Codequick 1.0

2015

Codequick

Tutorial do Codequick

Torres, Marcelo

Codequick

conteúdo

[Introdução 3](#_Toc426478983)

[Comandos do Codequick 5](#_Toc426478984)

[Lista de Comandos 5](#_Toc426478985)

[@@[template:param]@@ 5](#_Toc426478986)

[@@[project:param]@@ 5](#_Toc426478987)

[@@[version:param]@@ 5](#_Toc426478988)

[@@[author:param]@@ 5](#_Toc426478989)

[@@[licence:param]@@ 5](#_Toc426478990)

[@@[fileType:param]@@ 5](#_Toc426478991)

[@@[fileModule:param]@@ 5](#_Toc426478992)

[@@[filePathparam]@@ 5](#_Toc426478993)

[@@[filePrefix:param]@@ 6](#_Toc426478994)

[@@[fileSufix:param]@@ 6](#_Toc426478995)

[@@[content:param]@@ 6](#_Toc426478996)

[@@[getVar:param]@@ 6](#_Toc426478997)

[@@[get:param]@@ 6](#_Toc426478998)

[@@[if:param1 operador param2@@;param3@@;param4]@@ 8](#_Toc426478999)

[@@[loop:param1@@;param2 filter:param3]@@ 8](#_Toc426479000)

# Introdução

Codequick é um software open source desenvolvido para gerar código a partir do banco de dados. A idéia é que o trabalho de gerar código repetitivo possa ser minimizado através do Codequick.

Em geral, sistemas que acessam banco de dados, principalmente sistemas web, possuem uma arquitetura que pode resultar na criação de diversos arquivos para que um cadastro possa ser efetuado em uma tabela. Levando-se em conta que a maioria dos sistemas possuem muitas tabelas, o resultado seria a necessidade de se criar muitos arquivos, com um código que segue um padrão e que, pelo menos a parte do CRUD (create, read, update e delete) poderia ser feita de forma rápida e automatizada.

Com o Codequick, o desenvolvedor poderá criar suas configurações para gerar código para seu sistema, seja ele em JAVA, C# ou outra linguagem e para o banco de dados desejado, ou seja, para Oracle, DB2, MS-SQL Server, PostgreSQL e etc.

O desenvolvedor também poderá gerar código para sua arquitetura específica. Codequick trabalha com arquivos templates que seguem a arquitetura desejada e no lugar das informações que veem do banco, como nome de tabelas, campos, tipos, tamanho, nome de classes e propriedades que estão de acordo com a nomenclatura do banco, deverão ser substituídos por comandos específicos para que o Codequick possa processar junto das informações das tabelas e criar os arquivos desejados.

O fluxo básico para se gerar arquivos usando o Codequick segue mais ou menos esse:

Figura - Fluxo básico de operação

A arquitetura do sistema é fundamental para se obter resultados usando o Codequick. Por exemplo, sistemas web em JAVA geralmente tem uma arquitetura bem definida e que possuem muitos arquivos que se repetirão para todos os cadastros, como:

* Beans de mapeamento Objeto / Tabela (Model, Value Object, etc – Armazena as informações de um registro da tabela correspondente)
* Mapper (Faz o mapeamento de qual campo corresponde a qual propriedade, caso não seja feito no Model)
* Interface DAO (Data Access Object – camada de acesso ao banco)
* Implementação DAO
* Interface Service (EJB, Service, Business e etc – camada da lógica de negócio)
* Implementação Service
* Action – (No Struts contém a lógica de controle do MVC)
* Form – (No Struts corresponde ao Bean usado para receber e enviar dados para a tela)
* Mbeans (No JSF contém a lógica de controle do MVC e as propriedades para receber e enviar dados para a tela)
* JSP / JSF – (Página com as informações que serão apresentadas para os usuários)
* JS – (Arquivo Javascript)
* ApplicationResources.properties – (Arquivo que contém chaves de tradução para internacionalização)
* Struts-config.xml (No Struts contém a configuração das Actions e Forms)
* Tiles-Defs.xml (Arquivo de configuração de template de página do Tiles)

Esses são alguns arquivos que podem ser necessário criar ou alterar para cada cadastro do sistema. Daí a importância do gerador de código gerar esses arquivos automaticamente reduzindo o tempo gasto pelo desenvolver, consequentemente reduzindo prazo e custo do sistema.

