

Введение в платформу веб-инструментария Google Web Toolkit

PRINTABLE VERSION

Веб-инструментарий Google Web Toolkit (GWT) — платформа разработки с открытым исходным кодом, позволяющая разработчикам просто создавать высокопроизводительные приложения AJAX с использованием Java. GWT позволяет создавать интерфейсную часть на Java; он компилирует исходный код в код JavaScript и HTML, совместимый с браузерами, и с высокой оптимизацией. "Сегодня создание веб-приложений — это утомительный и подверженный ошибкам процесс. Вы тратите 90% времени на решении проблем браузера, а отсутствие модульности JavaScript делает совместный доступ, тестирование и повторное использование компонентов AJAX сложным и нестабильным. Это не обязательно должно быть так". См. веб-сайт [Google Web Toolkit](http://www.gwtproject.org/).

В этом учебном курсе вы узнаете, как указанные выше принципы применяются к реальным приложениям. Одновременно описывается поддержка IDE NetBeans для GWT и создается простое приложение, в котором используются некоторые из этих функций.

Содержание

- Настройка среды
 - Создание исходной структуры приложения GWT
 - Рассмотрение исходной структуры приложения GWT
- Создание средства случайных цитат AJAX
 - Создание заглушек службы
 - Проверка созданных классов
 - Расширение созданных классов
 - Настройка внешнего вида
- Компиляция и отладка
- Заключение
- Дополнительные сведения

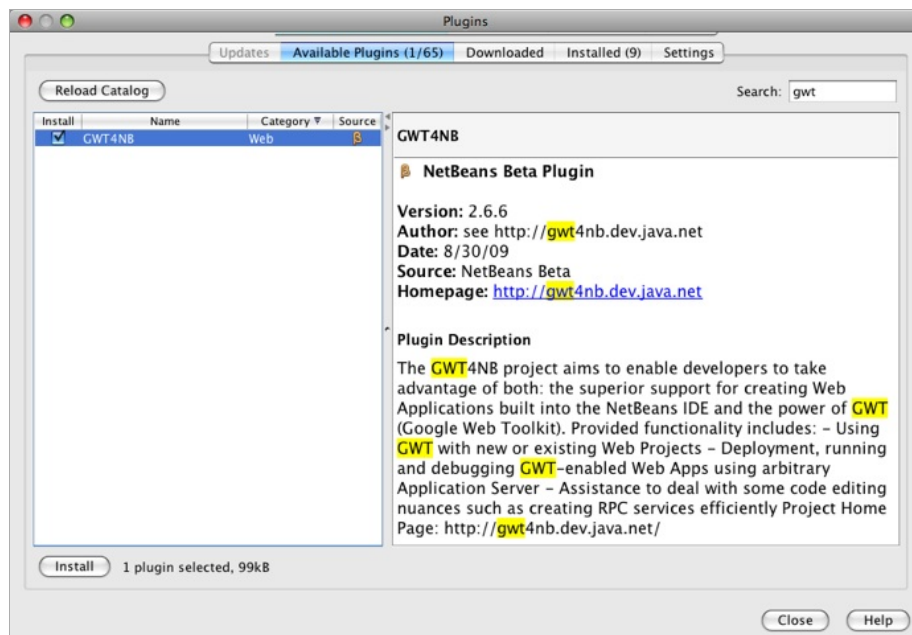


Для работы с этим учебным курсом требуются программное обеспечение и ресурсы, перечисленные ниже.

Программное обеспечение или материал	Требуемая версия
IDE NetBeans, комплект Java	версия 6.x
Комплект для разработчика на языке Java (JDK)	версия 5 или более поздняя
Сервер GlassFish или Контейнер сервлетов Tomcat	v3 или Open Source Edition 3.0.1 версия 6.x
Веб-инструментарий Google Web Toolkit (GWT)	версия 1.5 или более поздняя
Подключаемый модуль NetBeans GWT	версия 2.x

Примечания:

- Пакет загрузки Java позволяет установить сервер GlassFish и контейнер сервлетов Apache Tomcat 6.0.x. Их необходимо установить для работы с этим учебным курсом.
- Вместо загрузки подключаемого модуля NetBeans GWT с веб-сайта <https://gwt4nb.dev.java.net/> его необходимо загрузить и установить непосредственно из диспетчера подключаемых модулей IDE. Выберите "Сервис > Подключаемые модули" в главном меню и установите подключаемый модуль, как показано ниже:



Более подробные инструкции по установке подключаемого модуля в платформы в IDE приведены в разделе [Добавление поддержки веб-платформы](#).

[Download NetBeans IDE](#)

Training

- [Java Programming Language](#)



Support

- [Oracle Development Tools Support Offering for NetBeans IDE](#)

Documentation

- [General Java Development](#)
- [External Tools and Services](#)
- [Java GUI Applications](#)
- [Java EE & Java Web Development](#)
- [Web Services Applications](#)
- [NetBeans Platform \(RCP\) and Module Development](#)
- [PHP and HTML5 Applications](#)
- [C/C++ Applications](#)
- [Mobile Applications](#)

- [Sample Applications](#)
- [Demos and Screenshots](#)

More

- [FAQs](#)
- [Contribute Documentation!](#)
- [Docs for Earlier Releases](#)

- Можно загрузить пример рабочего приложения для этого учебного курса, а также другие приложения с использованием GWT.
- Дополнительные сведения о GWT приведены на веб-сайте <http://code.google.com/webtoolkit/>. Сведения о поддержке GWT в среде IDE приведены на веб-сайте <https://gwt4nb.dev.java.net/>. Если вы знакомы с GWT, вы можете отправить свой код для проекта подключаемого модуля GWT.
- В этом учебном курсе используются некоторые примеры, представленные в книге "Google Web Toolkit: GWT Java AJAX Programming" Прабхакара Чаганти, издательство Packt Publishing, февраль 2007 г..

Установка среды


Для начала использования IDE необходимо создать основную структуру исходных кодов. После создания ее можно подробно изучить для понимания внутренней работы GWT.

- Создание исходной структуры приложения GWT
- Рассмотрение исходной структуры приложения GWT


Создание исходной структуры приложения GWT

Исходная структура создаваемого приложения должна включать файлы JAR GWT, файл настройки проекта модуля GWT, а также некоторые стандартные артефакты, такие как точка входа Java. Поскольку используется среда IDE, не требуется создавать все эти файлы вручную. Вместо этого всю работу выполнит мастер. В частности, последняя панель мастера нового веб-приложения очень полезна в контексте создания приложения GWT.

