Architecture of Enterprise Applications 13
Clustering and Internationalization

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents

- Clustering
  - Nginx
  - Reverse proxy
  - Load balancing
- Internationalization
  - Locale
  - Resource bundle

Clustering

- A large-scale system typically:
  - Has many user, potentially in many different places
  - Is long-running, that is, required to be “always up”
  - Processes large numbers of transactions per second
  - May see increases in both its user population and system load
  - Represents considerable business value
  - Is operated and managed by multiple persons

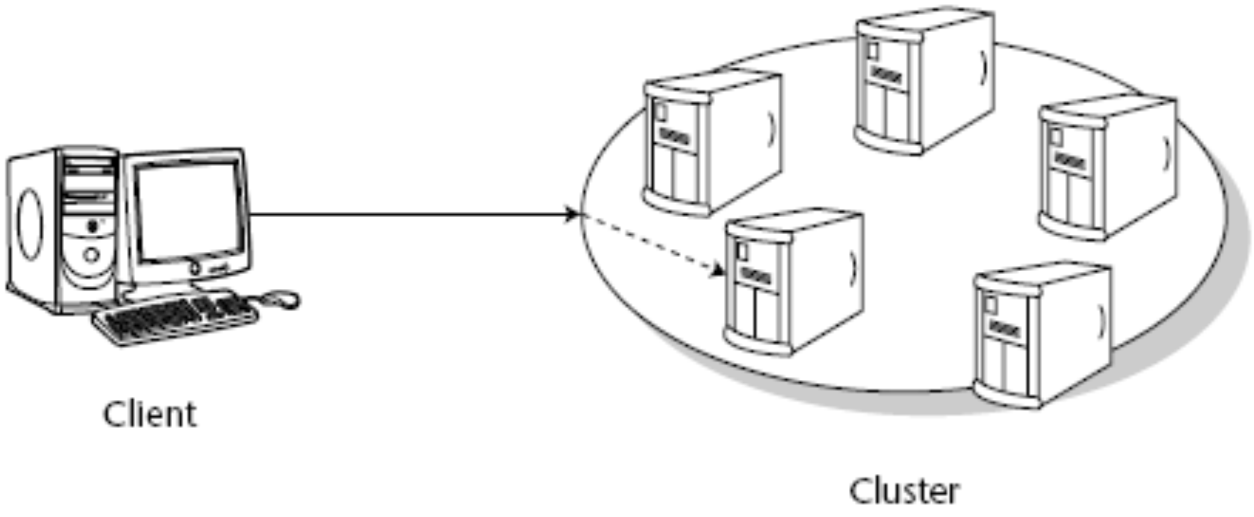
- Essential requirements on large-scale systems are often summarized by the following three properties(RAS):
  - Reliability
  - Availability
  - Serviceability
  - Scalability

Clustering

- Clustering addresses many of the issues faced by large-scale systems at the same time.
- A cluster is a loosely coupled group of servers that provide unified services to their clients.
- The client's view of the cluster is a single, simple system, not a group of collaborating servers. This is referred to as a single-system view or single-system image.
- Computers in a cluster are called nodes.

Clustering

- Clustering can be a very involved technology, potentially encompassing group communication and replication protocols, and network components such as load balancers and traffic redirectors at different layers in the protocol stack.