É necessário observar, entretanto, que o Codequick faz um trabalho sem dúvida importante, mas que requer atenção do desenvolvedor e, eventualmente, ajuste para que o resultado seja o mais próximo do esperado.

Por se tratar de um produto com poucas horas de desenvolvimento, algumas coisas poderiam ser melhoradas, como, por exemplo, tratar os relacionamentos do banco de forma automática, criando as associações nas classes correspondentes. O desenvolvedor terá que alterar o código para incluir esses relacionamentos nas classes. Além disso, o código gerado depente do template desenvolvido, sendo que, pode não ficar exatamente como esperado e necessitar de algum ajuste.

O Codequick (2015) foi desenvolvido em JAVA a partir do código do Codewiz (2012). O Codewiz, por sua vez, foi inspirado por um outro gerador desenvolvido em C# (2004), chamado Codebuilder. O Codebuilder, por sua vez, teve uma versão em Visual Basic por volta de 1998. Portanto, a necessidade de um gerador de código que seja flexível para se trabalhar com uma arquitetura específica, sempre existiu. E o resultado sempre foi positivo, mesmo quando haviam ajustes para serem feitos no código gerado. Os geradores de código nos ajudaram a ganhar tempo em diversos projetos, deixando de lado aquele trabalho de listar os campos do banco de dados de cada tabela e criar as classes correspendentes e, ao gerar todo o código para um cadastro, ajudou a reduzir o tempo de desenvolvimento substancialmente, principalmente para projetos com muitos cadastros.

# Comandos do Codequick

Os arquivos de template devem ser criados a partir de um exemplo funcional de código. A partir daí, é necessário incluir alguns comandos que serão interpretados pelo Codequick para poder gerar o código correspondente.

Os comandos possuem o seguinte padrão:

**@@[** - inicia um comando

**]@@** - encerra um comando

**@@;** - separa parâmetros em um comando com mais de um parâmetro

E possuem a seguinte sintaxe:

**@@[nome\_do\_comando:parâmetro]@@** - comando simples

**@@[nome\_do\_comando:parâmetro1@@;parâmetro2]@@** - comando simples com mais de um parâmetro

**@@[nome\_do\_comando:@@[nome\_do\_comando2:parâmetro]@@]@@** - comando simples que recebe o resultado de outro comando como parâmetro

**@@[nome\_do\_comando:parâmetro@@;nome\_do\_comando2:parâmetro2]@@** - comando composto

## Lista de Comandos

Codequick possui a seguinte lista de comandos na versão 1.0:

### @@[template:param]@@

Informarção sobre do que se trata o template. Informativo, porém obrigatório.

### @@[project:param]@@

Nome do projeto. Informativo, porém obrigatório.

### @@[version:param]@@

Versão do template. Informativo, porém obrigatório.

### @@[author:param]@@

Autor do template. Informativo, porém obrigatório.

### @@[licence:param]@@

Licença do template. Informativo, porém obrigatório.

### @@[fileType:param]@@

Informação sobre o tipo de template, ou seja, que tipo de arquivo será gerado (Java, C#, Procedure e etc).

### @@[fileModule:param]@@

Módulo que serão gerados os arquivos. Será criado uma pasta dentro do diretório de build com o modulo informado no param.

### @@[filePathparam]@@

Path em que serão gerados os arquivos dentro do módulo. Geralmente aqui é usado uma variável de usuário definida no arquivo config.properties que contém a estrutura de pacotes onde os arquivos devem ser gerados.

### @@[filePrefix:param]@@

Prefixo do nome do arquivo a ser gerado. O nome do arquivo gerado é composto do prefixo + nome da classe (nome da tabela ‘formatado’) + sufixo. Exemplo:

Tabela: tb\_user

Classe: User

Sufixo: ServiceImpl.java

@@[filePrefixo:]@@

Nome do arquivo gerado: UserServiceImpl.java

### @@[fileSufix:param]@@

Sufixo do nome do arquivo a ser gerado. O nome do arquivo gerado é composto do prefixo + nome da classe (nome da tabela ‘formatado’) + sufixo. Exemplo:

Tabela: tb\_user

Classe: User

Prefixo: (sem nada)

@@[fileSufix:ServiceImpl.java]@@

Nome do arquivo gerado: UserServiceImpl.java

### @@[content:param]@@

Conteúdo do arquivo que será processado e para gerar o conteúdo final. Apenas o conteúdo será incluído no arquivo gerado, ou seja, o conteúdo deve ser formado pelo template desejado.