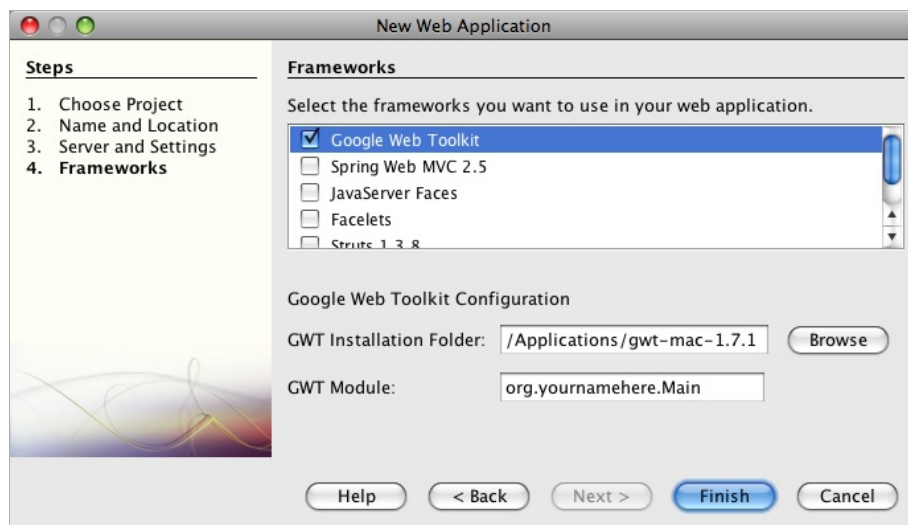
1. Выберите команды Файл > Новый проект (Ctrl-Shift-N; ⌘-Shift-N для Mac). В разделе "Категории" выберите "Веб" (или "Java Web"). В области "Projects" выберите "Web Application". Нажмите кнопку "Далее".
2. В действии 2, "Имя и местоположение", введите HelloGWT для имени проекта. Также можно указать расположение проекта, введя путь в поле "Местоположение проекта". Нажмите кнопку "Далее".
3. В шаге "Сервер и параметры" выберите сервер, зарегистрированный в IDE. При установке среды IDE были включены серверы Tomcat и GlassFish, они отображаются в раскрывающемся списке.

 Чтобы зарегистрировать сервер в среде IDE, нажмите кнопку "Добавить", чтобы открыть мастер выполнения процесса регистрации.

4. Укажите используемую версию Java. Нажмите кнопку "Далее".

 **Примечание.** Этот учебный курс поддерживает GWT версии 1.5 и более поздние версии. GWT 1.4 не поддерживает Java EE 5, поэтому при использовании этой версии необходимо также задать для версии Java EE значение 1.4. В противном случае, например, аннотации Java EE 5 вызовут ошибки компиляции.

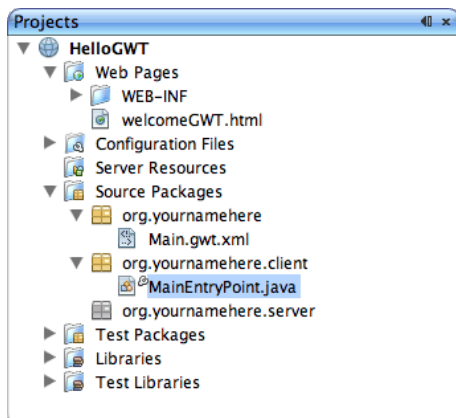
5. В действии "Платформы" выберите GWT.



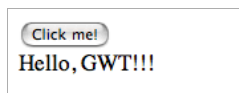
При выборе платформы GWT становятся доступными следующие поля:

- **GWT Installation Folder (Папка установки GWT):** указывается путь к папке, в которую был загружен веб-инструментарий Google Web Toolkit в начале этого курса. Если указан неверный путь, отображается красное сообщение об ошибке, и работу мастера нельзя завершить.
- **GWT Module (Модуль GWT):** указывается имя и расположение модуля проекта, который будет создан средой IDE при завершении работы мастера. Модуль проекта — это файл XML для настройки приложения GWT. Например, он используется для указания класса, экземпляр которого создается GWT при загрузке модуля. Обратите внимание, что это поле в мастере также определяет основной пакет приложения. По умолчанию основной пакет — org.yournamehere, а модуль проекта — Main. Для целей этого учебного курса оставьте поля по умолчанию без изменений.

6. Нажмите кнопку "Завершить". Средой IDE будет создан проект HelloGWT. Проект содержит все исходные файлы, библиотеки и метаданные проекта, такие как сценарий сборки Ant проекта. Проект откроется в среде IDE. Можно просмотреть его файловую структуру в окне "Файлы" (Ctrl-2; ⌘-2 в Mac) и логическую структуру в окне "Проекты" (Ctrl-1; ⌘-1 в Mac).



7. В окне "Проекты" щелкните правой кнопкой мыши узел проекта и выберите команду "Выполнить". Создается приложение и веб-архив (WAR). Выполняется его развертывание на сервере. Запустится сервер, если он еще не был запущен. Откроется браузер компьютера по умолчанию, и отобразится страница приветствия приложения.



Нажмите кнопку, текст под ней исчезнет.



В следующем разделе будут подробно рассмотрены созданные файлы и созданное выше простое приложение.

Рассмотрение исходной структуры приложения GWT

Мастер создания нового веб-приложения IDE создал несколько исходных файлов. Взгляните на файлы и посмотрите, как они соотносятся в контексте приложения GWT.

