springone 2GX

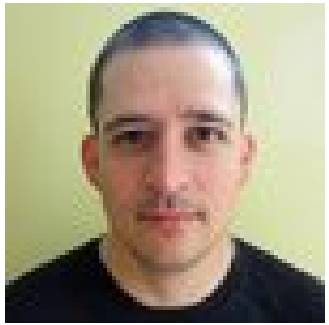
DALLAS

2014

Effective Application Development with GemFire using Spring Data GemFire

John Blum – Luke Shannon

Who Are We?



John Blum

Spring Data GemFire Lead
GemFire Engineer/Tech Lead
Pivotal Software, Inc.
Twitter: @john_blum

Luke Shannon

Community Engineer
Pivotal Software, Inc.

Agenda

- GemFire Overview
- Spring Data GemFire
  - Overview
  - Feature Enhancements
  - Productivity Features
  - New in 1.5.0.RELEASE
  - EA Concerns
  - Customer Case Studies
  - Looking Ahead
- Resources
- QA

GemFire Overview

What is GemFire?

GemFire is an *in-memory data grid* (IMDG)

GemFire can function as a OLTP/OLAP data management solution offering *high-throughput*, *low-latency* data access with *horizontal scale-out*

In the simplest case though, GemFire is just a *cache*...

– Key/Value Store (Objects/JSON)

In-Memory

GemFire stores application data (state) in the **JVM Heap**

GemFire and application **performance** is impacted by GC...

– Object graphs can be stored in serialized form to reduce strain on the *Garbage Collector*.

No **off-heap** storage...

– GemFireXD, Pivotal's in-memory SQL-based store, supports off-heap

In-Memory Continued...

Support for managing resources used by the local *cache* is handled by GemFire's *`ResourceManager`*

Important **configuration settings**:

– `critical-heap-percentage` - % of heap at which the *cache* might become inoperable due to GC pauses or `OutOfMemoryErrors`

– `eviction-heap-percentage` - % of heap when eviction begins on a Region configured with Heap LRU eviction

Data Grid

GemFire pools system resources across multiple nodes joined in a cluster to manage the entire application's state and behavior...

– CPU, memory, network and (optionally) local disks

Members join the cluster as peers using either *UDP multicasting* or GemFire's *Locator* process...

Shared-Nothing Architecture

Each member of the cluster is independent and self-sufficient...

– Members do not share memory or disk.

GemFire's "*Shared-Nothing Disk Persistence Architecture"* manages data in local disk files on each member independently from other members that host the same *cache (Region)*...

High Scalability

Additional GemFire Server data nodes (peer members) can be added to the cluster, and data **rebalanced**, to uniformly distribute the load

– GemFire is "*elastic*"; easily scales up or down

GemFire really scales when data is **partitioned** across many members of the cluster uniformly...

– Application behavior can be routed, distributed and run in parallel with close proximity to the data required by application logic to improve **throughput**

High-Throughput / Low-Latency

GemFire uses concurrent, in-memory data structures and a highly optimized distribution infrastructure to minimize context switching and contention

– **high read throughput** is achieved through *synchronous* or *asynchronous replication* of data, and...

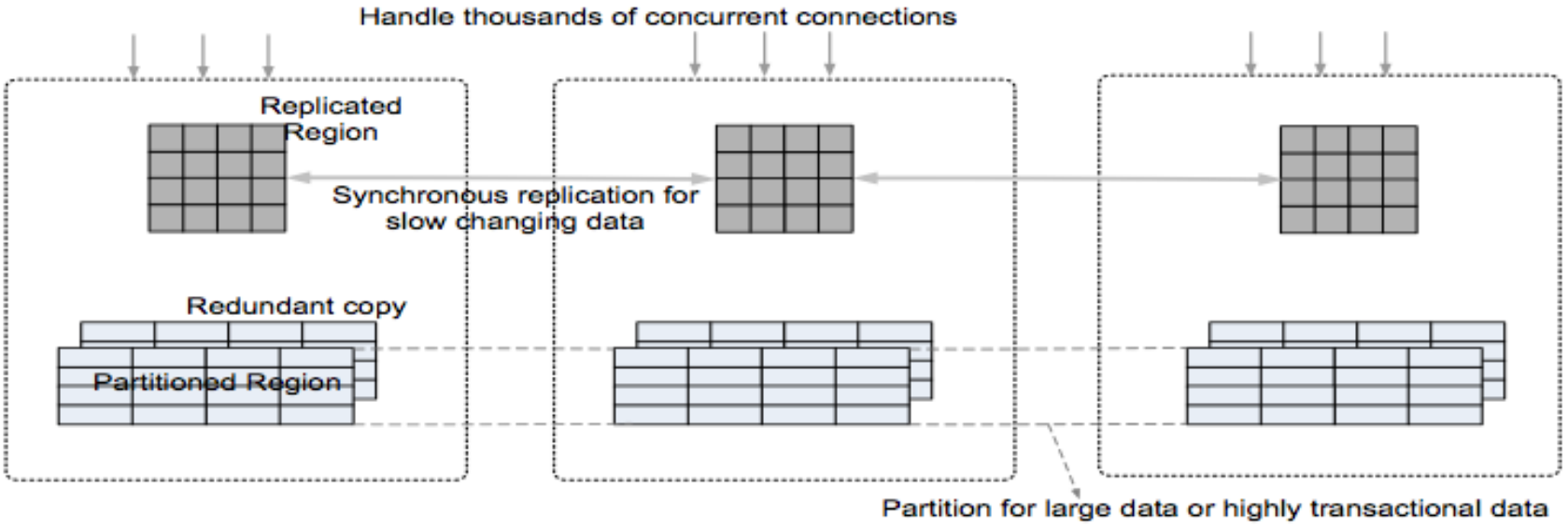
– **high read** and **write throughput** is achieved when data is *partitioned* across many nodes in the cluster