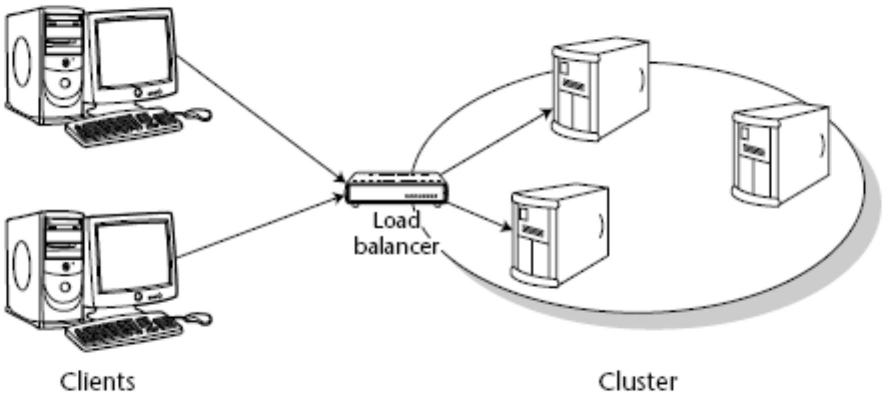
Clustering

- The main principle behind clustering is that of redundancy.
  - Reliability
    - Remove single points of failure
  - Availability
    - Overall availability is 1-(1-f%)^n
  - Serviceability
    - More complex than a single application server
    - But we could get ability for hot upgrade
  - Scalability
    - It is cheaper to build a cluster using standard hardware than to rely on multiprocessor machines.
    - Extending a cluster by adding extra servers can be done during operation and hence is less disruptive than plugging in another CPU board.

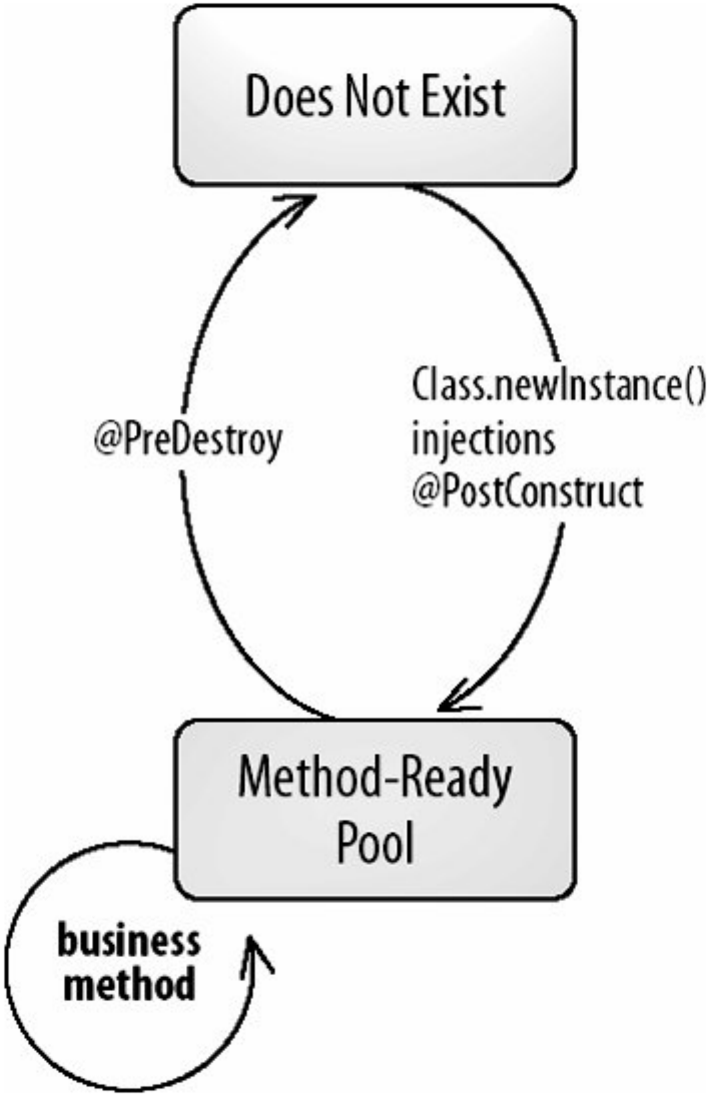
Load balancing and Failover

- Load balancing means distributing the requests among cluster nodes to optimize the performance of the whole system.
  - The algorithm that the load balancer uses to decide which target node to pick for a request can be systematic or random.
  - Alternatively, the load balancer could try to monitor the load on the different nodes in the cluster and pick node that appears less loaded than others.