- **Main.gwt.xml**: файл XML модуля проекта, содержащийся в корневом пакете проекта, это файл XML, содержащий всю настройку приложения, необходимую для проекта GWT. Модуль проекта по умолчанию, созданный мастером, выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
  <inherits name="com.google.gwt.user.User"/>
  <entry-point class="org.yournamehere.client.MainEntryPoint"/>
  <!-- Do not define servlets here, use web.xml -->
</module>
```

Ниже приведены элементы в модуле проекта по умолчанию:

- **inherits**: указывает модули, наследуемые этим модулем. В этом простом случае наследуется только функциональность, предоставленная модулем **User**, который встроен в платформу GWT. Когда приложение становится более сложным, наследование модуля позволяет быстро и просто повторно использовать части функций.
- **entry-point**: ссылается на класс, экземпляр которого будет создан платформой GWT при загрузке модуля.



Примечание. Дополнительные сведения см. в разделе [Структурирование проектов: файлы модуля XML](#).

- **MainEntryPoint.java**: класс соответствует основной точке входа приложения, как указано в **Main.gwt.xml**. Он расширяет класс **EntryPoint**, и при загрузке модуля GWT платформой создается экземпляр класса, затем автоматически вызывается его метод **onModuleLoad()**. Точка входа проекта по умолчанию, созданная мастером, выглядит следующим образом:

```

package org.yournamehere.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;

public class MainEntryPoint implements EntryPoint {

    /** Creates a new instance of MainEntryPoint */
    public MainEntryPoint() {
    }

    /**
     * The entry point method, called automatically by loading a module
     * that declares an implementing class as an entry-point
     */
    public void onModuleLoad() {
        final Label label = new Label("Hello, GWT!!!");
        final Button button = new Button("Click me!");

        button.addClickListener(new ClickListener(){
            public void onClick(Widget w) {
                label.setVisible(!label.isVisible());
            }
        });

        RootPanel.get().add(button);
        RootPanel.get().add(label);
    }
}

```

В приведенном выше фрагменте кода метод `EntryPoint` по умолчанию `onModuleLoad()` добавляет к приложению следующие компоненты:

- **Метка:** Создается новая метка GWT, в которой отображается текст `'Hello, GWT!!!'`. Метка добавляется к `RootPanel` в последней строке кода, `RootPanel.get().add(label)`.
- **Кнопка:** создается новая кнопка GWT, на которой отображается текст `"Click me!"`, вместе с прослушивающим процессом кнопки, реализуемым `ClickListener`. Прослушиватель кнопки указывает, что при нажатии кнопки скрывается метка.

```

public void onClick(Widget w) {
    label.setVisible(!label.isVisible());
}

```

Кнопка добавляется к `RootPanel` в предпоследней строке кода.

```

RootPanel.get().add(button)

```

- **welcomeGWT.html:** созданная страница узла HTML, являющаяся специальным файлом приветствия для приложения. Файл `web.xml` использует элемент `welcome-file` для указания на то, что главная страница — это исходная страница, отображаемая в браузере при развертывании приложения. Главная страница ссылается на путь к исходному коду JavaScript и может ссылаться на таблицу стилей приложения. Главная страница по умолчанию, созданная мастером, выглядит следующим образом:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta name='gwt:module' content='org.yournamehere.Main=org.yournamehere.Main'>
  <title>Main</title>
</head>
<body>
  <script language="javascript" src="org.yournamehere.Main/org.yournamehere.Main.nocache.js"></script>
</body>
</html>

```

Теги `meta` и `script` во фрагменте кода выше имеют особое значение для GWT:

- **meta:** указывает на каталог проекта приложения. Этот тег предоставляет ссылку между страницей HTML и приложением.
- **script:** импорт кода для файла JavaScript платформы GWT. Этот код содержит необходимый код для начальной загрузки платформы GWT. Он использует настройку в модуле проекта, а затем динамически загружает код JavaScript, созданный путем компиляции точки входа, для представления приложения. Файл JavaScript создается платформой GWT при запуске приложения в размещенном режиме или при компиляции приложения.

В этом разделе на веб-странице будет отображена случайная цитата. Этот пример приложения предназначен для ознакомления с различными компонентами приложения GWT. Случайная цитата будет выбрана из списка цитат, хранящегося на сервере. Каждую секунду приложение получает случайную цитату, предоставленную сервером, и отображает ее на веб-странице в подлинном стиле AJAX, то есть, без необходимости обновления пользователем страницы.

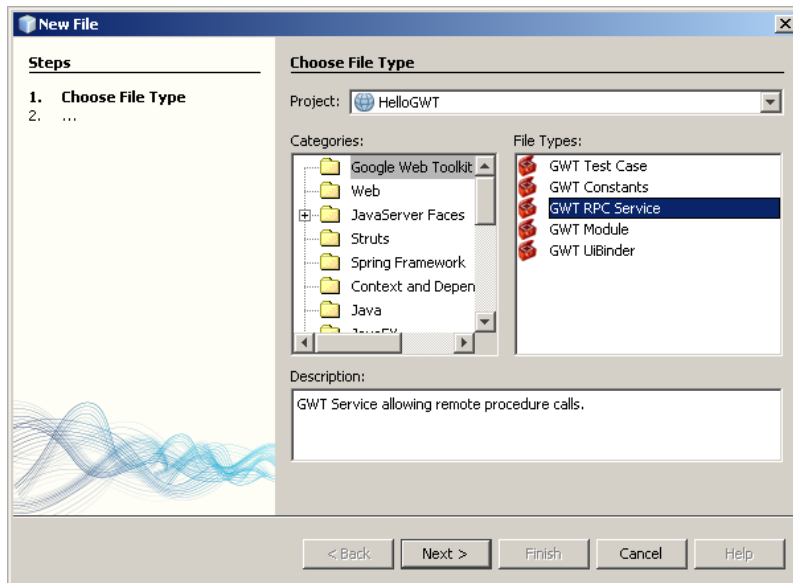
При создании этой функции используется служба RPC (удаленный вызов процедуры) GWT.

- [Создание заглушек службы](#)
- [Проверка созданных классов](#)
- [Расширение созданных классов](#)
- [Настройка внешнего вида](#)

Создание заглушек службы

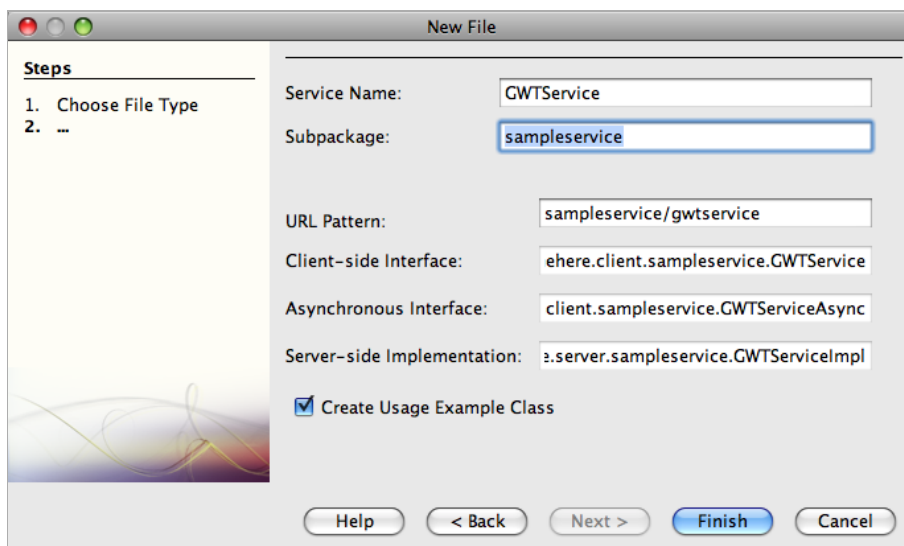
Подключаемый модуль GWT в NetBeans предоставляет мастер для создания службы RPC. Мастер создает основные классы службы. В этом подразделе описывается мастер службы RPC GWT.