"*Linear increase in throughput is limited only by the backbone network capacity.*"

GemFire Data Storage Models

REPLICATE vs. PARTITION

In GemFire, a `Region` is the *cache...*



Region Configuration

Cache-like configuration settings...

- **Initial Capacity** and **Load Factor**
- **Key/Value-Constraints** (Class type)
- *Entry* **Eviction**
- *Region* and *Entry* **Expiration**

Additional Region Configuration

Type – *DataPolicy*, *Shortcuts*

Persistence – *DataPolicy*

Distribution – *DataPolicy*, *Scope*, *Subscription*

Disk Store – *Overflow* & *Persistence*

Statistics (used for *Expiration*)

Async and Subscription Message Conflation

Async Event Queue (e.g. "*Write-Behind*")

Gateway Sender (Site-to-Site WAN Replication)

Partition Region Specific Settings

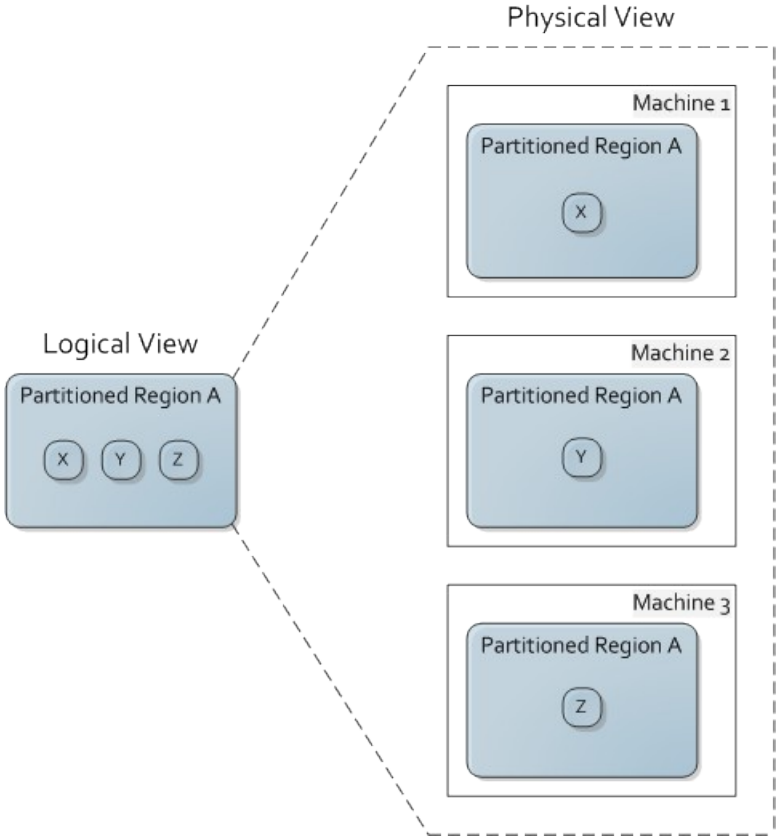
Local/Total Max Memory Size (MB)

Total Number of Buckets

Partition Resolver

Redundancy (# of Data Copies)

Collocation



Other Region Types

Distributed, Non-Replicated Regions

– Data is distributed; Each member only holds the data it has expressed "interests" in using *Subscription* or by defining cache entries (keys)

Local Region

– Data is not distributed; local-only (`Scope.LOCAL`) only to the member defining the *Region*

Empty Region

– Data accessor *Region* (PROXY); stores no data; receives events

Client Region

– `PROXY` or `CACHING_PROXY`

Consistency

Partition Regions

– *consistency* is maintained by routing all updates on a given key to the GemFire member holding the **primary** copy

– holds **lock** on key while distributing updates to other members having key

Replicated Regions

– concurrent updates → possibility of **conflicts** and **out-of-order updates**

– *conflict checking* ensures all members eventually apply the same value (*eventual consistency*)

– *conflict checking* ensures out-of-order updates are discarded.

