Persistence

Storing and Retrieving Objects

Executive Summary

The ability to store and retrieve objects between subsequent or concurrent executions of a program is called "object persistence." It is a problem with many solutions, none of them perfect. This presentation serves as an overview of the concepts and an introduction to a few of the solutions.

Agenda

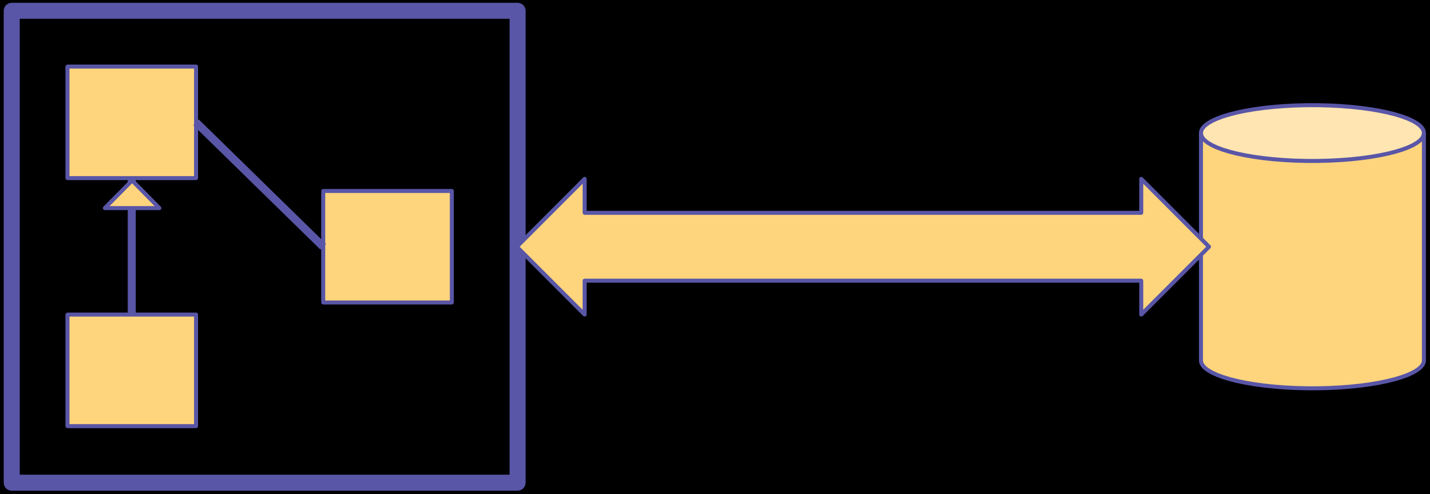
- Overview
- Concepts and Terminology
- Design Concerns
- Techniques
- Providers
- Summary

- Overview
- Concepts and Terminology
- Design Concerns
- Techniques
- Providers
- Summary

Overview

What is Persistence?

The storing and retrieving of information or state between subsequent executions of an application.



Overview

- Many developers use the word as verbal shorthand for "storing objects in a database."
- Persistence can be achieved through a variety of mechanisms, including:
    - Files stored on the hard disk
        - Flat files
        - Structured files
        - Binary files
    - Cloud storage
    - Database storage

Overview: Examples

- Web server
  - Content (web pages, web apps, etc.)
- Image viewer
  - Images and transforms thereof
- Banking
  - Account balances, transaction history, etc.
- Virtually every non-trivial application
  - User preferences and settings

Overview: Scope of Presentation

Given the thrust of this course, I'm going to focus on the aspects of persistence that pertain to object-oriented programming.

In particular, I'll be focusing on Java frameworks and persistence as it applies to storing objects in a database.

- Overview
- Concepts and Terminology
- Design Concerns
- Techniques
- Providers
- Summary

Concepts and Terminology

Persistence brings with it a whole set of jargon:

- Transparency
- CRUD
- POJO
- ACID

Concepts and Terminology: Transparency

Transparent or orthogonal persistence is a system in which state is preserved intrinsically, without any special actions from the application.

Concepts and Terminology: CRUD

The four basic operations involved in object persistence are <u>c</u>reate, <u>r</u>ead, <u>u</u>pdate, and <u>d</u>elete (CRUD).