1. Щелкните значок 'Создать файл' (📄) на главной панели инструментов IDE. В мастере нового файла в категории Google Web Toolkit отображается шаблон файла с именем GWT RPC Service.



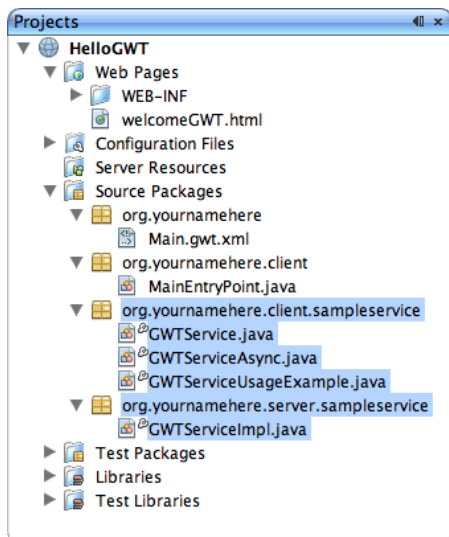
Выберите службу RPC GWT и нажмите кнопку "Далее".

2. Дополнительно можно заполнить дочерний пакет, где будут сохранены созданные файлы. Для целей этого учебного курса в поле "Дочерний пакет" введите `sampleservice`.



 **Примечание.** Если оставить параметр 'Создать класс примера использования' выделенным в этом шаге, то в IDE будет включена поддержка создания класса `GWTServiceUsageExample`, который можно использовать для вызова службы.

3. Нажмите кнопку "Завершить". Создаются файлы, перечисленные в мастере создания новой службы RPC GWT (показан на изображении выше), окно "Проекты" автоматически обновляется для отражения изменений.



Проверка созданных классов

Мастер службы RPC GWT создает несколько исходных файлов. Взгляните на файлы и посмотрите, как они соотносятся в контексте службы GWT.

Расширенное описание классов службы GWT приведено в разделе [Создание служб](#).

- **GWTService**: определение службы на стороне клиента. Этот интерфейс расширяет интерфейс тега [RemoteService](#).

```
package org.yournamehere.client.sampleservice;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("sampleservice/gwtservice")
public interface GWTService extends RemoteService {
    public String myMethod(String s);
}
```

- **GWTServiceImpl**: сервлет, реализующий интерфейс GWTService и предоставляющий возможность получения случайной цитаты с помощью RPC.

```
package org.yournamehere.server.sampleservice;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import org.yournamehere.client.sampleservice.GWTService;

public class GWTServiceImpl extends RemoteServiceServlet implements GWTService {

    public String myMethod(String s) {
        // Do something interesting with 's' here on the server.
        return "Server says: " + s;
    }
}
```

- **GWTServiceAsync**: асинхронный интерфейс, основанный на исходном интерфейсе GWTService. Он предоставляет объект обратного вызова, обеспечивающий асинхронную связь между сервером и клиентом.

```
package org.yournamehere.client.sampleservice;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface GWTServiceAsync {
    public void myMethod(String s, AsyncCallback<String> callback);
}
```

- **GWTServiceUsageExample**: пример пользовательского интерфейса, созданный как тестовый клиент. Он может использоваться для вызова службы.

```

package org.yournamehere.client.sampleservice;

import com.google.gwt.core.client.GWT;

import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.rpc.ServiceDefTarget;

import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.Widget;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;

import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;

public class GWTSERVICEUsageExample extends VerticalPanel {
    private Label lblServerReply = new Label();
    private TextBox txtUserInput = new TextBox();
    private Button btnSend = new Button("Send to server");

    public GWTSERVICEUsageExample() {
        add(new Label("Input your text: "));
        add(txtUserInput);
        add(btnSend);
        add(lblServerReply);

        // Create an asynchronous callback to handle the result.
        final AsyncCallback<String> callback = new AsyncCallback<String>() {
            public void onSuccess(String result) {
                lblServerReply.setText(result);
            }

            public void onFailure(Throwable caught) {
                lblServerReply.setText("Communication failed");
            }
        };

        // Listen for the button clicks
        btnSend.addClickHandler(new ClickHandler(){
            public void onClick(ClickEvent event) {
                // Make remote call. Control flow will continue immediately and later
                // 'callback' will be invoked when the RPC completes.
                getService().myMethod(txtUserInput.getText(), callback);
            }
        });
    }

    public static GWTSERVICEAsync getService() {
        // Create the client proxy. Note that although you are creating the
        // service interface proper, you cast the result to the asynchronous
        // version of the interface. The cast is always safe because the
        // generated proxy implements the asynchronous interface automatically.

        return GWT.create(GWTSERVICE.class);
    }
}

```

Теперь измените класс точки входа для вызова службы путем создания экземпляра объекта `GWTSERVICEUsageExample`. Как указывалось в предыдущем подразделе, класс `GWTSERVICEUsageExample` был создан из-за выбора параметра ["Create Usage Example Class"](#) (Создать класс примера использования) в мастере GWT RPC.

1. В методе `onModuleLoad()` основной точки входа приложения (`MainEntryPoint.java`) удалите Метку и Кнопку GWT и добавьте новый экземпляр `GWTSERVICEUsageExample` к `RootPanel`.

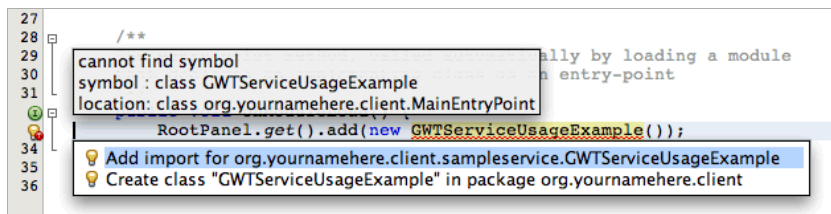
```

public void onModuleLoad() {
    RootPanel.get().add(new GWTSERVICEUsageExample());
}

```



Примечание. После изменения метода `onModuleLoad()` необходимо добавить оператор импорта к классу `sampleservice.GWTSERVICEUsageExample`. Для этого щелкните подсказку в левом столбце, где метод `GWTSERVICEUsageExample()` отображается в редакторе, и выберите "Добавить импорт" для `org.yournamehere.client.sampleservice.GWTSERVICEUsageExample`.