Conflict checking is enabled with... `--concurrency-checks-enabled`

Cache Region Callbacks

Cache Listener

- `afterCreate`, `afterUpdate`, `afterDestroy`, `afterInvalidate`, ...

CacheLoader

- Synchronous “Read-Through” on *cache* misses from external data source

CacheWriter

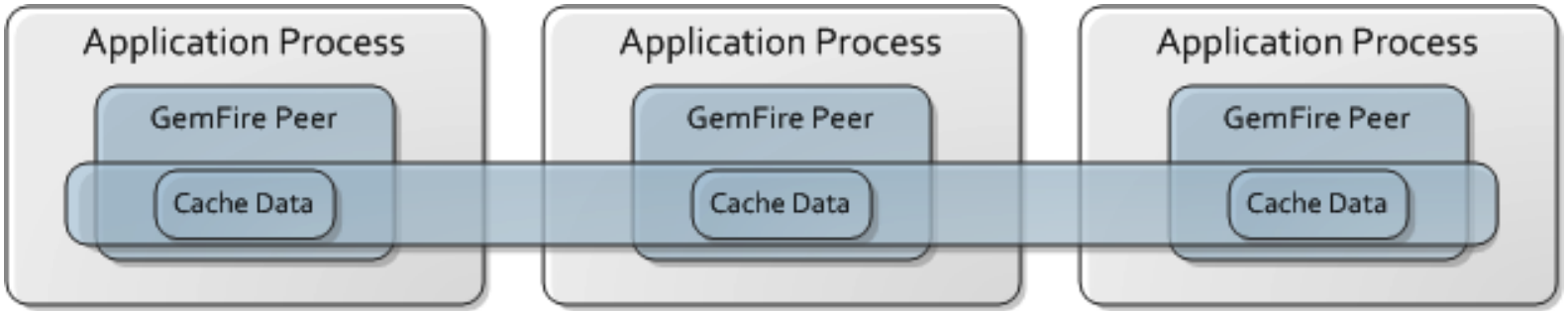
- Synchronous “Write-Through” to external data store

GemFire Topologies

Network Topologies
Peer-to-Peer (P2P)

Cache configuration designed for applications embedding the *cache* within the same process space and participate in the cluster...

– close proximity of application logic and data

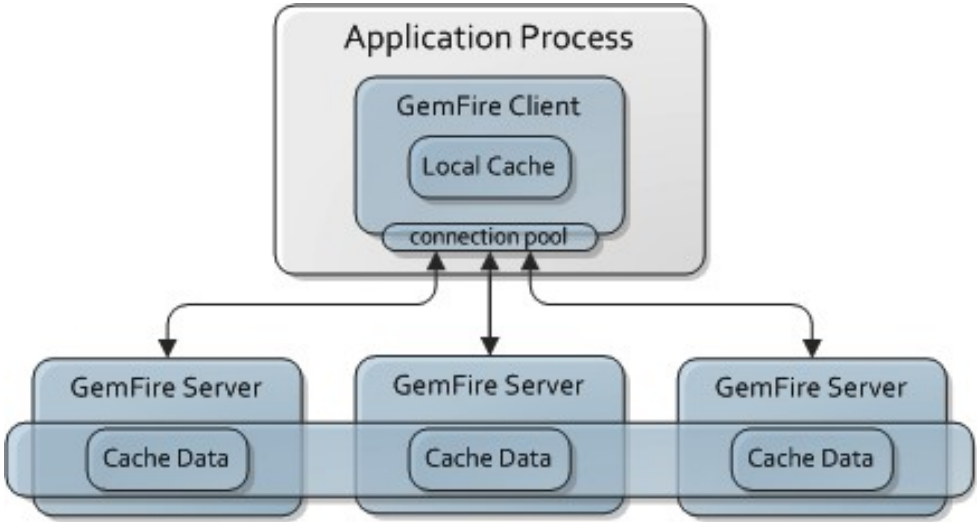


Network Topologies
Client / Server

Clients use single-hop data access...