CRUD	SQL
Create	INSERT
Read	SELECT
Update	UPDATE
Delete	DELETE

Concepts and Terminology: POJO

A plain old java object (POJO) is an object that need not extend a particular class or implement a specific interface or contain a prespecified annotation in order to work with a given framework or library.

The term is most commonly associated with JavaBeans and common persistence technologies.

Concepts and Terminology: POJO

Persisting a POJO is attractive because it prevents code from getting cluttered with implementation details related to the persistence framework. Unfortunately, it is rarely the case that an object can be stored and retrieved without some manner of alteration.

Concepts and Terminology: ACID

Atomicity, consistency, isolation, and durability (ACID) are four characteristics of a reliable database system.

Design Concerns

- Overview
- Scalability
- Flexibility
- Portability
- Maintainability
- Summary

Design Concerns: Overview

Although it may seem trite to say it, $g (Object/Relational Mapping) is the Vietnam of Computer Science. It represents a quagmire which starts well, gets more complicated as time passes, and before long entraps its users in a commitment that has no clear demarcation point, no clear win conditions, and no clear exit strategy.

- Ted Neward

http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx

Design Concerns: Overview

Persistence is at times a controversial topic in computer science. Some hate ORM, others insist it is the only viable option. By looking at design implications, we can decide when and where to employ it.

Design Concerns: Overview

Java Database Connectivity (JDBC), aka "the hard way," is the Java API for direct interaction with a database. This is the standard for comparison when we look at the different approaches to persistence.

Design Concerns: Scalability

As the footprint of the application grows (more users, more data), the persistence solution needs to be able to grow with it.

Design Concerns: Scalability

NoSQL is often a good solution when a large-scale or high-performance persistence framework is needed. We'll cover it in more detail in the Frameworks section.

Design Concerns: Flexibility

As the requirements of the system change, the persistence solution needs to remain able to support them.

Design Concerns: Flexibility

A good persistence solution offers a layer of abstraction that allows many different and diverse objects to be persisted without much special logic to massage them into the database.

Design Concerns: Portability

Moving from one architecture, database, operating system to another can be a Big Deal.

Design Concerns: Portability

An example from personal experience:

I implemented persistence on a project using OpenJPA with a Derby database. Our team lead felt that Derby wouldn't scale well so we switched to PostgreSQL. This required some configuration changes but no source code changes in our project. We later moved to Oracle because a new customer wouldn't let any other database system on their network. Again, none of our source code had to change -- thanks to the abstraction provided by OpenJPA.

Design Concerns: Maintainability

As the objects being persisted change, an inadequate persistence framework is going to need constant attention.

Design Concerns: Maintainability

Without ORM, one is forced to write SQL or JDBC queries by hand. This approach is quick and dirty, and often sufficient to get a small project off the ground, but it introduces a maintenance nightmare and is rarely feasible in the long run.

Design Concerns: Summary

If you need a persistence solution that is scalable, flexible, portable, and maintainable, you are most likely looking to ORM (or similar approaches) to meet your needs. Homegrown solutions rarely work past the initial stages of a project.

- Overview
- Concepts and Terminology
- Design Concerns
- Techniques
- Providers
- Summary

Techniques

- Object Relational Mapping (ORM)
  - Using a relational database to store and relate objects
- NoSQL
  - Using a database for nothing more than record storage
  - Highly optimized for insert and retrieve operations
- Object-oriented Database (OODB)
  - Using a database integrated to the point that objects are stored "natively"

Techniques: ORM

- Using a relational database in conjunction with frameworks is perhaps the most common approach to object persistence
- Popular frameworks include Hibernate, EclipseLink, and OpenJPA

Techniques: ORM

- Object-relational impedance mismatch
  - The principles of OOA/OOD/OOP aren't a good match for a relational database
  - Inheritance, Encapsulation, Polymorphism, and other concepts aren't supported by a relational database
    - For example, inheritance hierarchies won't map cleanly to tables
  - The winning approach will involve POJOs and treat use the database as an information repository and nothing more
  - Joins are particularly slow

Techniques: ORM - JPA

- JPA is an API, implemented by several popular frameworks.
- JPA is the "official" Java ORM.
- JPA is defined by JSR 220 (JPA 1.0) and JSR 317 (JPA 2.0)

Techniques: ORM - JPA

- The Java Persistence Query Language (JPQL), similar to SQL, is used to interact with Entities in a relational database.
- JPQL syntax is similar to that of SQL, but operates on Entities rather than database tables.