2. В окне 'Проекты' щелкните правой кнопкой мыши узел проекта и выберите 'Выполнить'. Запустится сервер, если он еще не был запущен. Выполняется компиляция (в данном случае перекомпиляция) и развертывание проекта на сервере. Открывается браузер, в котором отображается текстовое поле. Введите сообщение и нажмите кнопку. Появится метка с отправленным сообщением.

Input your text:

Send to server

Server says: All you need is love

Была успешно создана служба RPC GWT с помощью мастера RPC GWT среды IDE. После этого был добавлен экземпляр `GWTServiceUsageExample` к методу `onModuleLoad()` главной точки входа приложения для вызова приложением службы при выполнении. В следующем разделе выполняется настройка службы путем расширения созданных классов и прикрепления таблицы стилей к странице узла HTML.

Расширение созданных классов

В этом разделе выполняется расширение классов, проверенных в предыдущем подразделе. В конце этого подраздела будет создана работающая версия средства создания случайных цитат AJAX.

1. Как вы помните, `GWTServiceImpl` — это сервлет, реализующий создаваемую службу.

TIP Если открыть дескриптор развертывания `web.xml` приложения, можно увидеть, что описание и сопоставление сервлета уже добавлены.

```
<servlet>
  <servlet-name>GWTService</servlet-name>
  <servlet-class>org.yournamehere.server.sampleservice.GWTServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>GWTService</servlet-name>
  <url-pattern>/org.yournamehere.Main/sampleservice/gwtservice</url-pattern>
</servlet-mapping>
```

В классе `GWTServiceImpl` реализуется интерфейс `GWTService` с логикой для вашей службы. Для создания средства создания случайных цитат добавьте следующий код к `GWTServiceImpl`:

```
public class GWTServiceImpl extends RemoteServiceServlet implements GWTService {

    private Random randomizer = new Random();
    private static final long serialVersionUID = -15020842597334403L;
    private static List quotes = new ArrayList();

    static {
        quotes.add("No great thing is created suddenly - Epictetus");
        quotes.add("Well done is better than well said - Ben Franklin");
        quotes.add("No wind favors he who has no destined port - Montaigne");
        quotes.add("Sometimes even to live is an act of courage - Seneca");
        quotes.add("Know thyself - Socrates");
    }

    public String myMethod() {
        return (String) quotes.get(randomizer.nextInt(5));
    }

}
```

Примечание. Щелкните правой кнопкой мыши в любом месте в редакторе IDE и выберите 'Исправить импорт', чтобы создать в IDE правильные операторы импорта. При этом необходимо выбрать `java.util.Random` вместо `com.google.gwt.user.client.Random`:



- Вместо того, чтобы служба вызывалась созданным классом примером использования (`GWTServiceUsageExample`), вызовите ее напрямую из класса точки входа приложения (`MainEntryPoint`). Начните с копирования метода `getService()` `GWTServiceUsageExample` и вставьте его в `MainEntryPoint`. (Изменения **выделены полужирным шрифтом**.)

```
public class MainEntryPoint implements EntryPoint {

    /**
     * Creates a new instance of MainEntryPoint
     */
    public MainEntryPoint() {
    }

    public static GWTServiceAsync getService() {
        // Create the client proxy. Note that although you are creating the
        // service interface proper, you cast the result to the asynchronous
        // version of the interface. The cast is always safe because the
        // generated proxy implements the asynchronous interface automatically.

        return GWT.create(GWTService.class);
    }

    ...
}
```

- Щелкните в редакторе правой кнопкой мыши и выберите "Исправить операторы импорта". К `MainEntryPoint` добавляются три следующих оператора импорта.

```
import com.google.gwt.core.client.GWT;
import org.yournamehere.client.sampleservice.GWTService;
import org.yournamehere.client.sampleservice.GWTServiceAsync;
```

- Измените метод `onModuleLoad()` в классе точки входа на следующее:

```
/**
 * The entry point method, called automatically by loading a module
 * that declares an implementing class as an entry-point
 */

public void onModuleLoad() {

    final Label quoteText = new Label();

    Timer timer = new Timer() {

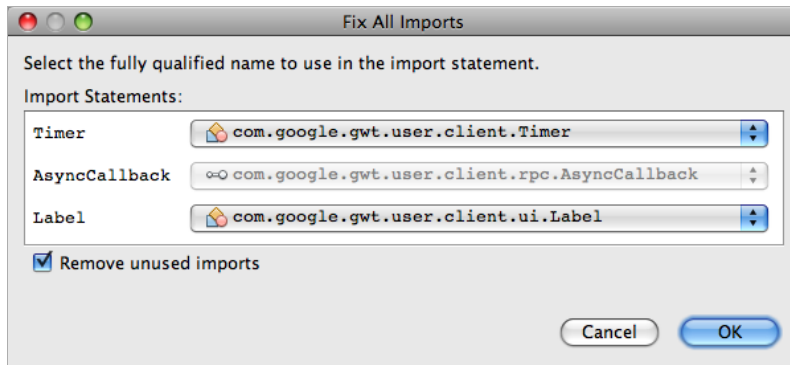
        public void run() {
            //create an async callback to handle the result:
            AsyncCallback callback = new AsyncCallback() {

                public void onFailure(Throwable arg0) {
                    //display error text if we can't get the quote:
                    quoteText.setText("Failed to get a quote");
                }

                public void onSuccess(Object result) {
                    //display the retrieved quote in the label:
                    quoteText.setText((String) result);
                }
            };
            getService().myMethod(callback);
        }
    };

    timer.scheduleRepeating(1000);
    RootPanel.get().add(quoteText);
}
```

- Щелкните в редакторе правой кнопкой мыши и выберите "Исправить операторы импорта". При этом необходимо выбрать `com.google.gwt.user.client.Timer` и `com.google.gwt.user.client.ui.Label`.



- Удалите класс `GWTServiceUsageExample`. Он больше не будет компилироваться. Поскольку приложение может вызвать службу из класса основной точки входа, класс примера использования `GWTServiceUsageExample` больше не требуется для вызова службы.
- Хотя созданные заглушки для `GWTService` и `GWTServiceAsync` предоставляют параметр `String` для `myMethod()`, он не требуется для средства создания случайных цитат.