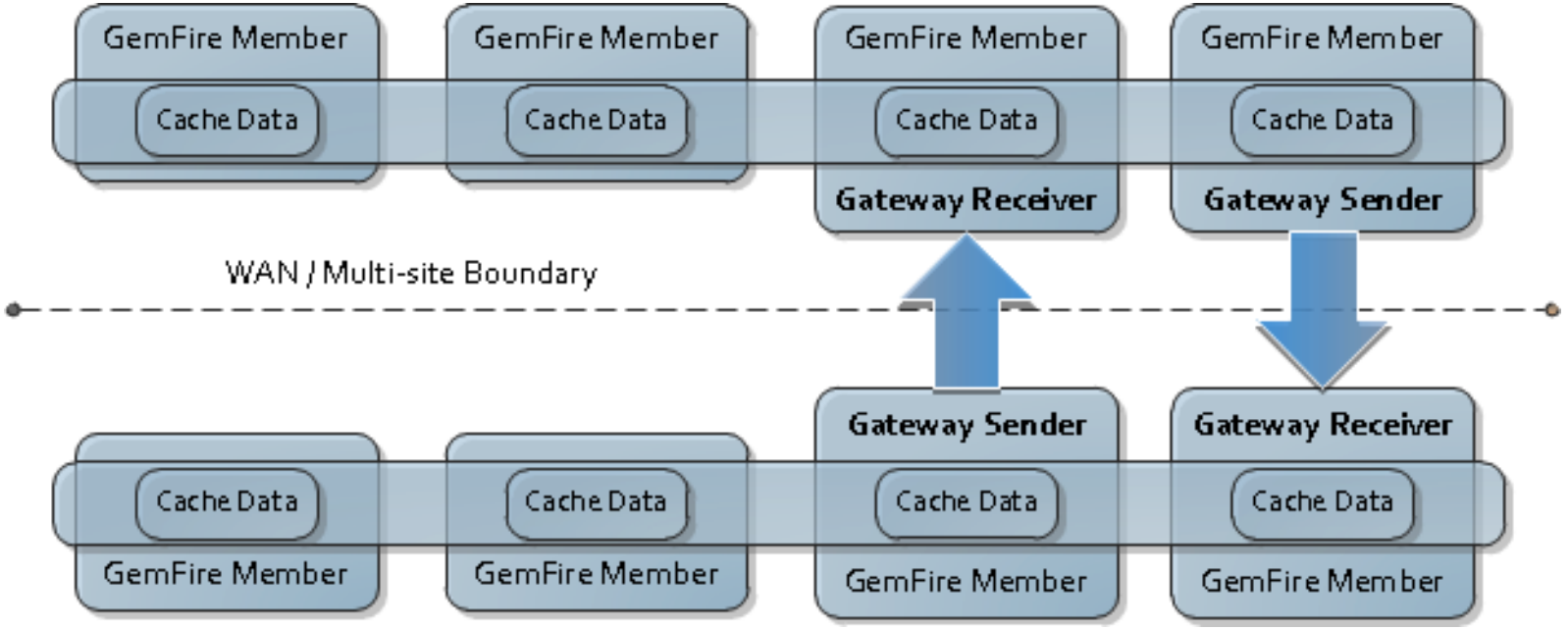
Continuously load balanced between servers based on load to get more predictable response-times...

Automatic fail-over when Servers fail or become unresponsive



Network Topologies
WAN

Site-to-Site Replication for DR



GemFire Features++

Still just scratching the surface...

- Cache and JTA-compliant Trans``action Management
- Functions
- OQL
- Client Register Interests or CQ
- Serialization (PDX, Java, DataSerialization)
- Security
- Native Support for Java and C++/C# Clients
- Memcache Integration
- REST Interface + JSON document storage
- Delta GII – when member **re-joins** the cluster
- Delta Propagation – distribute object changes only
- Rolling Upgrade - update GemFire version without having to bring the member/cluster down
- Split Brain Detection

GemFire Tips

1. Do not directly modify cached values...

   – Bypasses GemFire distribution as well as cache listeners/writers, expiration activities, transaction management leading to undesired results

   -- Instead, make a copy of the data by setting *–copy-on-read,* or use *CopyHelper*

2. *Key* class types must implement Object *equals(:Object)* and *hashCode()*;

3. GemFire serializes data entry keys and values during distribution, between client and server, and for persistence/overflow to Disk Stores, therefore keys/values must be serializable.