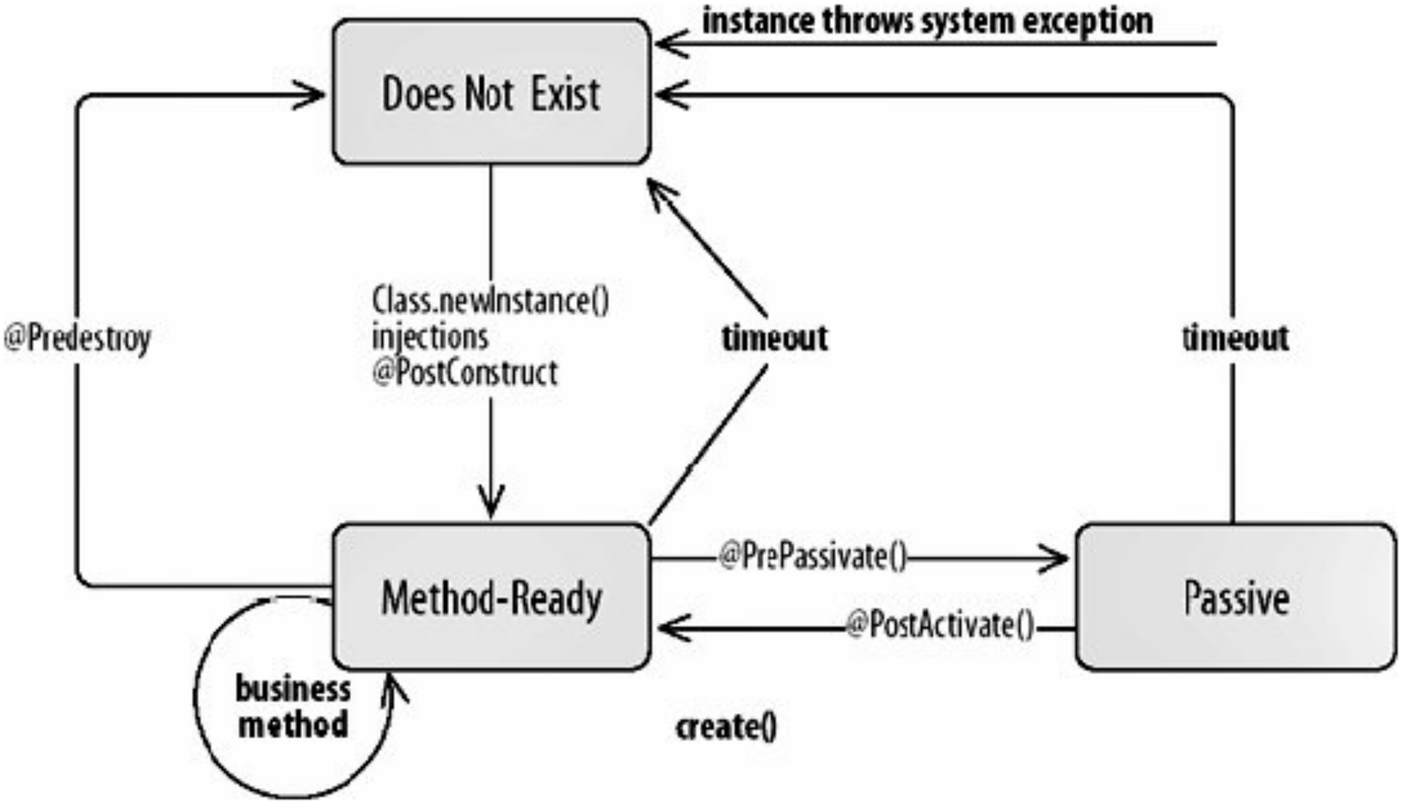
- An important feature for Web load balancers is session stickiness, which means that all requests in a client's session are directed to the same server.



