**[Properties](http://www.guj.com.br/articles/35)**

em 24/03/2003 , por [Rodrigo Medda Pereira](mailto:rodrigo@viewtec.com.br)

**Introdução**

Existem várias classes no java que trabalham com collections. Veremos quais são as vantagens de utilizar a classe Properties. Daremos também uma breve explicação sobre o que é um mapa.

**A classe**

A classe *java.util.Properties*, presente desde a versão 1.0 do JDK, é filha (extende) da classe *java.util.Hashtable* e, por consequência, implementa a interface Map. Ela armazena pares de valores, assim como a Hashtable, com uma diferença: ambos os valores (chave e valor) são Strings. Para quem nunca usou nenhum *Map*, é interessante saber que um *Map* associa dois objetos: dado uma chave, pego um valor. Um exemplo de Mapa que poderia ser representado em java:

[view plainprint?](http://www.guj.com.br/articles/35)

1. norte = tesouro
2. sul = pirata
3. oeste = pirata
4. leste = oasis

Como podemos ver, um *Map* é um conjunto de pares chave->valor associados. Uma chave só tem um valor, mas podem existir várias chaves que tem o mesmo valor. No nosso caso, *sul* e *oeste* tem o mesmo valor. O *Properties* é um mapa especial, pois não associa dois objetos, e sim duas *Strings*. Alguns dos métodos da classe Properties:

[view plainprint?](http://www.guj.com.br/articles/35)

1. // retorna null caso não exista um objeto na coleção em que a chave
2. // tenha o valor "nome" no nosso caso, retornaria "tesouro"
3. String nome = props.getProperty("norte");
5. // caso não exista chave com o valor "sudoeste",
6. // em vez de retornar null, retorna sempre o valor "vazio"
7. String nome2 = props.getProperty("sudoeste", "vazio");
9. // adiciona ao mapa a chave "extremoSul", com valor
10. // "caverna".  Caso já exista um elemento na coleção
11. // com esta chave, o valor na coleção é  alterado para este último
12. props.setProperty("extremoSul", "caverna");
14. // retorna um Enumeration, sem ordem definida, com as chaves das propriedades
15. Enumeration enum = props.propertyNames();
17. // lê os dados de um determinado InputStream
18. // veremos neste tutorial o formato que este arquivo deve ter
19. props.load(InputStream in)
21. // o head representa a comentário que será inserido no começo do arquivo,
22. // junto com a hora em que os dados foram armazenados.
23. props.store(OutputStream out, String head)

O método save(OutputStream out, String header) está depreciado (deprecated) porque ele não gera uma IOException caso aconteça algum problema. Os métodos put() e get(), herdados da classe *Hashtable*, podem ser utilizados ao invés de setProperty() e getProperty(), desde que os objetos inseridos sejam Strings, caso contrário será gerada uma ClassCastException no momento em que for feita a leitura dos valores presentes na coleção, em razão do tipo dos objetos inseridos não serem Strings. *Nota do editor:* Se a Sun refizesse essa classe hoje em dia, com certeza ela não herdaria de *Hashtable*, ela iria ter uma composição com esta classe, fazendo assim um delegate de funcionalidade.

**Estratégia**

É possível ler pares de valores armazenados em arquivos texto, a partir do método load(), desde que seja declarado um par de valores por linha no arquivo e que a chave seja separada do valor pelo caracter igual (=). Exemplo: O arquivo mail.properties possui as seguintes configurações:

[view plainprint?](http://www.guj.com.br/articles/35)

1. #Configurações do arquivo mail.properties
2. mail.pop3.host=pop.dot.com
3. mail.debug=false
4. mail.from=from\_user@mail.dot.com
5. mail.user=to\_user
6. mail.smtp.host=smtp.dot.com
7. mail.store.protocol=pop3

Para inserir um comentário de uma linha no arquivo, utilize a cerquilha (#). Se houver gravação no arquivo, o conteúdo anterior será apagado inclusive os comentários. Convencionou-se colocar a terminação .properties a esses arquivos textos de configurações utilizados pelo java, assim como existem os arquivos .ini no windows. É possível encontrar os arquivos .properties no JBoss, Tomcat e em outros projetos. Atenção: os dados do arquivo .properties serão utilizados na aplicação de acordo com as regras da classe String, por exemplo, user é diferente de User. Utilizando *Properties*, não é possível garantir a ordem em que os dados serão armazenados, quer dizer, se vocês armazenarem a coleção do mesmo objeto properties em momentos diferentes da aplicação, o conteúdo do arquivo poderá estar em ordens diferentes. Se a ordem for importante, veja a classe *LinkedHashMap*, que foi adicionada ao jdk1.4. Ela devolve um iterator com os valores ordenados pela ordem de inserção na *HashMap*.

**Lendo um arquivo .properties**

Para ler um arquivo, é necessário criar um *InputStream* a ser lido e um objeto da classe Properties. O passo seguinte é chamar o método load() da classe Properties para ler os dados que estão no arquivo.

[view plainprint?](http://www.guj.com.br/articles/35)

1. // o arquivo encontra-se no mesmo diretório //da aplicação
2. File file = new File("mail.properties");
3. Properties props = new Properties();
4. FileInputStream fis = null;
5. try {
6. fis = new FileOutputStream(file);
7. //lê os dados que estão no arquivo
8. props.load(fis);
9. fis.close(0;
10. }
11. catch (IOException ex) {
12. System.out.println(ex.getMessage());
13. ex.printStackTrace();
14. }

Agora é possível manipular os dados que foram lidos do arquivo.

[view plainprint?](http://www.guj.com.br/articles/35)

1. String user = props.getProperty("mail.user");
2. String from = props.getProperty("mail.from");
3. String smtp = props.getProperty("mail.smtp.host");
4. String pop3 = props.getProperty("mail.pop3.host");
5. String protocol = props.getProperty("mail.store.protocol");
6. String debug = props.getProperty("mail.debug");

Para alterar o valor de user no objeto props, utilize o método setProperty():

[view plainprint?](http://www.guj.com.br/articles/35)

1. String user = "guj";
2. props.setProperty("mail.user", user);

Notem que o valor anterior de "mail.user" foi perdido. A sequência a seguir mostra como persistir os dados do objeto props em um arquivo diferente:

[view plainprint?](http://www.guj.com.br/articles/35)

1. //o arquivo encontra-se no mesmo diretório da aplicação
2. File file = new File("mail2.properties");
3. FileInputStream fos = null;
4. try {
5. fos = new FileOutputStream(file);
6. //grava os dados  no arquivo
7. props.store(fos, "Configurações do arquivo mail.properties");
8. fos.close();
9. }
10. catch (IOException ex) {
11. System.out.println(ex.getMessage());
12. ex.printStackTrace();
13. }

**Conclusão**

A utilização da classe Properties com um arquivo de configurações permite mais facilidade para manter a aplicação sem a necessidade de recompilar o código fonte a cada mudança nas configurações. Basta alterar o arquivo com as configurações e reiniciar a aplicação. Existem formas de atualizar o conteúdo do objeto properties sem reiniciar a aplicação e serão abordadas em um outro tutorial.