4. Application classes must be added to each peer member's CLASSPATH using the data in non-serialized form

5. Close the Region to release the resources held

6. Distributed System and Region configuration settings must match to be compatible systems

GemFire Cluster Demo

Spring Data GemFire

Agenda

- GemFire Overview
- **Spring Data GemFire**
  - Overview
  - Feature Enhancements
  - Productivity Features
  - New in 1.5.0.RELEASE
  - EA Concerns
  - Customer Case Studies
  - Looking Ahead
- Resources
- QA

Advantages of Using *Spring Data GemFire*

Uses Spring's powerful, non-invasive *programming model* and concepts with Pivotal GemFire to simplify configuration and development

Integration with other powerful *Spring IO Platform* Technologies...

 – Spring's *Caching Abstraction*

 – Spring's *Transaction Management*

 – *Spring Data Commons/REST*

 – *Spring Integration* (Inbound/Outbound Channel Adapters)

 – *Spring XD* (GemFire Input Source and Output Sink)

And...

Pivotal GemFire combined with **Spring Data GemFire** can already be used as a **JSR-107** *caching provider* when using **Spring Framework 4.1**

Dependencies Baseline

Spring Data GemFire (SDG) was sync'd with the **Spring Data** release train as of *Dijkstra (~March 2014, v1.4)*; just released with *Evans (v1.5)*

- *Spring Framework* 4.0.7
- *Spring Data Commons* 1.9
- GemFire 7.0.2
- Built/ran with JDK 7 & JKD 8

Feature Enhancements

Region bean definitions default to create (not lookup)...

```
<gfe:partitioned-region id="Example" .. ignore-if-exists="[true|false]">
    ...
</gfe:partitioned-region>
```

'Manual Start' for *GatewayReceivers*...

```
<gfe:gateway-receiver id="Example" .. manual-start="[true|false]">
    ...
</gfe:gateway-receiver>
```

Ability to execute *Spring Data GemFire* defined GemFire Functions using annotations in Gfsh

Productivity Features

Spring Boot Starter Data GemFire (**new**)...

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-gemfire</artifactId>
</dependency>
```

```
@Configuration
@ImportResource("/spring-data-gemfire-cache.xml")
@EnableAutoConfiguration
@EnableGemfireFunctions
@EnableGemfireRepositories
@EnableTransactionManagement
public class SampleDataGemFireApplication {

  public static void main(String[] args) {
    SpringApplication.run(SampleDataGemFireApplication.class, args);
  }
}
```

Productivity Features

GemfireTemplate

- simplified GemFire data access...
  - `contains[Key|Value], get, getAll, find, query,`
  - `put, putAll, putIfAbsent, replace`

Exception Translation

- maps GemFire Exceptions to `org.springframework.dao` Exceptions
- e.g. `EntryExistsException` → `DuplicateKeyException`

Productivity Features

Automatic creation of *DiskStore* “Disk Directory Location“ (**new**)...

```
<gfe:disk-store id=”Example” auto-compact=”true” ...>
    <gfe:disk-dir location=”/path/to/disk/files” max-size=”1024”/>
</gfe:disk-store>
```

Region Lookup 101

Spring Data GemFire supports GemFire `cache.xml`...

```
<gfe:cache cache-xml-location="/path/to/cache.xml" ...>
  ...
</gfe:cache>
```

GemFire 8 Cluster-based Configuration Management...

```
<util:map id="gemfireProperties">
  <prop key="name">SpringGemFireServer</prop>
  <prop key="use-cluster-configuration">[true|false]</prop>
</util:map>

<gfe:cache properties-ref="gemfireProperties" ...>
  ...
</gfe:cache>
```

Region Lookup

Given GemFire `cache.xml`...

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC  "-//GemStone Systems, Inc.
        //GemFire Declarative Caching 7.0//EN"
  "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>

  <region name="Parent" refid="REPLICATE">
    <region name="Child" refid="REPLICATE">
      <region name="Grandchild" refid="REPLICATE"/>
    </region>
  </region>

  <region name="Example" refid="PARTITION"/>

</cache>
```

Spring `context.xml` Was...

```
<gfe:cache cach-xml-location=”/path/to/cache.xml”/>

<gfe:lookup-region id="Parent">
  <gfe:lookup-region name="Child">
    <gfe:lookup-region name="Grandchild"/>
  </gfe:lookup-region>
</gfe:lookup-region>

<!-- or... -->
<gfe:lookup-region id="/Parent/Child"/>

<gfe:lookup-region id=”Example”/>
```

Region Lookup

```
@Repository("exampleRepo")
public class RegionBasedRepository extends DaoSupport {

  @Resource(name = "Parent")
  private Region<?, ?> parent;

  @Resource(name = "/Parent/Child")
  private Region<?, ?> child;

  @Resource(name = "/Parent/Child/Grandchild")
  private Region<?, ?> grandchild;

  @Resource(name = "/Example")
  private Region<?, ?> example;

  ...
}
```

Auto Region Lookup

Given same GemFire `cache.xml`...