Life Cycle of a Stateless Session Bean

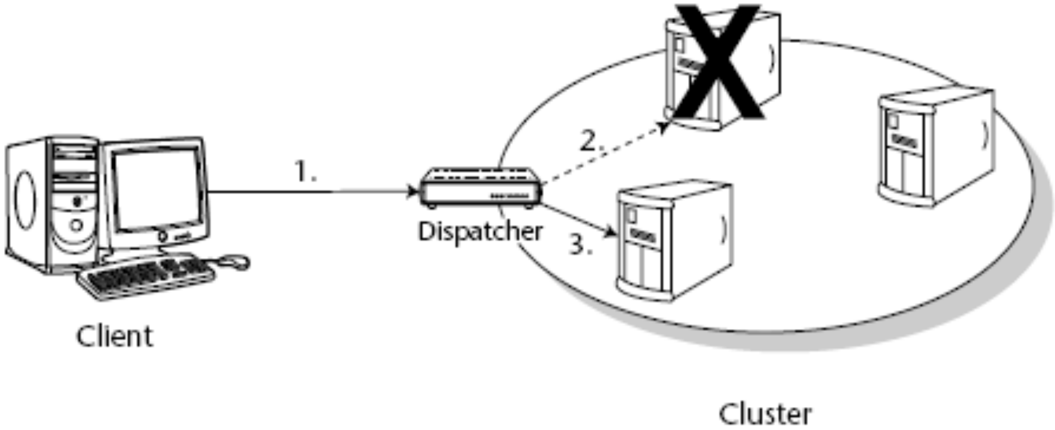


Life Cycle of a Stateful Session Bean



Load balancing and Failover

- For a cluster to provide higher availability to clients that a single server, the cluster must be able to failover from a primary server to another, secondary server when failures occur.
  - Request-level failover. It occurs when a request that is directed to one node for servicing cannot be serviced and is subsequently redirected to another node.
  - Session failover. If session state is shared between clients and servers, request-level failover may not be sufficient to continue operations. In this case, the session state must also be reconstructed at server node.



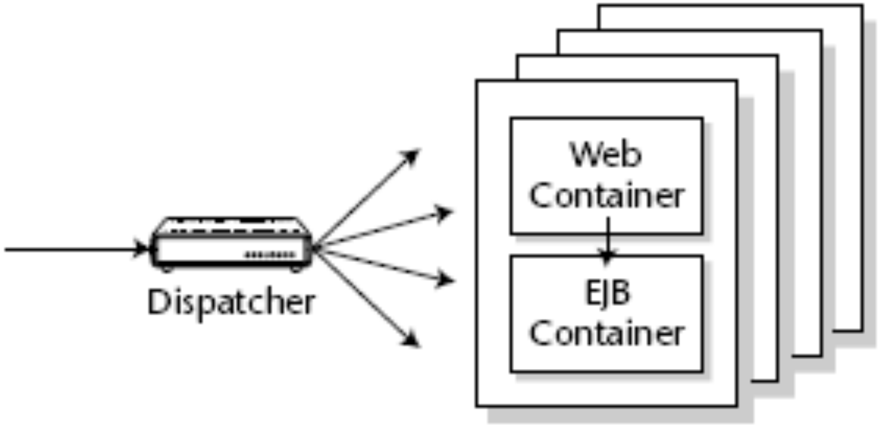
The concept of idempotence

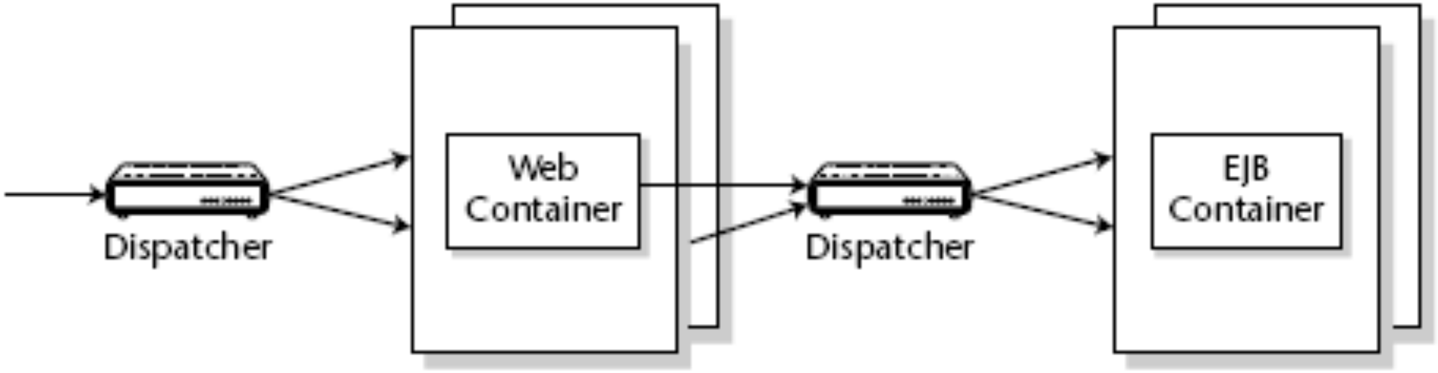
- An idempotent method is one that can be called repeatedly with the same arguments and achieves the same results each time.
  - HTTP GET
  - Generally, any methods that alter a persistent store based on its current state are not idempotent, since two invocations of the same method will alter the persistent store twice.