Techniques: ORM - JPA

- An entity is a lightweight persistence domain object. This term is specific to the Java Persistence API (JPA).
- Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table.
- The primary programming artifact of an entity is the entity class, although entities can use helper classes.

Concepts and Terminology: Entity

An entity class must follow these six requirements:

1. The class must be annotated with the `javax.persistence.Entity` annotation.
2. The class must have a public or protected, no-argument constructor. The class may have other constructors.
3. The class and its methods and persisted member variables cannot be declared final.

Concepts and Terminology: Entity

An entity class must follow these six requirements:

4. If an entity instance is passed by value as a detached object, such as through a session bean’s remote business interface, the class must implement the Serializable interface.
5. Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.

Concepts and Terminology: Entity

An entity class must follow these six requirements:

6. Persistent instance variables must be declared private, protected, or package-private, and can only be accessed directly by the entity class's methods. Clients must access the entity's state through accessor or business methods.

Techniques: NoSQL

NoSQL is a generic term for database systems that eschew the relational model for gains in performance or scalability.

NoSQL databases are not typically built around tables, and usually do not employ SQL for data manipulation.

Techniques: NoSQL

This approach is useful when dealing with an exorbitantly large amount of data, but won't work if that data requires a relational model.

Rather than relate data elements to each other, NoSQL provides fast storage/retrieval of *unrelated* records.

Techniques: OODB

An object-oriented database (OODB) is sometimes necessary when the complex relationships between objects can't be easily or efficiently mapped to the tables of a relational database.

In practice, this technique is much rarer than ORM.

- Overview
- Concepts and Terminology
- Design Concerns
- Techniques
- Providers
- Summary

Providers: Terminology

This is a topic that can easily be confusing, largely because people use language imprecisely and terms are often overloaded.

People may use the word "framework" to refer to libraries, applications, providers, databases, or APIs.

Providers: Terminology

Examples

- "Spring is a web *framework* that relies on Hibernate for persistence."
- "Hibernate is an ORM *framework* that supports JPA."
- "ORM is a *framework* for storing objects in a database."
- "JPA is a Java *framework* for persisting objects."

Providers: Terminology

- OpenJPA, Hibernate, etc., are persistence providers (implementations)
- JPA is an API or interface (not an implementation)
- Spring is an application framework
- ORM is a programming technique for persisting objects

Providers

- ORM
  - Java
    - Cayenne
    - EclipseLink (JPA, JAXB, JCA, SDO)
    - Hibernate (JPA)
    - OpenJPA (JPA)
    - ORMLite
  - C++
    - ODB
    - QxOrm

Providers

- NoSQL
  - Java
    - eXist (XML)
    - Jackrabbit (document)
    - OrientDB (document, graph)
  - C++
    - Clusterpoint (document)
    - mongoDB (document)

Providers

- OODB
  - Java
    - ObjectDB (JPA, JDO)
    - db4o
  - .NET
    - db4o

Providers: OpenJPA

- JPA implementation
  - Apache 2.0 license (F/OSS)
- Developed from BEA's Kodo

Providers: Hibernate

- JPA implementation
  - LGPL (F/OSS)
- High-level functions create buffer between database interaction and object manipulation
- Hibernate Query Language (SQL-like)

Providers: EclipseLink

- Bundles 3 persistence solutions
  - XML Binding: JAXB and SDO
  - ORM: JPA 2.0 reference implementation
  - Database Web Services: JAX-WS and JAX-RS
- Developed from Oracle's TopLink
- Maintained by the Eclipse Foundation

- Overview
- Concepts and Terminology
- Design Concerns
- Techniques
- Providers
- Summary

Summary

Persistence is one aspect of system design that requires special attention if the end product needs to be scalable, flexible, portable, or maintainable.

There are many options and techniques.

Persistence isn't one of the "solved problems" of computer science, but we have a number of viable options.

Additional Reading

- http://martinfowler.com/bliki/OrmHate.html
- http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx
- http://database-programmer.blogspot.com/2010/12/historical-perspective-of-orm-and.html
- "Persistence in the Enterprise" by Barcia, Hambrick, Brown, Peterson, and Bhogal

The End