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC  "-//GemStone Systems, Inc.
        //GemFire Declarative Caching 7.0//EN"
  "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>

  <region name="Parent" refid="REPLICATE">
    <region name="Child" refid="REPLICATE">
      <region name="Grandchild" refid="REPLICATE"/>
    </region>
  </region>

  <region name="Example" refid="PARTITION"/>

</cache>
```

Spring `context.xml` Now...

```
<gfe:cache cach-xml-location="/path/to/cache.xml"/>

<!--
<gfe:lookup-region id="Parent">
  <gfe:lookup-region name="Child">
    <gfe:lookup-region name="Grandchild"/>
  </gfe:lookup-region>
</gfe:lookup-region>
-->

<gfe:auto-region-lookup/>

<!-- optional -->
<gfe:lookup-region id="Example"/>
```

Auto Region Lookup

```
@DependsOn("gemfireCache")
@Repository("exampleRepo")
public class RegionBasedRepository extends DaoSupport {

  @Resource(name = "Parent")
  private Region<?, ?> parent;

  @Resource(name = "/Parent/Child")
  private Region<?, ?> child;

  @Resource(name = "/Parent/Child/Grandchild")
  private Region<?, ?> grandchild;

  @Resource(name = "/Example")
  private Region<?, ?> example;

  …
}
```

Region Templates

Region Templates are a way to share common configuration across multiple Region bean definitions in the Spring context...

```
<gfe:region-template id="base"/>

<gfe:replicate-region-template id="replica-tmpl" template="base"/>

<gfe:partition-region-template id="partition-tmpl" template="base"/>

<gfe:local-region-template id="local-tmpl" template="base"/>

<gfe:client-region-template id="client-tmpl" template="base"/>
```

Example...

```
…
<gfe:region-template id="Base" concurrency-checks-enabled="true"
    initial-capacity="51" load-factor="0.85" persistent="false" statistics="true">
  <gfe:cache-listener>
    <bean class="exmple.CacheListener" p:name="X"/>
  </gfe:cache-listener>
  <gfe:entry-tti timeout="600" action="DESTROY"/>
</gfe:region-template>

<gfe:region-template id="Ext" persistent=”true” template="Base">
  <gfe:gateway-sender-ref bean="ExampleGatewaySender"/>
</gfe:region-template>

<gfe:partitioned-region-template id="PartTmpl" copies="2" local-max-memory="8192"
    template="Ext">
  <gfe:cache-loader>
    <bean class="example.CacheLoader"/>
  </gfe:cache-loader>
  <gfe:subscription type="ALL"/>
</gfe:partitioned-region>

<gfe:partitioned-region id=”Example” colocated-with=”Neighbor” copies=”1”
     template=”PartTmpl”/>

<gfe:partitioned-region id=”Neighbor” persistent=”false” template=”Ext”/>
…
```

Region Template Oops

Cannot mix *Region* types; must be compatible...

```
<gfe:partition-region-template id="partition-tmpl" template="base"/>

<gfe:replicate-region id="Example" template="partition-tmpl"/>
```

Spring Data *Repositories* for GemFire

Spring Data GemFire's *Repository* is a provider implementation building on Spring Data Common's *Repository Abstraction*...

What is the *Repository Abstraction*...

- CRUD
- Querying
- ~~Pagination/Slices~~ & Sorting

Persisting Multiple Regions

Using Spring Data GemFire *Repositories*

Domain Class...

```
@Region("Gemstones")
public class Gemstone {

  @Id
  private Long id

  private String name;

  ...
}
```

Repository...

```
public interface GemstoneRepository extends GemfireRepository<Gemstone, Long> {
  ...
}
```

Repository Queries

```
public interface GemstoneRepository extends GemfireRepository<Gemstone, Long> {

     Gemstone findByName(String name);

     // @Query("FROM /Gemstones g WHERE g.price > $1")
     Gemstone findByPriceGreaterThan(BigDecimal price)

     List<Gemstone> findByNameOrderByNameDesc(Sort order);
}
```

Using *@Query* is useful for *joins* or *object graph traversal*...

```
public interface CustomerRepository extends GemfireRepository<Customer, Long> {

  @Query("<trace> SELECT DISTINCT c FROM /Customers c, c.addresses a
          WHERE a.city = $1")
  List<Customers> findCustomersInCity(String city)
}
```

@Region Repository

```
@Region("Gemstones")
public interface GemstoneRepository extends GemfireRepository<Gemstone, Long> {
}
```

```
@Region("ValuableRocks")
public interface GemstoneRepository extends GemfireRepository<Gemstone, Long> {
}
```

Transaction Management

Spring Data GemFire integrates nicely with the *Spring Framework's* **Transaction Management** infrastructure...

– supports both ***Cache*** and ***Global*** Transactions

For ***Cache*** Transactions...