- A failed request could have occurred at one of three points:
  - After the request has been initiated but before method invocation on the server has begun to execute.
  - After the method invocation on the server has begun to execute, but before the method has completed.
  - After the method invocation on the server has completed but before the response has been successfully transmitted to the remote client.

Multi-tier applications

- In a Web-based system, the following configurations are possible:
  - Collocated architecture
  - Distributed architecture





Multi-tier applications

FEATURE	COLLOCATED	DISTRIBUTED	WINNER?
Reliability	High	Low	Collocated
Availability	High	Low	Collocated
Serviceability	High	Low	Collocated
Network efficiency	No sockets	More marshalling overhead	Collocated
Efficient use of hardware	High	Low	Collocated
Security	No firewall	Firewall	Distributed
Serving quick Web requests that do not involve EJB components	Web servers are competing for hardware resources with the application server	Web servers are dedicated	Distributed
Conflicts over responsibility	High	Low	Distributed
Loading balancing	Dispatcher	Dispatcher	Equal

Clustering on Multicore Nodes

- How should we deploy app servers on a multicore node?
- Essentially, an instance of app server is a single process
  - Unless it is implemented in a parallel way
- To cluster multiple instances of app server running on the node
  - To modify the ports used in app server
  - Or to run them individually in Virtual Machines
- For example
  - A Tomcat cluster in a single multicore node

nginx

- nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server, written by Igor Sysoev.
  - For a long time, it has been running on many heavily loaded Russian sites including
    - Yandex, Mail.Ru, VKontakte, and Rambler.
- According to Netcraft nginx served or proxied 17.82% busiest sites in April 2014.
  - Here are some of the success stories:
    - Netflix, Wordpress.com, FastMail.FM.

nginx

- Starting, Stopping, and Reloading Configuration
  - To start nginx, run the executable file.
  - Once nginx is started, it can be controlled by invoking the executable with the `-s` parameter.
  - Use the following syntax:
  - `nginx -s` *`signal`* Where *`signal`* may be one of the following:
    - `stop` — fast shutdown
    - `quit` — graceful shutdown
    - `reload` — reloading the configuration file
    - `reopen` — reopening the log files

nginx

- Configuration File’s Structure
  - nginx consists of modules which are controlled by directives specified in the configuration file.
  - Directives are divided into simple directives and block directives.
  - A simple directive consists of the name and parameters separated by spaces and ends with a semicolon (;).
  - A block directive has the same structure as a simple directive, but instead of the semicolon it ends with a set of additional instructions surrounded by braces ({ and }).
    - If a block directive can have other directives inside braces, it is called a context (examples: events, http, server, and location).
  - Directives placed in the configuration file outside of any contexts are considered to be in the main context.
    - The events and http directives reside in the main context, server in http, and location in server.

nginx

- Serving Static Content
  - First, create the `/data/www` directory and put an `index.html` file with any text content into it and create the `/data/images` directory and place some images in it.
  - Next, open the configuration file. The default configuration file already includes several examples of the `server` block, mostly commented out.

```
server {
 location / {
   root /data/www;
 }
 location /images/ {
   root /data;
 }
}
```

nginx

- Serving Static Content
  - This is already a working configuration of a server that listens on the standard port 80 and is accessible on the local machine at `http://localhost/`.
  - In response to requests with URIs starting with /images/, the server will send files from the /data/images directory.
    - For example, in response to the `http://localhost/images/example.png` request nginx will send the `/data/images/example.png` file. If such file does not exist, nginx will send a response indicating the 404 error.
  - Requests with URIs not starting with `/images/` will be mapped onto the `/data/www` directory.
    - For example, in response to the `http://localhost/some/example.html` request nginx will send the `/data/www/some/example.html` file.