В классе `GWTService` удалите параметр `String` из `myMethod()`, чтобы интерфейс выглядел следующим образом.

```
public interface GWTService extends RemoteService {
    public String myMethod();
}
```

- Сигнатура метода для асинхронной службы (`GWTServiceAsync`) должна соответствовать сигнатуре `GWTService` (но включать объект `AsyncCallback` как окончательный параметр). Поэтому удалите параметр `String` из `myMethod()`, чтобы интерфейс выглядел следующим образом.

```
public interface GWTServiceAsync {
    public void myMethod(AsyncCallback callback);
}
```

TIP Дополнительные сведения об интерфейсе асинхронной службы приведены в разделах [Выполнение асинхронных вызовов](#) и [Знакомство с асинхронными вызовами](#) официальной документации GWT.

- Выполните проект. При развертывании приложения и открытия браузера каждую секунду с сервера будет получаться новая цитата:

Sometimes even to live is an act of courage - Seneca

В следующем разделе будет применена таблица стилей для изменения внешнего вида цитат.

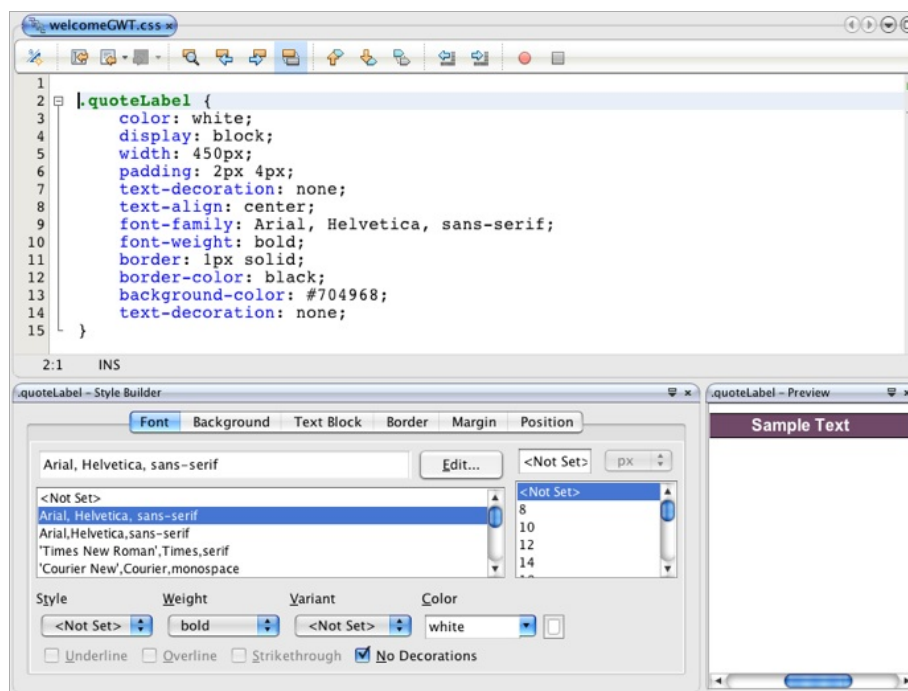
Настройка внешнего вида

В этой разделе к странице узла HTML прикрепляется таблица стилей. На нее также создается ссылка в классе точки входа. В частности, для названия стиля метки в классе точки входа необходимо установить название стиля в таблице стилей. Во время выполнения GWT соединяет стиль с меткой и отображает настроенную метку в браузере.

- Создайте таблицу стилей `welcomeGWT.css`. Для создания файла щелкните правой кнопкой мыши узел 'Веб-страницы' в окне 'Проекты' и выберите 'Создать' > 'Другие'. Появится мастер создания нового файла.
- В "Категории" выберите "Web", затем в "Типы файлов" выберите "Каскадная таблица стилей". По завершении работы мастера новый пустой файл откроется в редакторе.
- Создайте следующий селектор `quoteLabel` для новой таблицы стилей.

```
.quoteLabel {
    color: white;
    display: block;
    width: 450px;
    padding: 2px 4px;
    text-decoration: none;
    text-align: center;
    font-family: Arial, Helvetica, sans-serif;
    font-weight: bold;
    border: 1px solid;
    border-color: black;
    background-color: #704968;
    text-decoration: none;
}
```

Теперь в редакторе таблицы стилей должно отображаться следующее.



TIP Для отображения предварительного просмотра CSS и конструктора стилей выберите "Окно > Другое" в главном меню.

- На странице приветствия приложения создайте ссылку на таблицу стилей (**welcomeGWT.html**). Одновременно с этим добавьте некоторый текст, чтобы представить приложение пользователю. Новые части страницы HTML выделены ниже **полужирным шрифтом**.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
  <head>
    <meta name='gwt:module' content='org.yournamehere.Main=org.yournamehere.Main'>
    <link rel="stylesheet" type="text/css" href="welcomeGWT.css">

    <title>Main</title>
  </head>

  <body>
    <script language="javascript" src="org.yournamehere.Main/org.yournamehere.Main.nocache.js"></script>

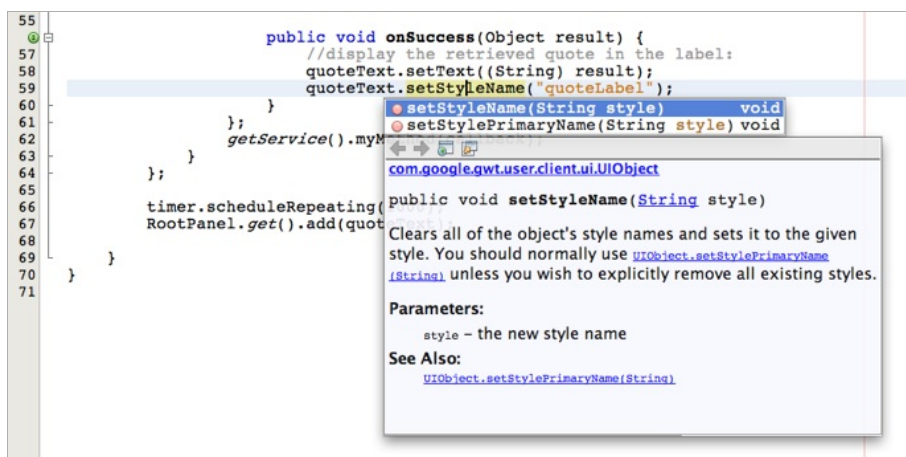
    <p>This is an AJAX application that retrieves a random quote from
    the Random Quote service every second. The data is retrieved
    and the quote updated without refreshing the page!</p>