```
<gfe:cache id="cache" …/>

...

<gfe:transaction-manager id="txManager" cache-ref="cache"/>
```

Transaction Management

@Service component...

```
@Service("gemstoneService")
public class GemsService {

protected static final List<String> APPROVED_GEMS = new ArrayList<String>(
    Arrays.asList( "ALEXANDRITE", "AQUAMARINE", "DIAMOND", "OPAL",
      "PEARL", "RUBY", "SAPPHIRE", "SPINEL", "TOPAZ"));

@Transactional(readOnly = false)
public Gemstone save(Gemstone gemstone) {
  gemstone = getGemFireGemsDao().save(getDatabaseGemsDao().save(gemstone));

  // NOTE deliberate, but stupid and naive business validation
  // after the mutating data access!
  if (!APPROVED_GEMS.contains(gemstone.getName().toUpperCase())) {
    // NOTE if the gemstone is not valid, blow chunks
    // (should cause transaction to rollback for GemFire and Database)!
    throw new IllegalGemstoneException(
      String.format("'%1$s' is not a valid gemstone!",
        gemstone.getName()));
  }

  return gemstone;
}
```

GemFire Function Support

Spring Data GemFire provides POJO-based, Function annotation support for creating, registering and executing GemFire Functions

- *@GemfireFunction*
- *@FunctionExecution*

Creating Function(s)

```
@Component
public class RegionFunctions {

  @GemfireFunction
  public String echo(String message) {
    return String.format("You said '%1$s'!", message);
  }

  @GemfireFunction
  public Integer regionSize(FunctionContext context, String regionNamePath) {
    Region region = getRegion(context, regionNamePath);

    if (region != null) {
      return region.size();
    }

    throw new RegionNotFoundException("The Region on which the size will be determined”
          was not found!");
  }

  protected Region getRegion(FunctionContext context, String regionNamePath) {
    …
  }
}
```

Registering Function(s) with GemFire

XML...

```
<gfe:annotation-driven/>

<bean class="example.RegionFunctions"/>
```

Component-scanning...

```
<gfe:annotation-driven/>

<context:component-scan base-package="example">
  ...
</context:component-scan>
```

Java Config...

```
@Configuration
@ImportResource("/spring-data-gemfire-context.xml")
@EnableGemfireFunctions
public class SpringGemFireConfiguration { … }
```

Function Execution

Peer Cache...

```
@OnMember(groups = "exampleGroup")
public interface OnMemberEchoFunctionExecution {

  String echo(String message);
}
```

Client Cache...

```
@OnServer(cache = "gemfireCache")
public interface OnServerEchoFunctionExecution {

  String echo(String message);
}
```

Function Execution

Peer Cache and Client Cache...

```
@OnRegion(region = "Example")
public interface OnRegionFunctionExecution {

  String echo(String message);

  Integer regionSize(String regionNamePath);
}
```

Function Execution Configuration

XML...

```
<gfe-data:function-executions base-package="example">
  <gfe-data:exclude-filter type="regex" expression=".*OnServer.*"/>
</gfe-data:function-executions>
```

Java Config...

```
@Configuration
@ImportResource("/spring-data-gemfire-context.xml")
@EnableGemfireFunctionExecutions
public class SpringGemFireConfiguration {
  ...
}
```

Function Execution Invocation

```
@Component
public class ApplicationComponent {

  @Autowired
  private OnRegionFunctionExecution functions;

  public ... someMethod(String regionNamePath) {
    Integer size = functions.regionSize(regionNamePath);
    …
  }

  …
}
```

Order Application Demo

Spring Data GemFire Case Studies

Case Study: GemFire Configuration with SDG

Set the stage... **3** Primary Use Cases of *Spring Data GemFire*...

1. **GemFire Configuration**
   - replacement for GemFire `cache.xml`
2. *Peer Cache* Application

```
<gfe:cache properties-ref=".." cache-xml-location=".." .../>
```

3. *Client Cache* Application

```
<gfe:client-cache properties-ref=".." cache-xml-location=".." .../>
```

But, how do you “launch” the GemFire Sever?

Java Main