nginx

- Setting Up a Simple Proxy Server
  - The configuration of a proxy server will look like this:

```
server {
 location / {
  proxy_pass http://localhost:8080/;
 }
 location ~ \.(gif|jpg|png)$ {
  root /data/images;
 }
}
```

  - This server will filter requests ending with .gif, .jpg, or .png and map them to the `/data/images` directory (by adding URI to the root directive's parameter) and pass all other requests to the proxied server configured above.

nginx

- Load balancing methods
  - The following load balancing mechanisms (or methods) are supported in nginx:
    - round-robin — requests to the application servers are distributed in a round-robin fashion,
    - least-connected — next request is assigned to the server with the least number of active connections,
    - ip-hash — a hash-function is used to determine what server should be selected for the next request (based on the client’s IP address).

nginx

- Default load balancing configuration
  - The simplest configuration for load balancing with nginx may look like the following:

```
http {
 upstream myapp1 {
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

  - Reverse proxy implementation in nginx includes load balancing for HTTP, HTTPS, FastCGI, uwsgi, SCGI, and memcached.

nginx

- Least connected load balancing

```
http {
 upstream myapp1 {
  least_conn;
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

nginx

- Session persistence
  - Please note that with round-robin or least-connected load balancing, each subsequent client's request can be potentially distributed to a different server.
    - There is no guarantee that the same client will be always directed to the same server.
  - If there is the need to tie a client to a particular application server
    - in other words, make the client's session "sticky" or "persistent" in terms of always trying to select a particular server — the ip-hash load balancing mechanism can be used.

```
http {
 upstream myapp1 {
  ip_hash;
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

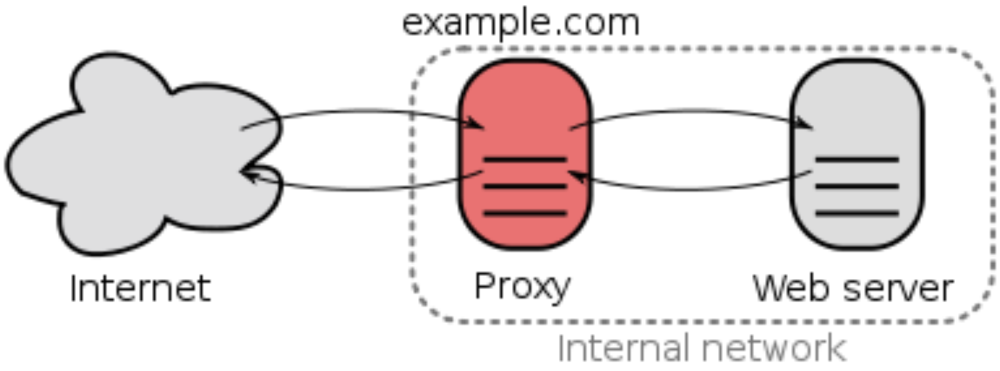
nginx

- Weighted load balancing

```
http {
 upstream myapp1 {
  server srv1.example.com weight=3;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

Reverse Proxy

- In computer networks, a **reverse proxy** is a type of proxy server that retrieves resources on behalf of a client from one or more servers.
  - These resources are then returned to the client as though they originated from the reverse proxy itself.



Uses of Reverse Proxies

- Reverse proxies can hide the existence and characteristics of the origin server(s).

- Application firewall features can protect against common web-based attacks.
  - Without a reverse proxy, removing malware or initiating takedowns, for example, can become difficult.

- In the case of secure websites, the SSL encryption is sometimes not performed by the web server itself, but is instead offloaded to a reverse proxy that may be equipped with SSL acceleration hardware.

- A reverse proxy can distribute the load from incoming requests to several servers, with each server serving its own application area.

Uses of Reverse Proxies

- A reverse proxy can reduce load on its origin servers by caching static content, as well as dynamic content.
- A reverse proxy can optimize content by compressing it in order to speed up loading times.
- In a technique known as "spoon feeding", a dynamically generated page can be produced all at once and served to the reverse-proxy, which can then return it to the client a little bit at a time.
- Reverse proxies can be used whenever multiple web servers must be accessible via a single public IP address.