  </body>
</html>
```

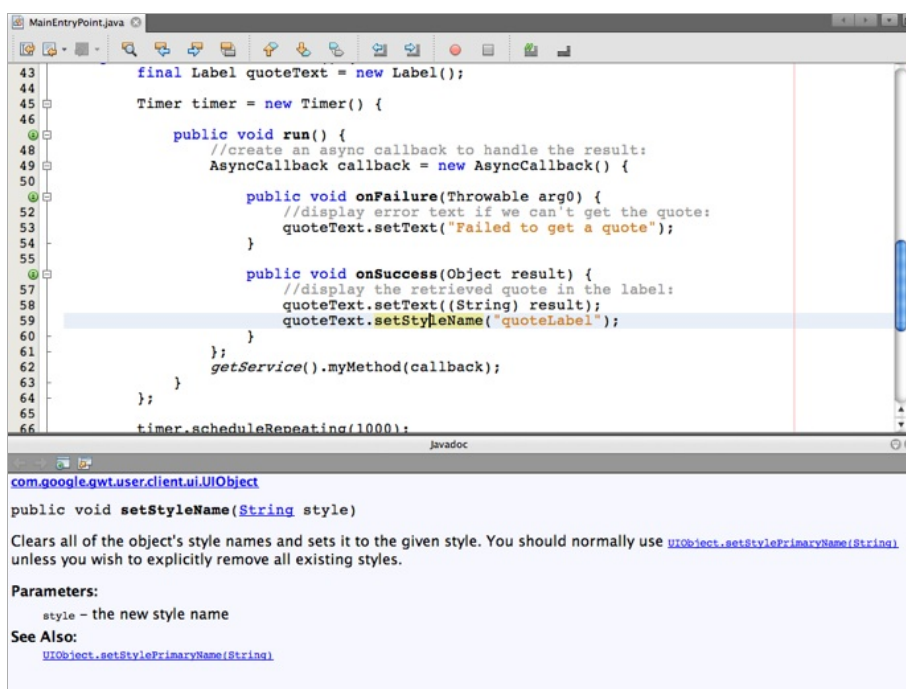
- В методе `onModuleLoad()` класса точки входа (`MainEntryPoint.java`) укажите, что при успешном выполнении стиль, определенный в таблице стилей, должен быть применен к метке. Новая строка выделена **полужирным шрифтом** ниже.