```
public class SpringGemFireLauncher {

  // e.g. args[0] = "classpath:spring-gemfire-context.xml
  private static void validate(String[] args) {
    ...
  }

  private static void main(String[] args) {
    validate(args);
    new ClassPathXmlApplicationContext(args);
    ...
  }
}
```

1.4 Initializer

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC  "-//GemStone Systems, Inc.
     //GemFire Declarative Caching 7.0//EN"
     "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>
  <initializer>
    <class-name>
org.springframework.data.gemfire.support.SpringContextBootstrappingInitializer
    </class-name>
    <parameter name="contextConfigLocations">
      <string>
        classpath:com.example.package,org.example.another.package...
      </string>
    </parameter>
  </initializer>
</cache>
```

1.4 Initializer

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC  "-//GemStone Systems, Inc.
     //GemFire Declarative Caching 7.0//EN"
     "http://www.gemstone.com/dtd/cache7_0.dtd">
<cache>
  <region name="Example" refid="REPLICATE">
    <region-attributes initial-capacity="51" load-factor="0.85">
      <cache-loader>
        <class-name>example.CacheLoader</class-name>
      </cache-loader>
    </region-attributes>
  </region>
  <initializer>
    <class-name>
org.springframework.data.gemfire.support.SpringContextBootstrappingInitializer
    </class-name>
    <parameter name="contextConfigLocations">
      <string>
        classpath:path/to/spring-gemfire-context.xml,
        classpath:path/to/app-context.xml
      </string>
    </parameter>
  </initializer>
</cache>
```

1.4 LazyWiringDeclarableSupport

Used to be, use *Spring Data GemFire's* `WiringDeclarableSupport`

```
<!-- Spring context.xml -->
<gfe:cache cache-xml-location="/path/to/cache.xml" ... />

<!-- GemFire cache.xml →
<cache>
  <region name="Example" refid="REPLICATE">
    <region-attributes initial-capacity="51" load-factor="0.85">
      <cache-loader>
        <class-name>example.CacheLoader</class-name>
```

Now, must use...

```
package example;
public class CacheLoader
     extends LazyWiringDeclarableSupport
     implements CacheLoader<String, User> {

  @Autowired
  private DataSource dataSource;

  ...
}
```

GemFire 8 Gfsh Support


```
gfsh>start server -spring-xml-location="classpath:path/to/context.xml"
```

Resource Location Prefixes...

```
    -spring-xml-location="file:/path/to/context.xml"
```

DEMO

Case Study: Global Transaction Management

Using GemFire with an external data store (e.g. MySQL RDBMS) in a Global Transactional context...

– Use Spring's `JtaTransactionManager`; Not...

```
<!-- WRONG -->
<gfe:transaction-manager id="txManager" cache-ref="cache"/>
```

– Uses **GemFire's JTA Transaction Manager** provider implementation along with GemFire's **JNDI context** to *register DataSource* as well as *locate* GemFire's *TX Manager* and *JTA UserTransaction*

– Why not **Atomikos, Bitronix, or JOTM**?

Code...

DEMO

Case Study: Spring Caching Abstraction

Use Spring's *Caching Abstraction*...

Use GemFire as ***caching provider*** to replace ***Oracle Coherence***, but absolutely no presence of GemFire in the application...

Support for varying (some old, some new) application client ***versions***...

– New Domain Object class field “additions”

– Old Domain Objects should trigger a “cache miss”

– New Domain Object state must be preserved when used by an old application (PDX)

The **10% case** *Juergen* was talking about in *Caching and Messaging Improvements*

DEMO

Looking Ahead...

P0: Finish **GemFire 8** Support in **Spring Data GemFire 2.0**!

P0: Edit ***Spring Data GemFire Reference Guide***

P0: Rewrite ***Spring GemFire Examples***

P0: Provide *Spring GemFire Case Studies*

P0: Provide *Spring Data GemFire Reference Implementation*

P1: Integration Test Framework for Distributed Test Cases, WAN

P1: Integration with *Spring Security* as a facade over GemFire's Security Model

P2: Support for *Java-based Configuration* (Groovy)

P3: *Pagination* and *Slices*

Resources

GemFire Overview
http://gemfire.docs.pivotal.io/latest/userguide/index.html#getting_started/product_intro.html

GemFire User Guide
http://gemfire.docs.pivotal.io/latest/userguide/index.html

Spring Data GemFire Reference Guide
http://docs.spring.io/spring-data-gemfire/docs/current/reference/html/

Guide: https://spring.io/guides/gs/accessing-data-gemfire/
Guide: https://spring.io/guides/gs/caching-gemfire/

GitHub (source): https://github.com/spring-projects/spring-data-gemfire
GitHub (examples): https://github.com/spring-projects/spring-gemfire-examples

JIRA: https://jira.spring.io/browse/SGF

StackOverflow: https://stackoverflow.com/questions/tagged/spring-data-gemfire

Thank You!