### @@[getVar:param]@@

Obém o valor da variável definida no arquivo de configuração. Exemplo:

Variável definida no arquivo de configuração:

author=Codequick

@@[getVar:author]@@

Resultado: Codequick

### @@[get:param]@@

Obém o valor do parâmetro. O parâmetro deve ser uma propriedade dos objetos tableDef ou columnDef. Abaixo a tabela das propriedades disponíveis nos objetos tableDef e columnDef:

|  |  |
| --- | --- |
| **tableDef** | |
| **Propriedades da Tabela** | **Descrição** |
| catalog |  |
| schema | Schema da tabela |
| name | Nome da tabela |
| type |  |
| remarks |  |
| typecatalog |  |
| typeschema |  |
| typename |  |
| selfreferencingcolname |  |
| refgeneration |  |
| **Propriedades Extras** | **Descrição** |
| packageName |  |
| className | Nome da classe correspondente a tabela |
| pkName |  |
| columnDefs | Lista de colunas da tabela |

|  |  |
| --- | --- |
| **columnDef** | |
| **Propriedades da Coluna** | **Descrição** |
| catalog |  |
| schema |  |
| tablename |  |
| columnname |  |
| datatype |  |
| typename |  |
| columnsize | Indica o tamanho do campo |
| decimaldigits | Indica a quantidade de casas decimais |
| radix |  |
| nullable | Indica se o campo aceita null ou não |
| remarks |  |
| columndefault |  |
| charoctetlength |  |
| ordinalposition |  |
| isnullable |  |
| scopecatalog |  |
| scopeschema |  |
| scopetable |  |
| sourcedatatype |  |
| isautoincrement |  |
| **Propriedades Extras** | **Descrição** |
| shortColumnName | Nome da coluna sem o prefixo removível definido no arquivo de configuração |
| propertyName | Nome da propriedade |
| propertyType | Tipo da propriedade conforme o tipo do banco e a linguagem |
| fkTableDef |  |
| tableDef | Referência para tableDef para acessar suas propriedades |
| fieldType | Tipo do campo do banco de dados |
| keySeq | Sequencia do campo na chave primária |
| identity | Valor true ou false que indica se a coluna é identity |

As propriedades da tabela e da coluna, para que estejam disponíveis nos templates, devem ser incluídas no arquivo de configuração nos campos “exportTableInfo” e “exportColumnInfo”, separados por vírgula.

As propriedades extras são criadas pelo Codequick para facilitar o desenvolvedor ao criar seus templates.

Exemplo:

Tabela: tb\_user

@@[get:tableDef.name]@@

Resultado: tb\_user

### @@[if:param1 operador param2@@;param3@@;param4]@@

Instrução de condição que avalia o param1 e param2 em relação ao operador e retorna o param3 no caso da avaliação da condição for verdadeira ou retorna o param4 no caso da condição negativa.

Exemplo:

Schema: security

Tabela: tb\_user

@@[if:@@[get:tableDef.schema]@@ ne null@@;.@@;]@@@@[get:tableDef.name]@@

Resultado: security.tb\_user

### @@[loop:param1@@;param2 filter:param3]@@

Instrução de iteração que avalia o param1, que deve ser uma lista, itera a lista executando o param2 para cada iteração. Também é possível, opcionalmente, filtrar os itens iterados incluindo a cláusula opcional filter:param3 onde o param3 deve ser uma condição verdadeira para que o item seja iterado.

Exemplo:

Schema: security

Tabela: tb\_user

@@[loop:tableDef.columnDefs@@;

public static final String @@[get:columnDef.shortColumnName]@@ = "@@[get:columnDef.propertyName]@@";

]@@

Resultado: security.tb\_user

@@[item:first]@@

@@[upperFirst:@@[get:columnDef.propertyName]@@]@@

@@[lowerFirst:@@[get:tableDef.className]@@]@@

@@[isNull:columnDef.keySeq]@@

@@[isNotNull:columnDef.keySeq]@@

@@[not:@@[get:columnDef.identity]@@]@@

@@[isNumeric:@@[get:columnDef.propertyType]@@]@@

@@[replaceAll:@@[getVar:jsFolder]@@@@;\.@@;/]@@

@@[isString:@@[get:columnDef.propertyType]@@]@@

@@[delStart:]@@

@@[delEnd:]@@