```
public void onSuccess(Object result) {
  //display the retrieved quote in the label:
  quoteText.setText((String) result);
  quoteText.setStyleName("quoteLabel");
}
```

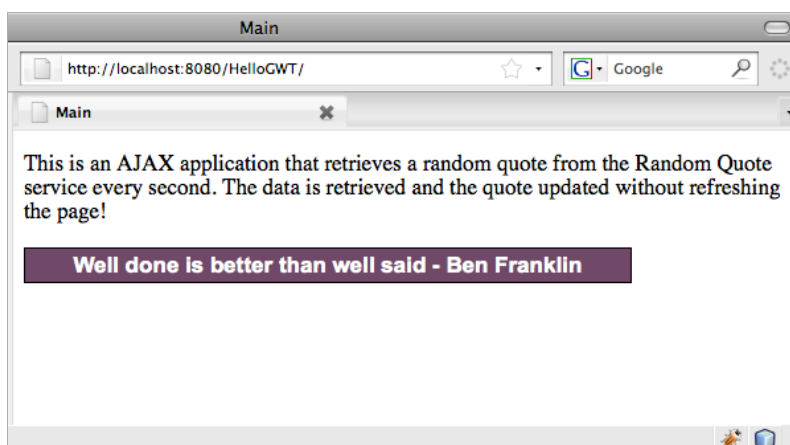
При вводе нажмите сочетание клавиш **Ctrl-ПРОБЕЛ** для включения функции автозавершения кода среды IDE. Функция автозавершения кода работает путем вызова всплывающего окна с предложениями по завершению кода и отображением связанной документации Javadoc.



TIP Также можно открыть окно "Документация" среды IDE, если требуется постоянный доступ к документации GWT при работе в среде IDE. Для этого выберите "Окно > Другое > Документация" в главном меню. Обратите внимание, что при вводе в редакторе документация в окне документации Javadoc обновляется в соответствии с положением курсора.

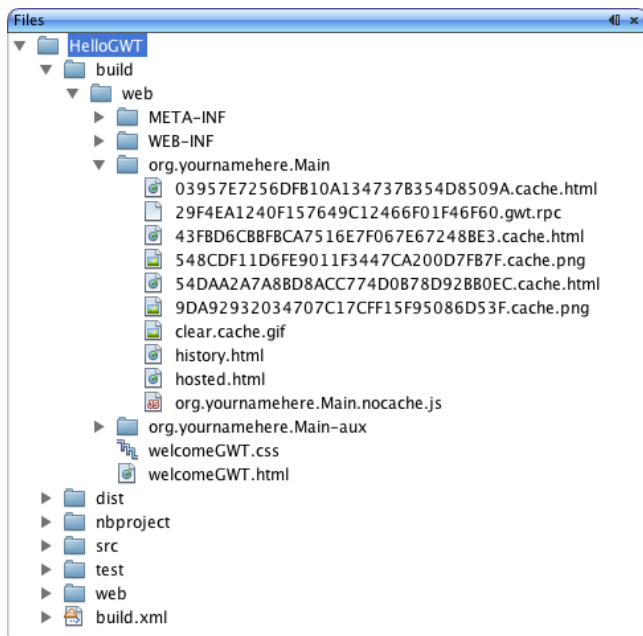


- В окне "Проекты" щелкните правой кнопкой мыши узел проекта и выберите команду "Выполнить". В этот раз метка отображается с пользовательским стилем, используя созданную в этом подразделе таблицу стилей.




Компиляция и отладка

Откройте окно "Файлы" (Ctrl-2; ⌘-2 в Mac) и разверните папку **build**. (Если папка **build** отсутствует, необходимо снова выполнить сборку проекта, чтобы среда IDE снова создала папку **build**.) Должно появиться следующее:

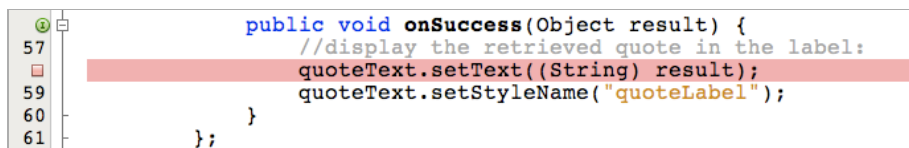



Эта папка создается GWT автоматически при компиляции приложения. Папка содержит готовую к развертыванию версию клиентского приложения. Описание того, что именно представляют эти файлы см. в разделе [Часто задаваемые вопросы по кодам Google - зачем нужны операции с кэшем и нелепые имена файлов?](#).

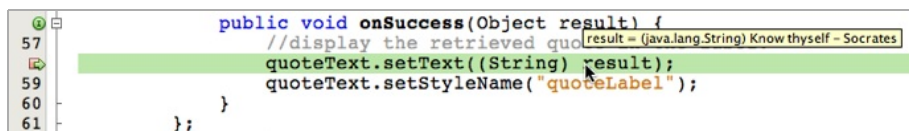
Также имейте в виду, что при работе с приложением GWT можно использовать встроенный [отладчик](#) среды IDE. Это позволяет выполнять отладку в [размещенном режиме GWT](#). Автоматически откроются GWT, главное окно основного режима и веб-браузер.


 **Примечания для пользователей для Mac OS X.** Размещенный режим GWT скомпилирован для 32-разрядной архитектуры в Mac OS X, которая существует только для Java 1.5. Если запущена 64-разрядная версия Java 1.6, необходимо переключиться на 32-разрядную версию. Для этого используется панели 'Предпочтения Java' в OS X. После переключения версий Java необходимо перезапустить среду IDE.

Установите точки останова поля, метода и строки в исходных файлах, щелкнув левое поле редактора IDE.

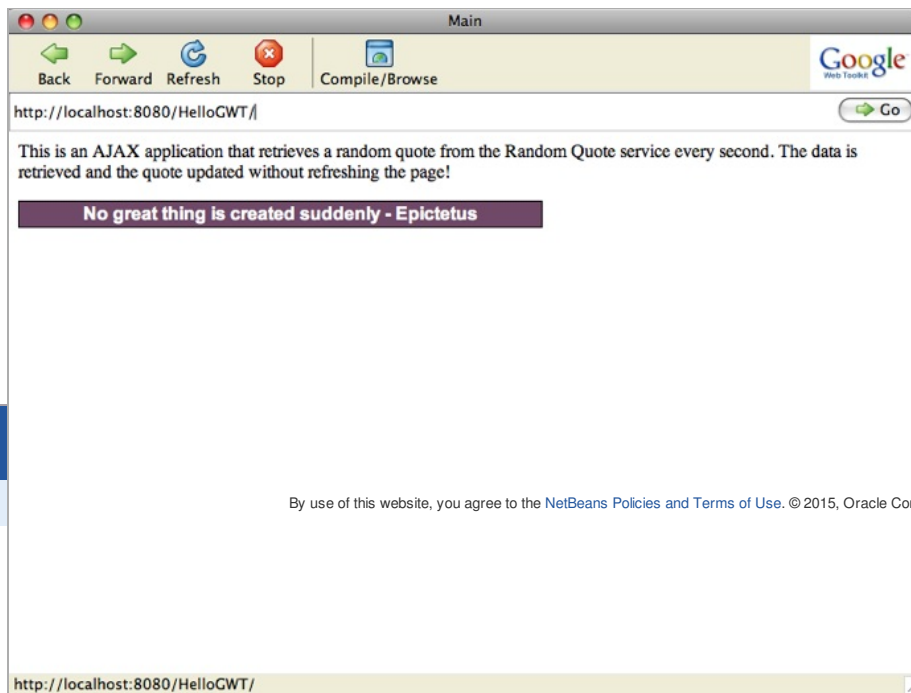


Затем просто выберите 'Отладка' как это обычно делается для любого веб-проекта (например, щелкните правой кнопкой узел проекта и выберите 'Отладка' или щелкните значок 'Отладка проекта') (). Выполнение приложения приостанавливается во всех установленных точках останова, позволяя выполнить код пошагово и проверить значения переменных и выражений (например, выберите "Окно > Отладка > Локальные переменные", чтобы просмотреть значения в окне "Локальные переменные").



 **ТIP** Также можно навести курсор мыши на выражение или значение в редакторе, после чего появится всплывающее окно с текущим значением (как показано на изображении выше).

Открытые главное окно размещенного режима GWT и веб-браузер. В браузере отображается работающая версия приложения.



Заключение

В этом учебном курсе вы изучили следующее.

- Как выглядит исходная структура обычного приложения в приложении Google Web Toolkit.
- Как соотносятся друг с другом артефакты Google Web Toolkit.
- Как настроить среду IDE на использование Google Web Toolkit.
- Какие инструменты доступны в среде IDE, в частности, для использования Google Web Toolkit.

Поскольку платформа GWT обрабатывает создание кода, связанное с браузером, а также создание кода API XMLHttpRequest нижнего уровня, можно использовать платформу для работы с функциями, которые должно предоставлять приложение. Поэтому как было указано во введении, GWT позволяет избежать трудностей обеспечения совместимости с браузерами, одновременно предоставляя пользователям динамичную соответствующую работу, обычно обеспечиваемую Web 2.0. Как было продемонстрировано в учебном курсе, можно применить платформу GWT для создания всей интерфейсной части на Java, поскольку известно, что компилятор GWT может преобразовать классы Java в код JavaScript и HTML, совместимый с браузером. Как также было продемонстрировано, среда IDE предоставляет полный набор инструментов для упрощения и повышения эффективности этого процесса без необходимости создания кода основной инфраструктуры приложения GWT вручную.

Мы ждем ваших отзывов

Дополнительные сведения

Это заключительный раздел введения в учебный курс по платформе Google Web Toolkit. Связанные и дополнительные материалы доступны в следующих источниках:

Ресурсы GWT

- [Руководство разработчика Google Web Toolkit](#)
- [Страница проекта NetBeans Google Web Toolkit](#)
- [Внутреннее устройство GWT4NB](#)
- [Блог по NetBeans Google Web Toolkit](#)

Документация NetBeans для платформ Java Web

- [Введение в JavaServer Faces 2.0](#)
- [Введение в платформу Spring](#)
- [Введение в веб-платформу Grails](#)
- [Введение в веб-платформу Grails](#)