Apache Reverse Proxy

- A reverse proxy, by contrast, appears to the client just like an ordinary web server.
  - No special configuration on the client is necessary.
  - The client makes ordinary requests for content in the name-space of the reverse proxy.
  - The reverse proxy then decides where to send those requests, and returns the content as if it was itself the origin.
- A typical usage of a reverse proxy is to provide Internet users access to a server that is behind a firewall.
  - Reverse proxies can also be used to balance load among several back-end servers, or to provide caching for a slower back-end server.
  - In addition, reverse proxies can be used simply to bring several servers into the same URL space.

Apache Reverse Proxy

- A reverse proxy is activated using the ProxyPass directive or the [P] flag to the RewriteRule directive. It is **not** necessary to turn ProxyRequests on in order to configure a reverse proxy.

```
ProxyRequests Off

<Proxy *>
  Order deny, allow
  Allow from all
</Proxy>

ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
```

Internationalization

- Locales
  - The local language is expressed as
  - a lowercase two-letter code,
  - following ISO 639-1

Common ISO 639-1 Language Codes

Language	Code
Chinese	zh
Danish	da
Dutch	nl
English	en
French	fr
Finnish	fi
German	de
Greek	el
Italian	it
Japanese	ja
Korean	ko
Norwegian	no
Portuguese	pt
Spanish	sp
Swedish	sv
Turkish	tr

Internationalization

- Locales
  - The country code is expressed as
  - an uppercase two-letter code,
  - following ISO 3166-1.

Common ISO 3166-1 Country Codes	
Country	Code
Austria	AT
Belgium	BE
Canada	CA
China	CN
Denmark	DK
Finland	FI
Germany	DE
Great Britain	GB
Greece	GR
Ireland	IE
Italy	IT
Japan	JP
Korea	KR
The Netherlands	NL
Norway	NO
Portugal	PT
Spain	ES
Sweden	SE
Switzerland	CH
Taiwan	TW
Turkey	TR
United States	US

Internationalization

- Locales

```
Locale german = new Locale("de");
Locale germanGermany = new Locale("de", "DE");
Locale germanSwitzerland = new Locale("de", "CH");
Locale norwegianNorwayBokmål = new Locale("no", "NO", "B");
```

- Number Formats

```
Locale loc = new Locale("de", "DE");
NumberFormat currFmt = NumberFormat.getCurrencyInstance(loc);
double amt = 123456.78;
String result = currFmt.format(amt);
```

  - The result is

```
123.456,78€
```

Internationalization

- Resource Bundles
  - When localizing an application, you'll probably have a dauntingly large number of message strings, button labels, and so on, that all need to be translated.
  - To make this task feasible, you'll want to define the message strings in an external location, usually called a resource.
  - The person carrying out the translation can then simply edit the resource files without having to touch the source code of the program.

- Locating Resource Bundles
  - for all country-specific resources, and use

    `bundleName_language`
  - You load a bundle with the command

```
ResourceBundle currentResources =
    ResourceBundle.getBundle(bundleName, currentLocale);
```

Internationalization

- Bundle Classes
  - To provide resources that are not strings, you define classes that extend the Resource Bundle class.
  - You use the standard naming convention to name your classes, for example

    MyProgramResources.java

    MyProgramResources_en.java

    MyProgramResources_de_DE.java

  - You load the class with the same `getBundle` method that you use to load a property file:

```
ResourceBundle bundle =
        ResourceBundle.getBundle("MyProgramResources", locale);
```

Project

- Requirement
  - Try to build a 2-nodes MySQL cluster, one instance for reading, and the other one for writing.
  - To add support to internationalization. For example, users can choose the language by links of Chinese and English.
  - The homepage of your Book Store should have at least two versions, however, you needn’t to add the support to the whole website of Book Store.

References

- Mastering Enterprise JavaBeans 3.0 4th Edition
  - Rima Patel Sriganesh, Gerald Brose, Micah Silverman
- Nginx
  - http://nginx.org/en/docs/beginners_guide.html
- Reverse Proxy
  - http://en.wikipedia.org/wiki/Reverse_proxy
- Apache reverse proxy
  - http://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse
- Core Java (volume 2) 9th Edition
  - by Cay S. Horstmann; Gary Cornell

Thank You!