systems and runtime configuration choices.
- Also referred to as Inversion of Control because removes the burden of object creation from the objects themselves.

Spring

- Spring is a framework that promotes dependency injection in Java and some other languages.
- There are many enterprise applications in service with the Spring Framework.
- Because Spring applications are loosely coupled, it should be straightforward to move them to App Engine.

Spring Demo

- When a Spring application starts, the framework scans all the packages and configuration files to instantiate the required classes and put together the application context.
- For example, our servlet is injected with a data retrieval service specified by interface, with the concrete implementation injected at runtime.

Convert for App Engine

- Modify the POM to include the necessary artifacts and include application ID as a property.
- Add appengine-web.xml to WEB-INF directory to indicate application ID and logging configuration.
- Add logging.properties to WEB-INF directory for App Engine logging.

Run Application

- Can be run thru Maven in either Jetty or the App Engine Dev Server:

```
mvn jetty:run-war
mvn appengine:devserver
```

- Deploy to App Engine:

```
mvn appengine:update
```

- Test

  http://silverlakedemoapp.appspot.com/

Convert to Eclipse Project

- Now that you have converted a team project to use App Engine, you might want to be able to test and run it locally thru Eclipse
- There are two primary areas to configure for this:
  - The Eclipse Build Path
  - The Google App Engine Settings

Convert Eclipse Project - Build Path

- Right-click project, select Build Path --> Configure Build Path
- In Libraries tab, click Add Library... and select Google App Engine
- In the same window, select the Order and Export tab and move the App Engine SDK above the Maven Dependencies

Convert Eclipse Project - App Engine

- Right-click project, select Google --> App Engine Settings... and check the Use Google App Engine box.
- In the same window, select Web Application just below and check This project has a WAR directory, then browse to your WAR directory
- If you get an error about having a JRE instead of a JDK, check to make sure that JAVA_HOME is set to your JDK and it is in

Convert Eclipse Project - Run App

- You should now be able to right-click the project and run the dev server from Eclipse
- NOTE: You will need to run a Maven build first to refresh that WAR directory
- You should also be able to Deploy to App Engine from Eclipse

Performance Considerations

- Spring does a lot of work when the application starts to build the application context before the application starts.
- App Engine Frontend instances have a 60 second deadline to respond, so if the app takes too long to load it will generate an exception.
- Spring apps need to be optimized for App Engine.

App Engine Spring Best Practices

- See information at:
  - https://developers.google.com/appengine/articles/spring_optimization
- Reduce or Avoid Component Scanning
  - Use explicit XML configuration.
- Disable XML Validation in Production
- Use Lazy Initialization Where Possible
- Avoid Constructor Injection By Name
  - This is a bad practice in general and should be reserved for cases when working with a legacy app that cannot be modified.

Dependency Injection Revisited

- In our example app we are using non-persistent local storage, but in a real app we would be working with some kind of persistence storage.
- When we move an app to App Engine, that is an area that we will probably want to change.
- If we have been careful about setting up dependency injection, this will be easier to do.

Datastore Persistence

- Let's assume we want to take advantage of the efficiency and redundancy of BigTable and get a more object oriented view of our data.
- We create a new implementation of the EventService interface that uses Datastore.
- Swap the existing local service for the new Datastore service in our Spring injection.
- Yes, it's just that easy when dependencies are injected.

Links

- Google App Engine
  - https://developers.google.com/appengine/docs/java/gettingstarted/
- App Engine Maven Plugin
  - https://developers.google.com/appengine/docs/java/tools/maven
- Dependency Injection
  - http://www.martinfowler.com/articles/injection.html
- Spring
  - http://www.springsource.org/
- Spring Optimization for App Engine
  - https://developers.google.com/appengine/articles/spring_optimization
- Slides and Demo Apps
  - http://www.kenpritchard.com/code.html

Contact Information

ken@kenpritchard.com

http://www.kenpritchard.com

http://www.linkedin.com/in/kenpritchard

Questions?